Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE DISERTACIÓN PÚBLICA

No. 00028

CONTROL SOBRE EL PROCESO DE TRABAJO EN LOS TRABAJADORES COGNITIVOS. EL CASO DE LOS PROGRAMADORES DE SOFTWARE EN EL VALLE DE MEXICO.

UNIVERSIDAD AUTÓNOMA METROPOLITANA
DIRECCIÓN DE SISTEMAS ESCOLARES
Casa abierta al tiempo

J GUADALUPE RODRIGUEZ GUTIERREZ
ALUMNO

REVISÓ

LIC. JULIO CESAR DE LARA ISASSI
DIRECTOR DE SISTEMAS ESCOLARES

En México, D.F., se presentaron a las 11:00 horas del día 14 del mes de agosto del año 2008 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. SERGIO GUADALUPE SANCHEZ DIAZ
DR. ENRIQUE MODESTO DE LA GARZA TOLEDO
DRA. ANA MARIA MONICA CASALET RAVENNA

Bajo la Presidencia del primero y con carácter de Secretaria la última, se reunieron a la presentación de la Disertación Pública cuya denominación aparece al margen, para la obtención del grado de:

DOCTOR EN ESTUDIOS SOCIALES (ESTUDIOS LABORALES)

DE: J GUADALUPE RODRIGUEZ GUTIERREZ

y de acuerdo con el artículo 78 fracción IV del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

DIRECTOR DE LA DIVISIÓN DE CSH

DR. PEDRO CONSTANTINO SOLIS PEREZ

PRESIDENTE

DR. SERGIO GUADALUPE SANCHEZ DIAZ

VOCAL

DR. ENRIQUE MODESTO DE LA GARZA TOLEDO

SECRETARIA

DRA. ANA MARIA MONICA CASALET RAVENNA

Universidad Autónoma Metropolitana
Casa abierta al tiempo
División de Ciencias Sociales y Humanidades. Unidad Iztapalapa

Control sobre el proceso de trabajo en los trabajadores cognitivos. El caso de los programadores de software en el Valle de México

Tesis en opción al grado de

Doctor en Estudios Sociales,
línea Estudios Laborales

Presenta

J. Guadalupe Rodríguez Gutiérrez

Dirigida por

Dr. Enrique de la Garza Toledo

Agosto de 2008
México. D.F.

I N D I C E

Parte I I

Gobernabilidad y estrategia nacional en la creación
de capacidades internas y flujos de aprendizaje

Capitulo V

Configurando los espacios socio-técnicos
de una incipiente industria de software en México

Capitulo VI

Perfil socio-profesional de un trabajo en construcción.
El trabajador cognitivo, entre profesión, ocupación y empleo

Capitulo VII

Contextos externos y procesos internos que configuran flujos de
aprendizaje en el proceso de trabajo cognitivo en el valle de México

Parte III

Configuraciones simbólicas en el saber –hacer, interacciones sociales y prácticas embebidas entre juegos y consensos, negociaciones y conflictos, saber y poder

Capitulo VIII

El saber como poder y el saber-hacer como aprendizaje:
Juegos y consensos en el proceso de trabajo cognitivo

Capitulo IX

Subjetividad, estructuras y acciones en el proceso de trabajo:
Una reflexión desde la subjetividad del conflicto como resistencia

Agradecimientos

En Junio de 2004, la comisión evaluadora de admisión del Doctorado en Estudios Sociales, me envío por correo electrónico una pregunta: ¿Estaría usted dispuesto a cambiar y aprehender nuevas corrientes teóricas?. A lo cual conteste, con cierto aire de ventaja, que por supuesto. Para Junio de 2004, el tema de tesis era: *Empleo, flexibilidad y acción sindical en la industria maquiladora de Sonora.* Para Noviembre el tema de tesis se intitulaba: *Los trabajadores de cuello de silicio en la industria del software en la zona metropolitana del Distrito Federal.* En Septiembre de 2004 las herramientas metodológicas se remitían a las del enfoque económico; para Diciembre me proponía emplear metodologías de la Sociología.

El redefinir el tema de investigación, fue sin lugar a dudas gracias al ambiente de discusión que se establece entre alumno, tutor y comité de evaluación. Así como la amplia tradición de apertura y solidez académica del cuerpo docente del Doctorado en Estudios Sociales de la Universidad Autónoma Metropolitana, unidad Iztapalapa.

Estudiar bajo la dirección del Dr. Enrique de la Garza Toledo, definitivamente transformó la perspectiva analítica de un servidor, a través de reuniones de trabajo semanal que representaban un desafió constante y permanente y, una verdadera prueba de resistencia y paciencia para el Director; quien no sólo condujo a un servidor en el proceso de aprendizaje, también aconsejó y asesoró en una serie de aventuras extra académicas de las cuales, sólo citare dos: estancia de investigación en la Universidad Complutense en Madrid, España y, asesoría en la investigación laboral de la STPS-BID 2006 de la cual resulte beneficiado.

La estancia en el grupo de investigación "Charles Babbage", dirigido por el Dr. Juan Jose Castillo de la Universidad Complutense de Madrid, España, fue de una utilidad significativa para abordar los procesos "realmente existentes" en el "trabajo fluido".

Las discusiones y observaciones en el comité de evaluación en el postgrado, fueron de gran relevancia para los avances teóricos de la presente; sin menoscabo de todos los que comentaron la presente tesis, es preciso señalar los aportes del Dr. Sergio Sánchez como comentarista en los cuatro años del postgrado, las observaciones metodológicas de la Dra.

Alicia Lindón y Dr. Luís Reygadas; así como los comentarios puntuales del Dr. Carlos Salas y Dra. Marcela Hernández. Así como la de los compañeros de postgrado Marco Tulio; Juan Carlos Celis; Oscar Calderón; Virginia Martínez; Victoria Ramírez y Héctor Gaspar.

Las observaciones oportunas de la Dra. Mónica Casalet Ravenna, como sinodal, permitieron que la tesis reforzara algunas perspectivas importantes y se identificaran algunas líneas de investigación para futuros trabajos. También las contribuciones del Dr. Sergio Sánchez, como sinodal condujeron a un servidor a revisar nuevamente las corrientes clásicas de la Sociología del Trabajo. Así como las reflexiones finales del Dr. Enrique de la Garza, permitieron que esta tesis llegara a buen término. Sin embargo, es importante señalar que los conceptos y propuestas de la presente, son responsabilidad absoluta de quien escribe.

Es trascendental en este tipo de retos el aliento incondicional de la familia, de la siempre sensible compañera Lupita Gutiérrez; de Karol Rodríguez, que con un abrazo me despidió para que viviera en la ciudad de México, pero con la otra mano sujetaba al papa-entrenador; mi pequeña Sofía que antes de saber leer y escribir, aprendió lo que significaba "jugar a las escondidas por chat"y alegrarse mediante la cámara Web y los iconos de Internet para comunicarse con sus "papa virtual".

La estancia en la ciudad de México, fue placentera gracias a diferentes amigos, como el apoyo absoluto del Dr. Oscar Conteras M., Dra. Laura Velasco Ortiz y Oscar Contreras V. Así como el personal del postgrado de la UAMI que siempre estuvieron atentos, como son Lic. María Luisa Cortés, Rosario Brito y, Marisela González Zayas.

A los profesores y amigos de UNT

A la familia Rangel-Gutiérrez

Definición del problema y metodología

> *El software Windows Vista, con valor de 450 millones de dólares, con mas de 4 años de investigación y desarrollo por el equipo de programadores de Microsoft, salió al mercado en 2006, con meses de retraso, con numerosos "hoyos" de seguridad, fallas graves de interoperabilidad, incluido errores de compatibilidad entre software interno de Microsoft... (http://barrapunto.com acceso 6.10.07)*

Introducción

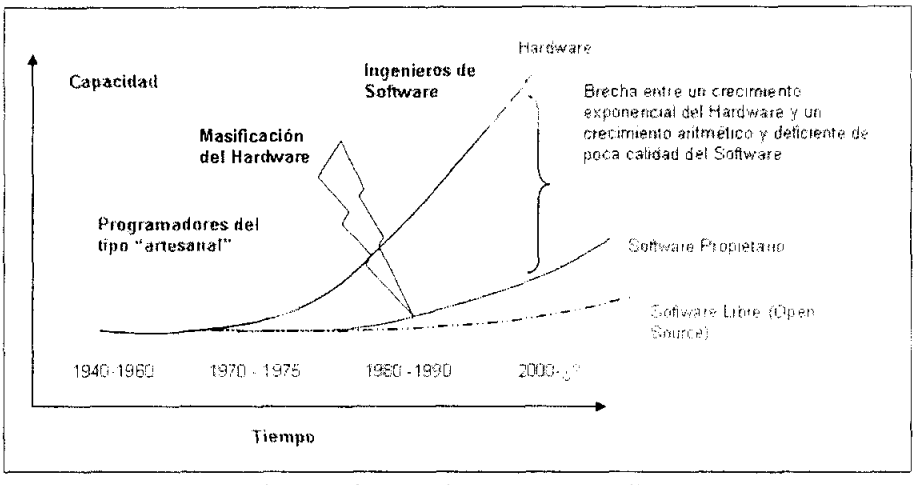
La producción de software no es nueva, nació en 1940 junto con la primera computadora en Estados Unidos (Steinmuller, citado por Salomón, 2000). Los primeros proyectos de los programas informáticos de hardware y software estaban financiados por el área militar norteamericana, demostrando su eficiencia en el periodo entreguerras, principalmente en el diseño de aviones, misiles, decodificación de mensajes enviados en claves, etc. Sin embargo hacia finales de la década de los cincuentas hacen presencia las primeras empresas de software y, una década después se hace más frecuente el vocablo software. Entre los sesenta y mediados se los setenta aparecen las primeras empresas informáticas que integran gratuitamente el software al hardware (*software embebido*). En estos mismos años, en Estados Unidos, la empresa informática IBM vende su primer programa de software por separado del hardware; surgen las primeras casas de software como SEMA en Francia, CAP en Inglaterra, entre otras empresas que guiarán la división del software con respecto al hardware. Segmentación que es posible gracias a cuatro procesos coyunturales que se sucedieron hacia principios de la década de los setenta: i) masificación de las primeras computadoras personales (IBM), por tanto mayor socialización del conocimiento hardware/software e impulso del software empaquetado o empotrado, lo que se traduce en una mayor masa critica y fluidez en el conocimiento; ii) desarrollo de lenguajes de programación de alto nivel como cobol (1960) y fortran (1957), éstos lenguajes de programación facilitaron un mayor acceso de programadores, que el antecesor lenguaje ensamblador; iii) mayor potencia en procesadores, miniaturización y menor precio en los microprocesadores 4004, 8008, 80880, 1971); iv) creación del sistema operativo (MS-DOS), es el "eslabón" entre hardware y software. En otras palabras, hacia mediados de los setentas y ochentas convergen dos contextos estructurales que dan lugar al desarrollo de la industria del software, por un lado una trayectoria tecnológica de hardware cada vez más potente, basado en microprocesadores veloces, pequeños y baratos, y por otro, un proceso acelerado en el uso de nuevas tecnologías como el Internet, comercio electrónico, desarrollo de páginas Web, etc. Ambos procesos de trabajo demandaban una

nueva clase de trabajadores informáticos, como son los desarrolladores de páginas Web, programadores de software, diseñadores de sitios punto.com, etc.; es decir nuevos trabajos y trabajadores; nuevos empleadores y consumidores de servicios informáticos que están inmersos en un nuevo contexto laboral, que denominamos *trabajadores cognitivos* que configuran nuevas formas de trabajo que operan símbolos inmateriales, más allá de la manipulación de productos físicos que integran el hardware, etc. Los sistemas informáticos como expresión "dura" del hardware alcanza el punto más alto en el consumo masivo hacia inicios de los ochenta en Estados Unidos, superando las ventas de 1981 que fueron de 800 mil unidades de computadoras, frente a 60 millones en 1987. El crecimiento exponencial no sólo dinamizó el sector en su conjunto, sino que dio origen a la denominada revolución tecnológica que tiene como corolario productos y procesos que no existían hace diez años. Hacia fines del siglo XX, C. Pérez (2007) denominó como "la era de la conectividad" al desarrollo de nuevas tecnologías basadas en los sistemas informáticos.

El software embebido o empaquetado, hacia las décadas de 1940 y 1960 se suministraba gratuitamente en el hardware, formaba parte del producto, se distribuía como un añadido más, sin embargo hacía mediados de los sesenta y principios de los setenta, se genero, gradualmente un mercado de software independiente del hardware, que ha mas de 35 años, se ha constituido una brecha importante (ver grafica 1), entre un crecimiento exponencial del hardware y un desarrollo aritmético de software, brecha que se explica en parte por el desarrollo muy potente de hardware que posee "capacidad ociosa instalada", debido a que no existe un software que maximice su uso y, por otro lado, la demanda de mayor capacidad de hardware frente a una limitada oferta de software. En 1999 la brecha en el mercado mundial era 29% para software y 71% para hardware (Chudnovsky et.al. 2001). Además de la brecha entre hardware y software, en la década de los noventas cobró fuerza un debate en torno al modo de desarrollar los programas informáticos: el modo privado y el modo libre. El primero hace referencia a la apropiación del conocimiento a través de las patentes y licencias de uso, el proceso de trabajo es una especie de "caja negra" para el cliente/usuario final, en el software privado el consumidor no tiene acceso al código fuente, no posee la "licencia" de modificar el programa, sino que debe recurrir al desarrollador para que éste le otorgue mantenimiento y actualización al software que adquirió previamente. En cambio en el modo del software libre (*open source*), el consumidor tiene acceso al código que integra el software adquirido

(licencias creative commons)[1], el cliente o consumidor dispone del código para solicitar a otros proveedores el mantenimiento o actualización del mismo, no paga licencias de uso y, no está supeditado al proveedor (ver gráfica1).

Gráfica 1
Brecha entre hardware y software



Elaboración propia en base a Chudnovski et.al. 2001 y Tikai:2003

La industria de software, según señala Torrisi (1998, citado por Chudnovski et.al. :2001:4), es una actividad relacionada con la codificación del conocimiento y la información, siendo sus inputs y outputs propiamente dichos inmateriales (Torrisi, 1998). Según la forma en que se proveen, dichos outputs pueden considerarse como productos o servicios. El software como producto, según Chudnovski, et.al. (2001) y Tikai (2003), entre otros, es la venta de licencias y patentes, para su uso dentro de una organización a nivel individual. En algunos casos, la firma desarrolladora provee algún tipo de servicio asociado al software (actualización de las versiones, soporte técnico, mantenimiento, etc.) que puede estar incluido dentro del contrato de licencia o comercializarse de manera independiente. Para Hoch et. al. (1999, citado por Chudnovsky et.al. 2001:4), es posible dividir el segmento de productos de software en dos grupos: como soluciones empresariales a la medida y como productos empaquetados de mercado masivo. La distinción entre ambos va más allá del mercado al cual se dirigen (en este sentido, un procesador de texto, por ejemplo, puede comprender tanto al mercado empresarial como al hogar). Por ejemplo, la diferencia sustancial entre un producto de mercado masivo y una solución a la medida, radica en que esta última siempre exige, de acuerdo a su complejidad, algún grado de personalización o adaptación a los requerimientos específicos de

[1] Véase http://creativecommons.org/licenses/by-nc-nd/2.5/es/legalcode.es Acceso 16 Junio de 2006

la organización en la cual va a ser implementada. En este último caso la "puesta en marcha" de la aplicación (es decir, la instalación y los ajustes necesarios para su correcto funcionamiento) suele implicar una inversión importante en términos de tiempo y dinero (verificar compatibilidad e interoperatividad, ya sea con el hardware ó software)[2].

El software como servicio, según Chudnovski et.al. (2001) y Tikai (2003), se define como los ingresos generados por servicios que provienen de actividades tan diversas como el diseño y desarrollo de soluciones a la medida, implementación y adaptación de productos de terceros, servicios de consultoría, capacitación, instalación y mantenimiento de productos de software, etc. Sin embargo, diferenciar ambos tipos es cada vez más difícil, ya que su frontera es muy difusa. Por ejemplo los Enterprise Resources Planning (ERP) son los programas de software industriales más difundidos entre el sector empresarial, en ellos se mezclan ambos procesos, donde el 30% del costo corresponde al producto como pago de licencia y 70% a los servicios profesionales. El número de licencias otorgadas podría ser una medida del desempeño para una empresa de productos, mientras que en el caso de una empresa de servicios la cantidad de horas de implementación asociadas a cada proyecto sería el indicador más relevante. Sin embargo, en la presente investigación partimos de la definición de software aceptada comúnmente, que es aquella señalada por los organismos internacionales como el Software Engineering Institute (SEI), el Internacional Standardisation Organisation (ISO), el World Intellectual Property Organisation, (WIPO) que lo definen como:

> "producción de un conjunto estructurado de instrucciones, procedimientos, programas, reglas y documentación contenida en distintos tipos de soporte físico (cinta, discos, circuitos eléctricos, etc.) con el objetivo de hacer posible el uso de equipos de procesamiento electrónico de datos".

En cuanto al aspecto de la eficiencia, productividad y calidad, para muchos investigadores e ingenieros de la industria del software éste tema se ha convertido en un conjunto de incertidumbres e equívocos, que se traduce en continuos problemas de calidad, incumplimiento de funciones, desbordamiento de tiempos y costos estimados, no observancia

---

[2] Compatibilidad: es la condición que hace que un programa y un sistema, arquitectura o aplicación logren operar correctamente tanto directamente, como indirectamente (mediante un algoritmo), a éste algoritmo se le denomina emulador, por el hecho de que es un intérprete entre el programa y el sistema, arquitectura o aplicación. Interoperatividad: Es la condición mediante la cual los sistemas heterogéneos pueden intercambiar procesos o datos. Por ejemplo en el campo de la informática, se habla de la interoperatividad de la Web como condición necesaria para que lo usuarios (humanos, hardware o softeware) tengan un acceso completo a la información disponible. http://es.wikipedia.org Acceso 28 de Julio de 2007

de requisitos solicitados por el cliente, ineficiente control de errores y fallas en el sistema, etc. El conjunto de problemas de eficiencia, calidad y productividad en la industria del software es tema amplio que ha sido abordado en la literatura bajo el concepto *crisis del Software* (Glass:1997; Flowers:1997; Yourdon: 1998, entre otros); sin embargo también es cierto que es ineludible hablar de un desarrollo de software competente a nivel global es decir software que ha tenido éxito, hecho que ha llevado a diversos autores como Glass, R. (1998) a señalar que debería replantearse la postura con respecto al concepto de *crisis del software,* al señalar que existen muchos fallos importantes en medio de muchos éxitos. Para Pressman R. (2002) el concepto de *crisis del software* no es tan exacto, al respecto propone que en términos de la tasa de cambio en la calidad y velocidad con la cual se desarrollan nuevos productos y sistemas informáticos, no se ha llegado aún a un "punto crucial" o "decisivo" en el que se plantee la "muerte del software" como consecuencia de la "crisis", por el contrario sólo ha habido un lento cambio, contextualizado por cambios tecnológicos explosivos en las disciplinas relacionadas con el desarrollo de software, como son, la Ingeniería del Software, Ingeniería de la Usabilidad, Ingeniería de Requisitos, entre otras. Pressman (2002) plantea que en lugar de *crisis del software* se utilice el concepto *Aflicción crónica del software,* propuesto por Daniel Tiechrow, de la Universidad de Michigan en 1979[3]; el concepto de «aflicción» hace referencia a "algo que causa pena o desastre" y «crónica» a "muy duradero" o que "reaparece con frecuencia, continuando indefinidamente" este último término es el que más se ajusta a lo que acontece en la industria del software. El termino «aflicción crónica» se adecua al conjunto de fallos y errores en el desarrollo de un programa informático[4]. Existen variadas anécdotas relacionadas con la *aflicción del software,* paralelas a los errores del hardware:

- El 9 de Noviembre de 1979 un sistema militar norteamericano de monitorización confundió la simulación de un ataque, con un ataque real de la Unión Soviética.
- La explosión del Arianne 5, el 19 de Julio de 1996, después de 10 años de desarrollo y una inversión de $7 mil millones de dólares. La causa fue un fallo en un programa que intentaba guardar un número de 64 bits en un espacio de 16 bits.

[3] conferencia en Ginebra, Abril de 1989, citado por Pressman: 2002:14. op.cit.

[4] De aquí en adelante sólo utilizare el termino "aflicción del Software" para hacer referencia al conjunto de incertidumbres que rodea el desarrollo de programas informáticos. Utilizare indistintamente programas informáticos y software, como sinónimos, a menos que se indique lo contrario.

- El ministerio de defensa del Reino Unido en el 2000, canceló un proyecto de guerra electrónica que ya había consumido $5 millones de libras, se retrasó más de 5 años y que no satisfizo sus expectativas.
- En 1995, se estimaba que las compañías y el gobierno de Estados Unidos se gastaron $81,000 millones de dólares en proyectos cancelados. Los proyectos terminados, pero cuyo plazo de ejecución fue superior supusieron un coste de $59.000 millones de dólares. Se estimo que sólo 16.7% de los proyectos de software fueron terminados en el plazo y presupuesto previstos.

La definición de *aflicción del Software* es más precisa que *crisis del software*. Igualmente se desarrolla software eficiente, por grandes empresas globales como Microsoft, IBM, Oracle, etc., como por pequeñas empresas; sin que ello signifique que todo el software desarrollado tiene errores, o que se está en crisis de producción al estilo de la manufactura. Consideramos importante, comprender éstos desconocidos fenómenos productivos para crear nuevos conceptos que nos permitan explicar la paradoja productiva del relativo éxito del software frente a la aflicción crónica del software. Debemos construir conceptos que nos permitan explicar el contenido inmaterial de códigos y algoritmos que forman parte central de la producción simbólica del software y el acelerado cambio técnico que dan forma a las nuevas tecnologías de la información y la comunicación; es decir, aún con errores y fallas en los programas de software, el hardware que constituye las nuevas tecnologías continúan operando. Dichos errores no son nuevos, ya hacia fines de los sesenta se cuestionaba la eficiencia del desarrollo de software, primero en 1968 (Garmisch,) y después en 1969 (Roma) se realizaron debates internacionales para abordar la incertidumbre que encierra el desarrollo de software (que básicamente son de naturaleza similar a los de hoy día): retrasos en los plazos de entrega, mayores costos a los estimados inicialmente, fallas, errores, defectos en el programa, incumplimiento en requisitos, etc. En estas conferencias organizadas por la NATO[5] concluían que la complejidad de los programas informáticos se derivan de la naturaleza abstracta del software, compuesta por sistemas discretos basados en lógica matemática, algoritmos complejos representados en lenguajes de programación que sólo los que

[5] Información de conferencias http://homepages.cs.ncl.ac.uk/brian.randell/NATO/. Acceso 15 Julio de 2007. Véase: Naur, P., y B. Randall (eds), 1969, Software Engineering: A report on a Conference sponsored by the NATO Scince Committee, NATO.

intervienen en el desarrollo del software están familiarizados en la construcción lógica de los códigos (Schoen:1992). El contexto en el cual debe analizarse la *aflicción del software* debe comprender aspectos como eficiencia algorítmica, estructuración del proceso y requerimientos, así como el esfuerzo cognitivo por parte del programador para comprender los problemas del usuario y traducirlos en términos lógicos, y diseñar-codificar-computar las líneas de código que den por resultado algoritmos que representa la posible solución a los requerimientos expuestos por el usuario (Schoen:1992)[6]. Al respecto diversos especialistas en Ingeniería del Software, como Gonzalo A. (1991); Mark, N. y P. Rigby (1994); García R. y Sánchez, S. (1993); entre otros expertos en el tema han señalado la existencia de una serie de aproximaciones teórico-practicas relativas a diseñar formalismos metodológicos que den por resultado métricas de programación estructurada, metodologías orientada a objetos, procesos de calidad, reutilización de programas modulares, ciclo de vida con énfasis en calidad, entornos de desarrollo tipo red (programación en red), etc.; sin embargo ninguna de estas propuestas en las últimas tres décadas han sido suficientemente eficientes como para aportar una solución a la *aflicción del software*. Aún no se ha esbozado "un único mejor método" en el desarrollo de software, hasta el momento es imposible unificar en un sólo paradigma o método las inagotables singularidades y excepciones que constituyen la norma habitual de esta disciplina.

[6] Para una mayor información véase la siguiente liga www.ics.uci.edu Acceso 2 de Mayo de 2006

Planteamiento del problema, preguntas de investigación y metodología

Hacia fines de los noventa, los errores en programación cobraron relevancia mundial, debido a las voces de alarma por parte de los gobiernos, académicos, industriales y todos aquellos que estaban en contacto con los medios informáticos: El error del milenio (millennium bug) denominado Y2K[7], generaba incertidumbre y un alto riesgo en la credibilidad en los sistemas informáticos, sin embargo, este mismo error ha servido de empuje en la investigación y desarrollo de mejoras en los proceso para generar software. Este error en la programación forma parte habitual de un conjunto de incertidumbres, fallas y errores que entraña el desarrollo mismo del software. Este conjunto de errores ha sido denominado *aflicción del Software* tema que no es asunto coyuntural o de moda, es un problema trascendental que pretenden resolver las nuevas disciplinas como son la Ingeniería del Software, Arquitectura del Software, Ingeniería del Diseño, Ingeniería de Requisitos, entre otras ciencias que plantean distintos métodos y herramientas formales para administrar eficientemente el proceso de desarrollo de software; también significó un replanteamiento del papel de la administración, la gerencia, cuerpo técnico y trabajadores con respecto a las formas y procesos en que fluye el conocimiento, la transformación de la información en conocimiento y los procesos de aprendizaje. El conjunto de ingenierías supone la moderación de la *aflicción del Software* a través de resolver los problemas en administración de la calidad, productividad, diseño, mantenimiento y actualización en el desarrollo de software. Paralelamente, éstas ingenierías presionan el proceso de trabajo a través de un conjunto de normas y herramientas que cobran forma en métricas de calidad y metodologías en los procesos, así como la implementación de un conjunto de técnicas estandarizadas, que suponen un enfoque integral del problema; abarcando todas las fases de programación, que en su mayoría no se consideraba en los desarrollos tradicionales (etapa artesanal). El conjuntote normas y herramientas metodológicas constituyen el cuerpo teórico de la Ingeniería del Software, que puede definirse como *"el tratamiento sistemático de todas las fases del ciclo de vida del software"*, cuya tesis central es

[7] El problema del año 2000, Y2K, es un bug o error de software causado por la costumbre que habían adoptado los programadores de omitir el año para el almacenamiento de fechas (generalmente para economizar memoria), asumiendo que el software sólo funcionaría durante los años cuyos nombres comenzaran con 19. Lo anterior tendría como consecuencia que después del 31 de diciembre de 1999, sería el 1 de enero de 1900 en vez de 1 de enero de 2000. La corrección del problema costó miles de millones de dólares en el mundo entero, sin contar otros costes relacionados. Para una mayor información de cómo afecto el Y2K véase: Yourdon E. y J. Yourdon: 1998, Time Bomb 2000, Prentice-Hall; De Jager, P. et.al. : 1998, Countdown Y2K: Business Survival Planning for the year 2000, Wiley; Karlson, E., y J. Kolber, A basic Introduction to Y2K: How the year 2000 Computer Crisis Affects You?, Net era publication, Inc.

la reducción de costes, cumplimiento en tiempos y mejora en la eficacia del producto final (Pressman:2002). Sin embargo aún con la implementación de diversas metodologías de quinta generación en el siglo XXI, la *aflicción del software* continua presente, tal es el caso del fracaso de Windows Vista en programación, el software implementado en Boeing en aviación, etc., entre otros programas de software con errores y fallas; por tanto, el conjunto persistente de errores plantea una serie de preguntas: ¿Cuáles son las causas de la *aflicción del software*?; ¿Cuáles son los orígenes de la falta de calidad, eficiencia, productividad y cumplimiento de requerimientos en un programa de software?; ¿La *aflicción del software* tiene relación con las dificultades de la gerencia para controlar tiempos y proceso de calidad?; ¿La gerencia en que sentido y magnitud participa en el flujo de conocimiento del *cómo se resuelven* los problemas planteados por el usuario al solicitar un software?. En otras palabras la *aflicción del software*, ¿Podría conducir a una forma primigenia de taylorización en la industria del software, con el fin de minimizar los efectos de los errores y fallas en el desarrollo de software?; ¿Estaremos en presencia de la configuración de una "mano invisible de la administración" que pretende intervenir en el proceso de trabajo?; ¿La gerencia se interesa en construir un "único mejor camino" para *resolver* los problemas que se presentan en el proceso de trabajo?.

En esta investigación solo trataremos la pertinencia de la hipótesis de que *la aflicción del software* influye la forma del proceso de producción del conocimiento y el control en el saber hacer que el programador posee con respecto a su trabajo, sin entrar a analizar otros factores internos o externos que pueden estar influyendo. El interés se centra en los posibles mecanismos de generación, transmisión y contradicciones en la creación de conocimiento en el proceso de trabajo del software a la medida. En el desarrollo de software a la medida la participación del cliente es sustancial para definir los requerimientos del programa, en este sentido surgen otras preguntas: ¿Hasta que grado participa el cliente en la *aflicción del Software*?: ¿El cliente tiene claro los requerimientos del programa?; ¿Qué significado tiene para el proceso de trabajo, costos y tiempos de entregar, si el cliente modifica o incorpora nuevos requerimientos?; ¿En caso de modificarse algunos módulos, una vez ya iniciado el proyecto, que implica en términos de problemas de interoperabilidad?; ¿Se incrementan las probabilidades de fallas y errores?. Los trabajadores-programadores resuelven los requerimientos planteados por el cliente/usuario a través de un conjunto de símbolos denominados lenguajes de programación, por medio de los cuales crea un conjunto de rutinas

algorítmicas que representan una serie de conocimientos tácitos que no necesariamente pueden formalizarse en el proceso de trabajo, al respecto nos preguntamos ¿Cuáles son los límites en el control del proceso de trabajo por parte de los programadores y gestores de proyectos y, como se traducen éstos límites en la aflicción del software?; Los empresarios consideran a los programadores como "componentes generadores de código" en este sentido, ¿Qué significado tiene para el proceso de trabajo éste énfasis?; La capacidad de saber resolver determinado problema de programación ¿Qué tanto influye la acumulación de experiencia y habilidades informales o bien, en que grado depende más de las destrezas cognitivas aprendidas formalmente?; ¿Será posible estandarizar la producción de software a la medida como un siglo antes sucedió con la producción manufacturera?.

Para resolver parte de éstos problemas, la Ingeniería del Software emprende todo un conjunto de herramientas metodológicas y normas de calidad en el desarrollo de programas informáticos, como es la programación asistida por computadora (CASE: Computer Arded Software Engineering) o bien normas de calidad como CMM (Capabilty Maturity Model) y UML (Unified Modelling Language), entre otros, cuyo fin es establecer normas de calidad para minimizar errores y fallas en el desarrollo de software. En este sentido preguntamos ¿Estamos ante una forma primigenia de taylorización del proceso de trabajo en la programación de software a la medida?; ¿Significa el empleo de la "mano visible de la administración" por parte de la gerencia para diseñar y planificar programas de software a la medida?; ¿Es posible aplicar un estándar, una métrica, una metodología desde la gerencia?, ¿Qué resistencia y conflictos, contradicciones y consentimientos se provocan en el proceso de trabajo del programador?; ¿Qué réplicas formales e informales, intrínsecas y extrínsecas se generan en el proceso de trabajo, cuando la heterogeneidad de atributos del software es una condición común?; ¿Qué expresiones objetivas y subjetivas de control, poder y resistencia se generan en un proceso de trabajo altamente creativo, donde se reflexionan los movimientos y se estiman los tiempos de creación en la medida que se interactúa con el cliente?.

En la industria del software convergen heterogéneas formas de organización industrial para desarrollar software, desde las fábricas de software, pasando por las empresas que desarrollan software embebido (software añadido en el hardware), hasta las microempresas que desarrollan software a la medida en pequeña escala; poseen diversas formas de organización, no es el mismo tipo de software desarrollado, no es el mismo tipo de clientes, el

contexto local y de interacción con el sistema educativo, político y empresarial es distinto. El caso que analizaremos en la presente investigación, tiene que ver con el software hecho a las necesidades del cliente, y se parte de dos preguntas: ¿Cómo se sucede la comunicación y coordinación de la información entre el cliente, gestor del proyecto (gerente) y desarrollador de software?; ¿Cómo se resuelven los requerimientos y/o necesidades del cliente y, que significados/representaciones implica para el programador que resuelve?. Por otro lado, los proyectos de software a la medida tienden comúnmente a organizarse en equipos de trabajo, para lo cual indagaremos: ¿Cómo se dividen las responsabilidades de cada programador, quien las otorga y porque?; ¿Cómo se mide la eficiencia de los programadores de Software?; ¿En caso de no cumplirse con las metas, tiempos y/o requerimientos establecidos, quien aplica las sanciones y como las establece?; ¿Existe resistencia a la implementación de herramientas metodológicas y normas de calidad por parte de los programadores?; ¿Cómo influye el grado de preparación formal en el "estatus" social y salarial del programador?; ¿Como influyen las destrezas y habilidades personales en el proceso de trabajo?; ¿Cómo se definen las jornadas laborales, horas extras y bonos en el proceso de trabajo, si resolver el problema puede no necesariamente hacerse dentro del horario de trabajo o bien resolverse fuera de la oficina?.

El trabajo de campo llevado a cabo, se tuvieron que superar una serie de obstáculos, los primeros fueron de aproximación al objeto de estudio; los segundos -de carácter metodológico- para plantear el problema de investigación y un tercero fueron las estrategias de campo. Con respecto a la aproximación del objeto de estudio, surgieron varias dificultades. La primera, corresponde a que se trata de un problema de investigación no tradicional, la Ingeniería del Software no sólo es una profesión reciente y, los conceptos que le integran adquieren significados diferentes a los de la manufactura. Además, la Ingeniería del Software es el punto de partida de diversas profesiones como la Ingeniería de Requisitos, Ingeniería de la Usabilidad, Ergonomía del Software, Arquitectura del Software, entre otras.

Las estrategias que se implementaron para penetrar al objeto de estudio fueron variadas: asistir a clases de ingeniería, asistir como ponente y congresista a congresos y seminarios sobre software; diseño de un guión de entrevista semiestructurada, lecturas sobre Ingeniería de Software y de trabajos originales sobre el tema. Estas tácticas se traslaparon, en un periodo de tiempo que inicio en Febrero de 2005, y culmino hacia mediados de Julio de 2006. En lo relativo a la asistencia de sesiones de Ingeniería de Software, se acudió a ocho

clases en la escuela de Ingeniería de Sistemas en la Universidad de Sonora, en Hermosillo, Sonora, en el mes de Mayo de 2005. Se asistió a dos sesiones de programación orientada a objetos en la Maestría en Administración de Software en la Fundación Rosenbloot, a cargo del Maestro Martín Zavala en Octubre de 2005. Dichas sesiones tuvo como objetivo establecer empatía con contactos claves, además se realizaron entrevistas de exploración y acercamiento al objeto de estudio. Con respecto a la presentación de ponencias y asistencia congresos y seminarios de software y temas a fines, destacan los siguientes:

- Congreso nacional de software libre (CONSOL 2005), celebrado en la Universidad Autónoma Metropolitana, del 22 al 25 de Febrero del 2005, México, Distrito Federal. Ponencia: "Hacia una problematización de las relaciones laborales en la ingeniería del software en México. Una perspectiva desde la Sociología del Trabajo"

- *Encuentro Internacional de Educación Superior UNAM Virtual Educa 2005.* Organizado por la Dirección General de Computo de la UNAM; Coordinación de Universidad Abierta y a Distancia de la UNAM, (México); Secretaría de Cooperación Iberoamericana (SECIB- España) y la Organización de Estados Americanos (OEA). http://www.virtualeduca2005.unam.mx Realizado en México, Distrito Federal; del 20 al 24 de Junio del 2005.

- Jornadas Internacionales: *Sociedad y Economía del Conocimiento*. Convocado por la Benemérita Universidad de Puebla (México), Consejo Latinoamericano de Ciencias Sociales (CLACSO); Fundación Iberoamericana del Conocimiento (España); Universidad de Ciencias aplicadas de Wiesbaden (Alemania) y la Universidad de San Martín de Porres, (Perú). Puebla, Puebla; del 21 al 28 de Febrero de 2006.

- Primer Congreso Iberoamericano de Ciencia, Tecnología, Sociedad e Innovación. Organizado por la Organización de Estados Iberoamericanos para la Educación, la Ciencia y la Cultura (OEI), la Agencia española de Cooperación Internacional (AECI) y la UNAM-México. http://www.oei.es/congresoctsi/index.html .México, D.F. del 19 al 23 de Junio de 2006. Ponencia: Aprendizaje, poder y cultura laboral en los programadores de software.

- Segundo Annual Gartner Outsourcing Summit. Minimice el riesgo y maximice el beneficio de sus servicios terciarizados de tecnologías de la Información. http://www.gartner.com/2_events/conferences/2006/mex23l/mex23l.html Convocado por Gartner México. México, Distrito Federal; 27 y 28 de Junio de 2006. Asistencia financiada por la empresa Heurística Sistemas S.A. C.V.

- Congreso Nacional de Software Libre 2006. http://www.consol.org.mx/2006/ Organizado por el Instituto Politécnico Nacional (IPN) y comunidades virtuales de Software Libre de México, América Latina y Europa. Realizado en México, Distrito Federal; del 15 al 18 de Agosto de 2006.

- Encuentro Nacional PROSOFT 2006. Impulsando la Economía Mexicana a través de las Tecnologías de Información. http://www.software.net.mx/evento2006 Organizado por la Secretaría de Economía y Software.Net. México, Distrito Federal; el 28 y 29 de Agosto de 2006.

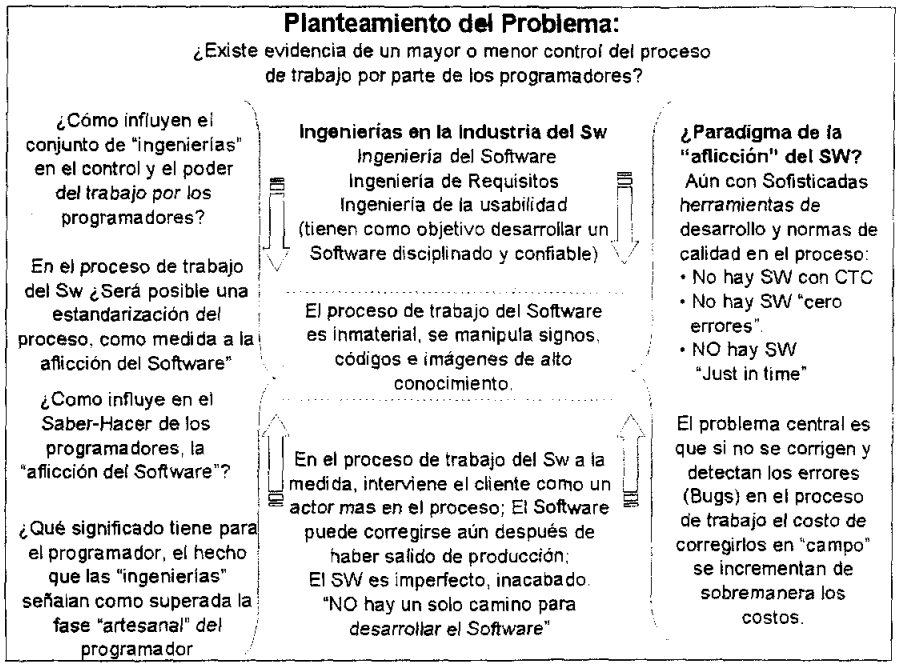
Con el fin de comprender mejor los conceptos de la denominada sociedad de la información y el conocimiento se participó en: Cátedra ALCUE: Sociedad del Conocimiento. Diciembre de 2005 a Agosto de 2006 http://www.flacso.edu.mx/csc.shtml; El Espacio común para la enseñanza superior América Latina, El Caribe y la Unión Europea (ALCUE, por sus siglas en ingles), fue impartido por la Facultad Latinoamericana de las Ciencias Sociales Unidad México (FLACSO). Inscripción financiada en parte por el postgrado en Estudios Sociales de la UAM y la Universidad de Sonora.

Paralelamente, se procedió a realizar entrevistas no estructuradas con respecto a la temática: la *aflicción del software*. Se realizaron veinticuatro entrevistas exploratorias, de las cuales catorce fueron entre Febrero y Junio del 2005, se aplicaron a tres directivos de la industria del software, ocho programadores y tres académicos en el Distrito Federal, así como a siete programadores de software y tres académicos de la Universidad de Sonora en Hermosillo, Sonora, entre Mayo y Junio de 2005. Al par que se aproximaba al objeto de estudio a través de las entrevistas exploratorias, se solicitó a diversos colegas que estaban estudiando el tema del software entre 2005 y principios de 2006, que facilitaran información nueva del tema; al respecto cabe mencionar que quienes tuvieron la amabilidad de compartir información inédita fue el equipo de trabajo del Dr. Gabriel Yoguel y los investigadores José Borillo y Verónica Roberth, todos de la Universidad de General Sarmiento en Argentina (Mayo de 2005). Así como las importantes contribuciones del Dr. Juan José Castillo de la Universidad Complutense de Madrid (Mayo-Diciembre de 2006), sobre todo en el periodo de la estancia de investigación que se realizó en el grupo TRABIN II, en Madrid, España, realizado de Octubre a Diciembre de 2006, financiada por CONACYT y el postgrado en Estudios Sociales de la UAM-Iztapalapa.

Una complicación aún no resuelta del todo, fue interpretar comprehensivamente el problema de investigación en términos de la propia Ingeniería del Software y establecer "puentes" con la Sociología del trabajo y el concepto de *aflicción del software*. Para ello, en el primer acercamiento al objeto de estudio (esquema 1), se construyo un esquema de dimensiones que dieran cuenta del planteamiento del problema, a partir de tres premisas, la

primera esta relacionada con la sociología del trabajo, con una pregunta central ¿Se puede estandarizar el proceso de trabajo en el software?. Una segunda premisa es la continuidad de los errores y fallas presentes en el desarrollo del software (*aflicción del software*) y la tercera, como paradoja del proceso de trabajo, señalamos que aún con herramientas y normas metodológicas de cuarta generación en el siglo XXI, no hay software libre fallos, no hay software con "cero errores", no hay software que se desarrolle "justo a tiempo", no existe un "sólo mejor camino" para el desarrollo de software a la medida.

Esquema 1
Dimensiones del problema en el proceso de trabajo cognitivo en el desarrollo de software a la medida

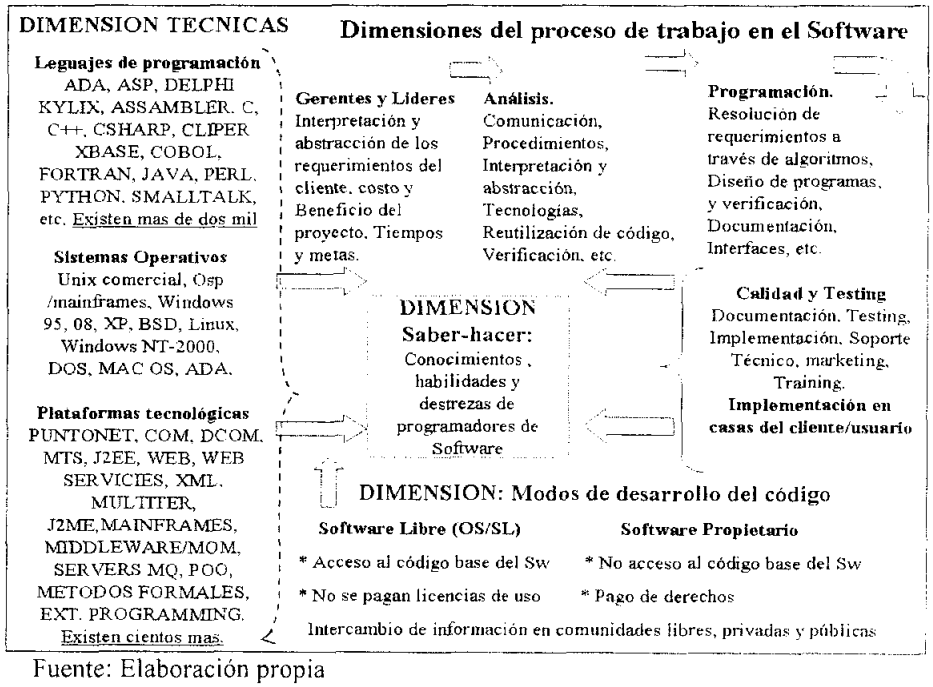


Fuente: elaboración propia

Para la comprensión del objeto se estudio se construyó un segundo esquema donde se representaron las dimensiones que influyen en el proceso de trabajo de los programadores de software. Entendiendo que el contenido de este esquema se construyó a partir de las entrevistas y documentos indagados en el primer periodo de aproximación al sujeto de estudio entre Febrero y Junio de 2005. En el esquema 2, se plantean las dimensiones que presionan el proceso de trabajo, las técnicas que son representadas por el conjunto de lenguajes de programación, sistemas operativos y plataformas tecnológicas, que representan por si mismas una heterogénea posibilidad de combinaciones; la dimensión del proceso de trabajo, que con fines metodológicos se fragmentó en gerentes y lideres, analistas y programadores, calidad y

tester, sin embargo ello no quiere decir que el proceso de trabajo siga esta división. Otra dimensión que constriñe el proceso del saber hacer de los programadores, es el modo de desarrollo del software privado y software libre (*open source).* Éstas estructuras presionan el proceso de trabajo cognitivo de los programadores de software a la medida (ver esquema 2).

Esquema 2
Dimensiones técnicas del proceso de trabajo cognitivo
en el desarrollo de software a la medida



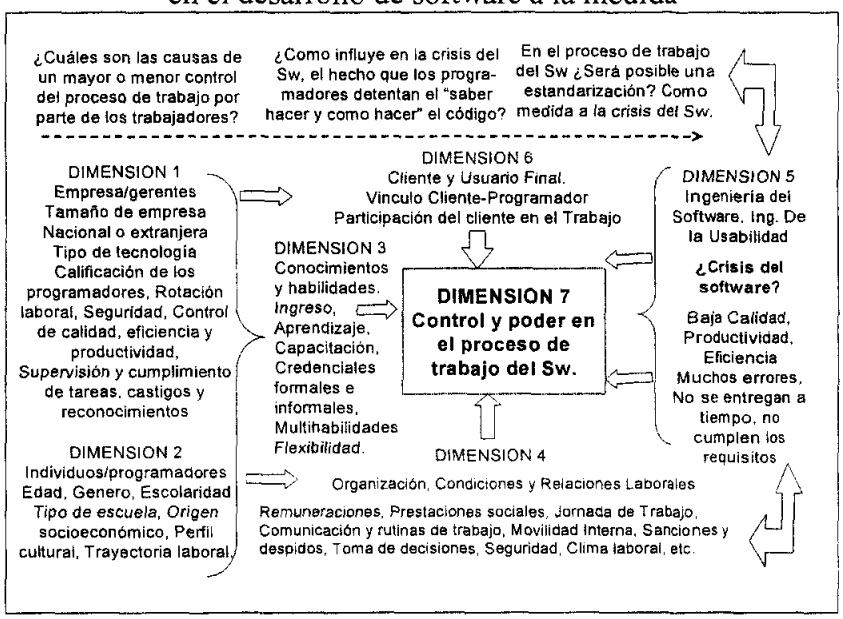
Fuente: Elaboración propia

Una vez que se planteó el problema a investigar en los esquemas 1 y 2, y las dimensiones que le pudieran explicar, se procedió a elaborar un tercer esquema donde se problematizó el objeto de investigación a partir de características *sui generis* denominadas dimensiones que comprimen y ciñen el proceso de trabajo del software a la medida; cada dimensión de análisis, representa una posibilidad de entrada para explicar la *aflicción del software*, partiendo de un supuesto: que el programador detenta el control en el saber hacer y el saber como poder en la generación de líneas de código (algoritmos) que resuelven el problema planteado por el cliente/usuario, ahora bien éste "saber hacer" y "como hacer" el trabajo esta embebido en un contexto de incertidumbres y contingencias que impregnan el proceso de trabajo. Cada dimensión aporta posibles salidas a la aflicción del software, pero también aporta elementos para continuarla y/o apuntalarla. En el esquema 3, se plantea una

serie de características que impregnan, constriñen y presionan el saber hacer y el control como poder en el proceso de trabajo cognitivo.

Esquema 3

Dimensiones que componen el control y el poder del saber-hacer en el en el proceso de trabajo cognitivo en el desarrollo de software a la medida



Fuente: elaboración propia

Los tres esquemas anteriores, permiten construir una serie de preguntas eje: ¿la industria del software a la medida se encuentra ante un proceso de "estandarización del software" a través de la intervención de la "mano visible de la administración científica de la programación"?; La serie de herramientas s y normas de calidad de la Ingeniería del Software e ingenierías similares, implican a su vez una serie de interrogaciones: ¿Representan una "administración científica" por parte de la gerencia?. Al respecto varios autores señalan que la estandarización no es posible, ya que la industria del software se halla aún en una etapa "artesanal" donde las métricas y procesos de trabajo no se homologan; existe una polarización amplia y heterogénea en los procedimientos para construir un programa; de tal forma que el conjunto de conocimientos implícitos en el saber-hacer del programador es "complejo". A partir de los esquemas anteriores y las entrevistas preeliminares se desagregaron una serie de sub-apartados temáticos y líneas de trabajo que posteriormente se tradujeron en preguntas semi-estructuradas que formaron parte de un guión de entrevista, construyéndose tres tipos de entrevistas semiestructuradas, i) Para el gerente o dueño de la empresa; ii) Para el cliente/usuario que solicita el software a la medida; iii) Para el programador (Véase anexo I).

semi-estructuradas que formaron parte de un guión de entrevista, construyéndose tres tipos de entrevistas semiestructuradas, i) Para el gerente o dueño de la empresa; ii) Para el cliente/usuario que solicita el software a la medida; iii) Para el programador (Véase anexo I).

Aunado a las entrevistas, se utilizaron fuentes secundarias cuantitativas del Instituto Nacional de Estadísticas e Información Geográfica de México (INEGI, www.inegi.com.org); Estadísticas e informes del portal electrónico de la Asociación de Software (www.software.com.mx, acceso sólo por suscripción), entre otras fuentes de la OCDE, Banco Mundial y DigiWorld. También, se utilizaron tres bases de datos cuantitativas de la Secretaria de Economía relacionadas con el desarrollo de la Industria del Software en México: i) *Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México*, (2004), realizado por la Universidad Autónoma Metropolitana a solicitud de Secretaría de Economía en Noviembre de 2004. 118 páginas. México, D.F.; formato electrónico obtenido a través de suscripción en www.software.com; ii) *Informe de Evaluación externa del Programa de desarrollo de la Industria del Software* (PROSOFT). Ejercicio fiscal 2006, realizado por la Universidad Autónoma Metropolitana, unidad Xóchimilco, Junio de 2007. 182 páginas, México, D.F.; formato electrónico obtenido a través de suscripción en www.software.com; iii) *Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información en el área metropolitana de Monterrey, Nuevo León y el Distrito Federal y su área metropolitana,* 2004. Realizado por la Universidad Autónoma Metropolitana a solicitud de Secretaría de Economía en Noviembre de 2004. 64 páginas, México, D.F. formato electrónico obtenido a través de suscripción en www.software.com.

Las entrevistas se consideran como herramienta metodológica (Sierra, F., 1998; Sierra, B., 1994; entre otros[8]). Sierra (1994) señala que la entrevista es una de las técnicas más socorridas por la investigación cualitativa, explicando que existen distintos tipos, como la entrevista sociológica, la entrevista clínica, la entrevista periodística, entre otras. Por ejemplo

[8] Sierra Bravo, R. (1994), Técnicas de investigación social, Editorial Paraninfo, Madrid, pp. 304-322; Sierra, Francisco (1998). "Función y sentido de la entrevista cualitativa en investigación social", en: Jesús Galindo Cáceres (coord), Técnicas de investigación en Sociedad, Cultura y Comunicación, CONACULTA-Addison Wesley Longman, pp. 277-333; Piña, Carlos (1989), "Sobre la naturaleza del discurso autobiográfico", Argumentos, Universidad Autónoma Metropolitana, Xochimilco, núm. 7, agosto, México, pp. 131-160.; Chanfrault-Duchet, Marie-Françoise (1988), "Le système interactionnel du récit de vie", Sociétés, mayo, París, pp. 26-31. (mimeo, traducido); Arfuch, Leonor (2002), El espacio geográfico. Dilemas de la subjetividad contemporánea, Fondo de Cultura Económica, Buenos aires, pp. 177-202.

Jean-Baptiste Fages (Citado por Bravo, 1994:285) señala siete técnicas de la entrevista: no estructurada, focalizada, no directiva, provocada de formulación libre, con preguntas abiertas pero no organizadas, con preguntas estructuradas y, con preguntas cerradas. Sin embargo, el método elegido para realizar el trabajo de campo fue la entrevista semi estructurada-enfocada con preguntas abiertas, partiendo del hecho epistemológico que "la entrevista está mediatizada por la necesidad pragmática que justifica el encuentro conversacional" (Bravo, 1994:282). En dicho encuentro entre entrevistador y entrevistado, convergen un conjunto innumerable de factores objetivos y subjetivos, de códigos e intereses tangibles e intangibles, que le atañen a los propios sujetos que interaccionan. El diseño de la entrevista fue flexible, adecuándose a las características del entrevistado (tiempo, conocimiento del tema, posición en la empresa, experiencia en el puesto de trabajo). Para diseñar el cuestionario, se llevaron a cabo 10 entrevistas exploratorias, de las cuales fueron tres académicos, cinco programadores y dos dirigentes de cámaras nacionales del software. Una vez que se diseñaron los guiones de las entrevistas (Abril del 2006) se llevaron a cabo tres pruebas piloto con programadores de software. Para llevar a cabo las entrevistas finales de trabajo de campo, se implemento la siguiente estrategia: 1.- Se definió que por esfuerzo, tiempo y recursos económicos 17 empresas eran idóneas; 2.- Se hizo un listado de treinta y cinco empresas, considerando que de éstas, 17 empresas aceptaran participar en el proyecto. El listado se construyó a partir de varias fuentes, como son Cámara Nacional de la Industria Electrónica, Telecomunicaciones y Tecnologías de la Información CANIETI, (http://www.canieti.org.mx); Asociación Mexicana de la Industria de Tecnologías de Información, A.C. AMITI (http://www.amiti.org.mx): Asociación Mexicana Empresarial de Software Libre, A.C., AMESOL, (http://www.amesol.org.mx) y el Portal del Software de la Secretaría de Economía (http://www.software.net.mx). Se procedió a elaborar un censo de las empresas desarrolladoras de software en la zona metropolitana del valle de México, para lo cual se tuvo acceso a la base de datos de AMITI, AMCIS, AMESOL y CANIETI. Además de estas listas, se procedió a ampliar la lista a través de la sección amarilla del Distrito Federal; de éstas listas se elaboró un censo de las empresas desarrolladoras de software en la zona metropolitana del Valle de México, arrojando un total de 120 empresas (Enero de 2005). Esta lista es similar a la cantidad de empresas que señala el Programa de Software (PROSOFT) de la Secretaría de Economía. Sin embargo, uno de los principales problemas de éstas bases de datos (AMITI y

AMCIS) es que integra a empresas de tecnologías de la información, es decir comprende desde aquellas empresas que operan como proveedores de software de grandes corporaciones como IBM, SUN, SAP, ORACLE, etc., así como a empresas tipo consultorías en la implementación de soluciones informáticas; hasta aquellas empresas que implementan software de otras empresas (partner) hasta las microempresas de software. De la base de datos construida se eliminaron aquellas empresas que funcionan como "socios proveedores"-especie de distribuidores de grandes corporaciones de software como IBM, SAP, SUN, ORACLE, etc.- en función de dos criterios: a) Si en la base de datos se señalaba que sólo administraban paquetería de una empresa como IBM, SAP, HP, etc.; b) Que eran proveedores de soporte técnico para la implementación del software desarrollado por otra empresa como IBM, SAP, etc. Donde se tenía dudas se consultó a la propia empresa vía telefónica. En Febrero de 2005, se seleccionó un universo de 88 empresas de software, cantidad que representa 72.5% de la muestra seleccionada[9] y de ésta se eligió al azar las 17 empresas.

Cuadro 1
Universo de empresas de software
en el valle de México

Tamaño	Empleos	Empresas	Porcentaje	Acumulado
Micro	1 - 10	33	37.9%	37.9%
Pequeña	11 - 50	41	47.2%	85.1%
Mediana	51 - 100	8	8.0%	93.1%
Grande	+ de 101	6	6.9%	100%
Total		88	100 %	

Fuente: Elaboración propia en base a la lista de empresas de AMITI.

En el cuadro 1 se observa la distribución de las 88 empresas por tamaño, estructura porcentual similar a los estudios realizados por PROSOFT en 2004 y la Universidad Autónoma Metropolitana-Secretaría de Economía en 2005. En la estructura del número de empleados por empresa, se optó por lo señalado en la clasificación por número de empleados de la ley para la competitividad de micro, pequeña y mediana empresa (MPyMEs) para el Distrito Federal, donde se observo que 92% de las empresas manufactureras son MPyMEs. Esta clasificación por tamaño de empresas es similar a la composición que presenta la estructura industrial de México y, acorde a nuestro universo seleccionado de empresas de software a la medida 93.1% son MPyMEs. De este universo de empresas, según observamos

[9] Porcentaje que es similar al estudio de la PROSOFT 2004, donde 78% de las empresas de software en la zona metropolitana del Distrito Federal, se concentran en actividades de software a la medida y software empaquetado.

en el cuadro 2, la proporción que se pretendió cubrir fue la muestra de 17 empresas, distribuidas por tamaño de empresa. Para elegir al azar las 17 empresas se procedió a dividir el universo en 50% y 50%, distribuidas al azar. La primera lista se le denominó primera fase de trabajo de campo, a desarrollarse entre Marzo y Julio de 2006. La segunda lista, integra la segunda fase de trabajo de campo, a desarrollarse entre Febrero y Abril de 2007.

La estrategia del trabajo de campo que se siguió fue la siguiente: se envío por correo electrónico a todo el universo de empresas de la primer fase o segunda -según sea el caso- contenía un mensaje de presentación con hoja embretada, dirigida a un contacto clave que se seleccionaban de las páginas web de la empresa o del representante que se señalaba en la base de datos de la empresa, en el 80% de los mensajes, previamente se hizo contacto vía telefónica. El mensaje contenía dos anexos, el primero correspondía a un oficio de presentación del investigador, del objetivo de la investigación y fines de la misma. El segundo anexo, correspondía a la presentación del proyecto en cuatro diapositivas electrónicas diseñadas en power point. Una vez que se había enviado el mensaje electrónico, se procedía ha llamar por teléfono a las empresas e identificarse con el contacto clave. Preferentemente se contactó primero a las empresas que respondieron al correo electrónico. Una vez que se estableció contacto con los agentes claves, se acordó una primera reunión de "información amplia del proyecto" que, en la mayoría de las veces, se convirtió en una entrevista informal no estructurada, pero dirigida hacia el objetivo central de la investigación.

Cuadro 2

Muestreo por cuota de mercado entre las empresas desarrolladoras de software a la medida en el valle de México

Tamaño	Total	Procedimiento	Cuota
Micro	33	33 / 88 X 17 =	7
Pequeña	41	41 / 88 X 17 =	6
Mediana	8	8 / 88 X 17 =	2
Grande	6	6 / 88 X 17 =	2
Total	88		17

Fuente: Elaboración propia

Entre el 90 y 95% de los casos, la primera entrevista fue con los gerentes de desarrollo o con los líderes de proyectos de software. Esta primera reunión tenía como fin sensibilizar a los contactos clave, que la investigación comprendía varias sesiones con agentes claves de la propia empresa. Se procedió a concentrarse en la programación de las entrevistas, que comprendía una entrevista a gerente de la empresa, una entrevista al líder del proyecto, dos entrevistas a programadores y una entrevista a un cliente de la empresa. De estas primeras

reuniones informales se acordó quien sería el contacto clave para que organizara las entrevistas al interior de la empresa y, que se coordinará con el entrevistador la planificación de las entrevistas. En la mayoría de los casos, quienes llevaron la agenda de las entrevistas fueron los líderes de proyecto, estableciéndose comunicación básicamente por correo electrónico y teléfono. En total se realizaron 35 entrevistas semi-estructuradas en 17 empresas, distribuidas por tipo de entrevistas: seis de sensibilización; cuatro con gerentes; doce con líderes; once a programadores y dos a clientes. Del total de las 88 empresas seleccionadas, 30 empresas no se estableció contacto por limitaciones de tiempo; 15 contestaron que sólo eran oficinas distribuidoras de software, que el desarrollo de software estaba en otro Estado, 17 empresas explicaron que "por políticas internas" no se permitían este tipo de entrevistas; 9 no contestaron y los teléfonos no funcionan o ya no están las empresas en esa dirección.

Cuadro 3

Numero de empresas de software en México y en el valle de México. Cantidad seleccionada de empresas por tamaño

Tamaño	Rango de empleo	México (09-2006)*	%	V. México (02.2005)	%	Trabajo Campo	%
		1,429	100%	88	100%	17	100%
Micro	1-10	619	43.3%	33	37.5%	7	41.1%
Pequeña	11-50	629	44.0%	41	46.6%	6	35.3%
Mediana	51-100	130	9.1%	8	9.1%	2	11.7%
Grande	+100	114	7.9%	6	6.8%	2	11.7%

Fuente: Elaboración propia en base a datos de CANIETI, AMITI y AMCIS.
Febrero de 2005. *Datos de INEGI, Septiembre de 2006. Acceso a Software.com

Con las 17 empresas entrevistadas se acordó un compromiso verbal de colaboración. donde la colaboración fue lenta en el avance de las entrevistas, debido principalmente a los tiempos de los líderes, gerentes y programadores. En el cuadro 4 damos cuenta del numero de entrevistas realizadas en trabajo de campo, se excluyen las entrevistas exploratorias.

Cuadro 4

Numero de entrevistas realizadas en trabajo de campo

Tipo de entrevistas	Numero de entrevistas	Tiempo (minutos)
Numero Total	**35**	**3,616**
Informales/sensibilización	9	596
Gerentes	8	1,015
Líderes	8	698
Programadores	8	1,232
Clientes	2	75

Fuente: Elaboración propia en base a trabajo de campo.

En las entrevistas se utilizó la siguiente nomenclatura: primero se señala el puesto del entrevistado: Gerente; Líder; Programador; Cliente; Otro (como diseñador, académico, directivo, etc.); posteriormente una clave de dos letras seguida de un numero, generada por el investigador para asegurar la confidencialidad del nombre del entrevistado, por último una clave de 4 letras que indica el nombre de la empresa. etc. En el cuadro 5 observamos que las empresas entrevistadas, se reagruparon en una tipología distribuidas por tamaño de empresa.

Cuadro 5

Empresas entrevistadas que desarrollan software a la medida en el Valle de México por tamaño de empresa y por tipología

	Total	**Empresas tipo 1** Colectividad sistémica	**Empresas tipo 2** Crisis y reestructuración interna	**Empresas tipo 3** Comunidades virtuales (*open source*)
Total	**17**	**6**	**5**	**6**
Micro	7	--	TADI	**M I C R O:**
Pequeña	6	DYYA ELLA	SITA MEKI CATI HUTI	LIES WALS BOCS UFOS
Mediana	2	DASA MIXE	--	BICS DINS
Grande	2	SOCU CISE	--	

Elaboración de acuerdo a las entrevistas. Trabajo de campo 2005-2006.

Limitaciones del presente estudio

La muestra es estadísticamente no representativa. Las entrevistas se realizaron a gerentes y programadores y sólo dos clientes, quedando limitado el estudio. El análisis en cuanto a los modos de desarrollo de software comercial y libre (*open source*) esta sesgado por tipo de empresa, es decir las empresas que desarrollan sistemas informáticos basadas en software libre son las empresas tipo 3, las cuales son básicamente micro empresas. Otra limitación hace referencia a que no se incorporaron entrevistas amplias a los académicos e investigadores de empresas de software, sólo se entrevistaron académicos de centros de educación superior. Se oriento a empresas que desarrollan software a la medida y son de capital nacional. Las entrevistas se enfocaron al microanálisis y no se abordo una visión macro y meso y del contexto interactivo entre actores, flujos de aprendizaje, comunidades en red que influyen y constriñen el proceso de trabajo, por ejemplo la asociaciones empresariales (AMITI, CANIETI, etc.) las políticas públicas como PROSOFT, MOPROSOFT, hizo falta observación no participante, así como análisis de fotografías en el lugar de trabajo, entre otros.

Parte I

Economía del conocimiento, y control sobre el proceso de trabajo

Capitulo I

Agotamiento del modelo de producción Taylorista-Fordista:

El conocimiento como carburante de la nueva fase capitalista

> Hubo una vez un hombre que fue a una feria de computadoras. El primer día al entrar, le dijo al guardia de la puerta: "Yo soy un gran ladrón, renombrado por mis hazañas de robar tiendas. Estás avisado de antemano, porque esta feria no escapará sin ser saqueada. En el último día de la feria, el guardia no pudo resistir más su curiosidad. "Señor Ladrón" -dijo- "estoy tan confundido, que no puedo vivir en paz. Por favor ilumíneme. ¿Qué es lo que está robando?". El hombre sonrió. "Estoy robando ideas" dijo.
>
> *El Tao de la programación*[10]

1.1.- Introducción

En el presente capitulo abordaremos la categoría analítica del capitalismo del siglo XXI: el conocimiento y su relación con la producción; Machlup (1962) y Bell (1967) en lo sesenta hacian mención de la importancia del conocimiento en la sociedad posindustrial; Karl Polanyi en 1967, se refería al conocimiento como parte nodal de la transformación productiva y social; Marx (1847, [1967]) mucho antes, señalaba la jerarquía del conocimiento en el proceso de producción. Sin embargo en el siglo XXI el conocimiento, su producción y relación con otras formas de producción, como son la información disponible, la innovación, el aprendizaje formal e informal, el conocimiento tácito, el conocimiento implícito, así como la importancia de las instituciones sociales, políticas gubernamentales, educativas, entre otras que están embebidas en interacciones en redes, en flujos o comunidades en red con presencia de inter-juegos, flujos de conocimiento y aprendizaje; con limitaciones, restricciones o impulsos en la generación de conocimiento y/o transformación de la información en nuevos aprendizajes. Es en el proceso de trabajo del software a la medida donde se configuran nuevos procesos de aprendizaje, nuevas formas de organización como son las comunidades de conocimiento, flujos de aprendizaje que rompen con el pasado taylorista, con la rigidez de un "sólo camino" para producir "con cero errores y cero defecto, justo a tiempo y con calidad total", el trabajo cognitivo del software a la medida rompe con éstas directrices del fordismo taylorismo.

---

[10] El Tao de la Programación, Traducido por Geoffrey James, trascripción de Seth Robertson, Versión Española por TESI. Acceso en fotocopia.

1.2.- El conocimiento como categoría analítica

¿Qué es lo nuevo y qué lo viejo en las denominadas nuevas formas de organización del trabajo? pregunta planteada por Castillo, J.J. (1983, 1988)[11] hacia principios de la década de los ochenta, que junto a otras preguntas sugeridas por De la Garza (1997, 1999, 2001, 2002) en torno al proceso de reestructuración del capitalismo de los ochenta y noventa, se preguntaba si acaso ¿Los conceptos clásicos del trabajo, se ajustan en general al capitalismo moderno?; ¿Estaremos en presencia de construir un análisis crítico, que de cuenta de las limitaciones del trabajo clásico?; ¿Estamos en condiciones de reflexionar sobre conceptos ampliados del trabajo más allá de la manufactura o de los modelos de desarrollo?. Éstas preguntas de De la Garza, sugieren hacer frente a las hipótesis de los teóricos posmodernos y para-posmodernos, quienes suscriben como un hecho la pérdida de la centralidad del trabajo en la producción (De la Garza, 2008)[12]. Los antecedentes de éstas hipótesis las encontramos entre los teóricos pesimistas de la sociedad pos-industrial como Machlup (1962); Bell (1967); Gortz (1982); Touraine (1985); Offe (1985); entre otros; para quienes los cambios técnicos en el trabajo implican cambios en la estructura de las ocupaciones y una perdida de identidad obrera. Otros pesimistas como Rifkin (1996); Castells (1996); OCDE (1996); OIT (1997) entre otros, ponen el acento en la perdida de empleos, como producto de la "revolución de la inteligencia"[13] que pronostica la retirada del trabajo como creador de valor y la cada vez mayor presencia de la información y el conocimiento como generador de este, como si la información y el conocimiento constituyesen materia prima disponible por si misma. Las tesis de Bell (1967) y, en menor medida de Machlup (1962) y Touraine (1985); fueron fuertemente recuperadas en la década de los noventa, a medida que tiene presencia una segunda oleada de pesimistas acerca del trabajo, que hacia principios del presente siglo redescubre al

---

[11] Castillo, J.J. (1988, 1991), (Compilador), Las nuevas formas de organización del trabajo. Viejos retos de nuestro tiempo, Colección Informes, numero 3, Ministerio del trabajo y seguridad social, España. 554 pp.: Castillo, J.J; Carlos Prieto (1983), (editores) Condiciones de trabajo. Hacia un enfoque renovador de la Sociología del Trabajo, Madrid, Centro de investigaciones sociológicas, 385 pp.

[12] Disponible el concepto en la pagina web del autor http://docencia.izt.uam.mx/egt/

[13] "Revolución de la Inteligencia". En la actualidad, 85% de todos los científicos que han vivido a lo largo de toda la historia están vivos y cuentan con herramientas más avanzadas y mayor creatividad. Ello ha conducido a que la tasa de cambio científico y tecnológico sea más rápida que en el pasado. Actualmente el conocimiento científico se duplica aproximadamente cada 5 años. Las áreas donde están surgiendo más innovaciones tecnológicas son energía nuclear, informática, robótica, biotecnología, telecomunicaciones y ciencias del espacio. http://es.wikipedia.org

conocimiento como actividad generadora de valor (Hardt y Negri: 2000, 2005)[14], invisibilizando el hecho que, quien genera el conocimiento es en parte el conjunto de trabajadores interactuando a través del trabajo mismo (Marx:1963). En cambio, para Chandler (1990) el crecimiento económico (de los Estados Unidos durante el siglo XX) corresponde en gran parte a la aplicación sistemática de la ciencia que se produce dentro de los sectores más organizados de la empresa, es decir a medida que la producción de nuevos conocimientos se combina con su aplicación a través de inversiones complementarias en investigación, producción y comercialización de productos o procesos se sucede el crecimiento económico. Esta visión de Chandler (1990) junto a los pesimistas actuales acerca del trabajo no es suficiente en la economía actual para explicar la competitividad de las empresas, porque si bien es cierto la competitividad se fundamenta en un sistema complejo de alianzas estratégicas, adquisiciones, fusiones y contratación de servicios (subcontratación), no es menos cierto que dichas estrategias a su vez tejen una serie de redes, dentro y fuera de la empresa, pero el átomo del trabajo como creador de conocimiento y de valor no desaparece frente a la *constelación de relaciones* que se suceden en la estructura en que se inserta el trabajo, por el contrario subsisten a nivel micro una serie de interacciones individuales que tienen que ver con las representaciones, sentidos e imágenes que el trabajador construye en el proceso de trabajo y fuera de éste.

La creación asilada de conocimiento ya no es lo importante, no estamos en un mercado de tecnologías segmentado al estilo Chandler (1990) o como señalaba Bell (1976), con respecto al dominio del conocimiento teórico sobre el empírico, hoy en día se están configurado nuevas formas de intercambio tecnológico, contextualizada por dinámicas ágiles en el intercambio de información, en la fluidez de prácticas de aprendizaje entre individuos, en la tasa de innovación (creativa y destructiva), donde predominan una serie de interacciones en red, en comunidades, en colaboración, en consensos entre comunidades, entre individuos y comunidades, en tiempo real y virtual, dentro y fuera del proceso de trabajo, en redes o alianzas que fluyen, se limitan o restringen el aprendizaje y el conocimiento. Por ejemplo,

[14] Para una lista extensa de la retórica de la nueva economía basada en el conocimiento véase Business Week "21st century capitalism" 12 de Septiembre de 1994; Castells, Manuel e Himanen (2002) El estado del bienestar y la sociedad de la información, Alianza, Editorial, Madrid, 215 pp.; Organización Internacional del Trabajo (2002) Aprender y formarse para trabajar en la sociedad del conocimiento, Ginebra, Informe IV. 136 pp.: Banco Mundial (2003) Aprendizaje permanente en la economía global del conocimiento. Desafió para los países en desarrollo: Editorial Alfaomega, pp. 152.

Grindley y Teece (1997), señalan que el intercambio creciente de licencias de tecnología entre empresas como IBM, Texas Instruments, Hewlett Packard y AT&T durante los noventa favorece los flujos de información y conocimiento. Otro ejemplo relevante del intercambio tecnológico en Estados Unidos son las más de 15,000 transacciones en tecnología con un valor mayor a $330 mil millones de dólares, es decir un promedio simple anual de 1,150 transacciones, con un valor de $27 mil millones de dólares generadas entre 1985-1997. Sin embargo, ello no quiere decir que la investigación mas desarrollo (I&D) de las empresas este siendo simplemente sustituida por una I&D realizadas externamente; aunque son las condiciones actuales de intercambio de tecnología, ya sea en empresas establecidas o empresas que se especializan en la producción de tecnología a través de comunidades tecnológicas publicas, semipúblicas o privadas, que forman parte de un importante papel en la promoción de la innovación (Arora, 2001). Lo anterior no anula nuestra intención de analizar el trabajo del programador frente a la red, simplemente el trabajo de programación es parte de la red y, es importante abordar desde el proceso de trabajo las interacciones individuales del programador frente a la pantalla de trabajo.

El conocimiento, sea generado en la red, internamente en la empresa, en las interacciones en una comunidad, colectividad ó través de prácticas individuales, se instituye entre las nuevas formas de trabajo como una nueva categoría o "materia prima intangible" imprescindible para lograr competitividad en las empresas, lo cual ha implicado que las unidades empresariales gestionen nuevas formas de crear y diseminar el conocimiento. Al respecto Nonaka y Takeuchi (1999) señalan que las empresas que alcanzarán el éxito de un modo firme, son aquellas que generen conocimiento, lo difundan por toda la empresa y lo incorporen a nuevas tecnologías, procesos y productos. Para estos autores, el conocimiento se relaciona en la manera en que lo tácito puede hacerse explícito, para lo cual proponen una teoría sobre la creación de conocimiento organizacional, el cual según autores como Álvarez et. al. (2001) posee dos dimensiones: a).- La epistemológica, en la cual se distinguen dos tipos de conocimiento: explícito y tácito; b).- La ontológica, en la que se distinguen a su vez cuatro niveles de agentes creadores de conocimiento: el individuo, el grupo, la organización, y el nivel organizativo. Existen otros modelos de conocimiento, por ejemplo el desarrollado por Zander y Kogut (1992) el cual es un modelo dinámico de crecimiento del conocimiento de la empresa, donde los individuos poseen el conocimiento, pero además cooperan en una

comunidad social que puede ser un grupo, una organización o una red. El modelo de Hedlund (1994) no solo se centra en la creación del conocimiento, sino en la manera de transferirlo, transformarlo y difundirlo en toda la organización. El modelo se construye sobre la interacción entre conocimiento articulado (explícito o codificado) y conocimiento tácito, estructurado en cuatro niveles de agentes generadores: individuo, pequeño grupo, organización y dominio organizativo (Álvarez et. al. 2001). A estos modelos se agrega el enfoque de la innovación, entendida como la aplicación y generación de conocimiento nuevo. Al respecto Cimoli (2000:6-7) señala que es a través de la internacionalización de la tecnología y la innovación, la producción y mejora de capacidades que se han conformado redes internacionales, donde se produce conocimiento y tecnología. Es decir, la innovación no es aislada, por el contrario la innovación tiene que ser considerada y definida como un proceso interactivo en el cual las empresas nunca innovan en aislamiento, como bien plantean los enfoques de la economía de la innovación de Edquist (1997) y Nelson (1993).

En este contexto es importante considerar las alianzas, redes estratégicas e interacciones entre empresas, universidades y otras instituciones que como agentes sociales agilizan, activan e impulsan la innovación y los flujos de aprendizaje, entendidos éstos como un proceso social en red, en la cual hay interacciones intensivas, laxas con claroscuros entre información, conocimiento, aprendizaje innovación y tecnología. El concepto de información se construye como "el acto o proceso por el cual el conocimiento es transmitido" (Machlup, 1980, citado por Casas, 2001). David y Foray (2002:13) establecen que "mientras el costo de reproducir cantidades de información no implica mas que el precio de hacer copias, reproducir el conocimiento es un proceso bastante mas complejo (...) la capacidad cognitiva no es fácil de articular explícitamente o de transferirla". La innovación necesita de la información y el conocimiento para generarse, lo cual puede tomar la forma de innovación tecnológica y de organizacional social. Al respecto Lundvall (1992:9) señala que la "innovación no constituye un evento o una etapa, es un proceso resultante de aprendizajes interactivos y acumulación de conocimiento, el cual puede tener lugar en muchas organizaciones sociales, empresas e instituciones de educación superior". La categoría analítica en las tecnologías de la información es el conocimiento, el cual ha dado lugar al desarrollo de innovaciones, como la microelectrónica e informática, las cuales, parecería ser, constituyen la base de un nuevo paradigma tecnológico que modifica las condiciones de la producción a través de la

automatización de procesos y distintos mecanismos de transmitir la información y transformación del conocimiento (innovación).

No es que en el siglo XXI se redescubra la categoría analítica de conocimiento, sino que hay una redefinición conceptual no tradicional de dicha categoría. Por ejemplo Foray (1996 y 1997) señala que el conocimiento siempre ha ocupado un puesto central en el crecimiento económico y progreso del bienestar social; se caracteriza por la capacidad de innovar y crear nuevos conocimientos e ideas, proceso que tuvo un boom entre las décadas de los cincuenta y setenta, no obstante que en este período predominaban las empresas trasnacionales; con una organización vertical y centrada en la rigidez de la producción, donde el acceso a la información en tiempo real era prácticamente imposible y prevalecía un aprendizaje aislado, un conocimiento dividido entre la gerencia que establecía el "que hacer" y "como hacer"; los trabajadores ejecutaban "tiempos y movimientos mecánicos" vigilados por la gerencia (Taylor, 1949). Después de la década de reestructuración de los ochentas, hacia principios de los noventa cobra fuerza la reestructuración flexible del capital que restituye capacidad de decisión a los trabajadores; flexibilidad que tiene como eje una mayor interacción y toma de decisión entre individuos y de manera creativa; de tal forma que la organización industrial de las empresas es mas del tipo horizontal y coexiste un mayor flujo de información y transformación del conocimiento en la cadena de producción -en tiempo real- lo que permite la posibilidad de modificar y transformar el proceso de producción en cualquier momento (producción flexible). Foray (1996 y 1997) establece que hacia fines del siglo XX, estamos en presencia de una nueva etapa del conocimiento, el cual se posiciona en el centro de la economía como categoría analítica; además consideramos que se enmarca en un contexto de reestructuración, flexibilidad y fluidez de la información, innovación y conocimiento, en relación a los años anteriores, destacando algunas tendencias recientes:

- Aceleración de la producción de conocimientos: existencia de una fuerte intensidad del progreso científico y tecnológico a partir de una rapidez en la creación, acumulación y apreciación del conocimiento.
- Surge un nuevo tipo de institución fundamental: las *comunidades del conocimiento*, que son redes de individuos de diferentes entidades cuyo objetivo es la producción y circulación de nuevo conocimiento.

- Expansión del capital tangible e intangible: con la incapacidad del sistema de detener y explicar la caída en la productividad y la creciente diferencia de ésta entre los países, emerge una nueva teoría que tiene que ver con las mejoras en la calidad a partir de invertir no sólo en equipo físico, sino también en el capital humano, pero orientado a la creación de nuevos conocimientos y nuevas ideas.
- Alta intensidad y aceleración de la innovación: se refiere a superar la actividad tradicional de investigación y desarrollo que genera productos y patentes, hace alusión a integrar toda una serie de conocimientos que se generan "fuera de la línea", es decir promover el aprendizaje tácito, que implica aquel conocimiento que todo individuo "aprende haciendo" dentro o fuera del proceso de trabajo.

Estas tendencias están ligadas a las nuevas formas de generación de conocimiento, que se imponen al concepto de gestión simple del conocimiento de la era taylorista-fordista; hacen referencia a una gestión del conocimiento, que no tiene que ver con la linealidad, sino con la existencia de individuos que colectivamente coproducen conocimiento en comunidades virtuales o reales (por ejemplo los doctores que comparten información de sus pacientes en amplias bases de datos, las redes de científicos, usuarios de software libre, etc.) en estas *comunidades de conocimiento* se integran mecanismos de intercambio de información, donde lo mas importante no es la existencia de la información, sino su transformación en conocimiento nuevo; es decir la capacidad de generar capacidades cognoscitivas nuevas. Un ejemplo, es el modo de producción de software libre (*open source*)[15], donde el desarrollo del conocimiento se sucede en *comunidades de conocimiento virtuales*, en las que se accede al conocimiento generado, siempre y cuando se cuente con herramientas técnicas y cognoscitivas que requieren éstas comunidades como son Debian, Ubuntu, Gnome, Linux, etc. donde éstas comunidades son creadas por instituciones sin fines de lucro, por miles de individuos que se afilian virtualmente y colaboran gratuitamente en la creación de programas, así como por donaciones económicas de instituciones gubernamentales trasnacionales, políticas de apoyo, subvenciones de grandes empresas como IBM, Sun, HP y Apple entre otras que han invertido

[15] Este modelo de software libre es un ejemplo tácito, hartamente interesante, que esta atrayendo la mirada de tecnólogos, sociólogos, economistas, entre otros investigadores, es este fenómeno de compartir información gratuita, sólo con el fin de ampliar la "comunidad de software libre", esta compartimiento gratuito impacta a los colosos de los sistemas informáticos, por citar un dato, en los últimos meses del 2005 se dio a conocer en la red un navegador basado en plataforma de Software libre: Firefox, con 17 millones de descargas (puede descargarlo en www.mozilla.org) en los primeros meses, cantidad que significo arrebatarle casi 15% del mercado mundial de Explores del coloso tecnológico: Microsoft

en desarrollar plataformas en software libre (*open source*), por ejemplo las más conocidas es Red Hat, creadora del navegador Firefox, que funge como competencia directa del navegador Explorer de Microsoft: al respecto Mitchell Baker comenta en una revista especializada "*Microsoft dejo de innovar su propio producto. Firefox...ha surgido con excitantes características*". El jefe de diseñadores Brendan Eich, comenta "t*al vez la mayor motivación sea la seguridad. Con frecuencia los usuarios de Explorer (Microsoft) son victimas de brechas de seguridad e invasiones de spyware, y dado que Explorer (Microsoft) esta integrado a Windows, su vulnerabilidad le da a los de afuera acceso a toda su computadora*" (Newsweek, 14 de febrero, 2005 p. 6). Si suponemos que las *comunidades de conocimiento* forman parte de las nuevas formas de generación del conocimiento y éstas representan el surgimiento de un nuevo paradigma, que tiene como eje el uso intensivo del conocimiento, información e innovación; tendríamos que la sintaxis de éstos componentes (conocimiento, información e innovación) "...dependen positivamente de la complejidad y articulación de redes (cluster, sistemas locales), competencias endógenas de los agentes, desarrollo de capacidades tecnológicas, grado de movilidad de los recursos humanos..." (David y Foray, 2002; Novick, et.al. 2002; OCDE: 2000; Casas, 2001 y 2003; Casalet, 2002). Estos "dispositivos" (conocimiento, información, e innovación) nos conducen a su vez, a otros debates relacionados con la gestión del conocimiento y sus agentes de cambio, como son las instituciones educativas, centros científicos y tecnológicos; tipos o modos colectivos de acción en investigación tecnológica (Gibbons, et. al. citado por OCDE: 2000). Sin embargo, en la presente tesis no abordaremos el análisis de redes entre agentes de cambio, como son las universidades, centros de desarrollo de software, laboratorios de innovación tecnológica; tampoco se analizará la toma de decisiones empresariales o el papel de éstas en las redes de innovación, tampoco se pretende hacer un estudio de las comunidades de conocimiento, sean estás generadas en colectividad, en colaboración, en forma virtual o mediante políticas institucionales. Consideramos abordar una perspectiva de análisis micro, de los individuos en interacción frente al proceso de trabajo, se pretende abordar las eventualidades, conflictos y luchas por el poder y el saber hacer entre los distintos actores que están en interacción en la base del trabajo: líderes de proyecto, programadores y clientes. Es decir, nos interesa el nivel más micro de esta estructura de relaciones.

1.3.- Las comunidades de conocimiento y el aprendizaje

Uno de los debates presentes, son las formas de generación de conocimiento a través de la configuración de las *comunidades de conocimiento* que se están conformando -como las de software libre, médicos, reporteros sin frontera, documentos comunitarios como wikipedia, etc.- éstas se constituyen como dispositivos generadores de información e innovación del conocimiento; por ejemplo cuando en una *comunidad de conocimiento* (Debian, Ubuntu, Linux, Gnome, Microsoft, Oracle, etc.) ingresa una determinada información que les permite innovar, coproducir, experimentar y desplegar nuevas aplicaciones que genere mejores métodos que aquellos que les dieron origen (es decir la información que llego inicialmente se transforma en conocimiento nuevo) son puestos a disposición de la comunidad que le generó[16]. Parecería ser que esta actividad intrínseca a las *comunidades de conocimiento* generaría "islas del saber-hacer", sin embargo la característica del acceso libre (*open source*),, permite una *fluidez en la información* que paralelamente funciona como un mecanismo restrictivo, es decir, aquellos que participan en dichas comunidades de conocimiento deben poseer cierto grado de conocimientos técnicos y tácitos, así como conocer las prácticas sociales e informales hacia el interior de las comunidades que restringen el acceso a determinados procesos de transformación de la información disponible. Lo cual no quiere decir, que no se tenga acceso libre a la información. La particularidad de las *comunidades del conocimiento*, es la *fluidez de la información*, que genera mayor disponibilidad de información, una tasa de innovación que impulsa nuevos aprendizajes, disminución de costos y -en términos de Marshal- las *comunidades del conocimiento* no están aisladas, sino interconectadas donde transitan conocimientos específicos, por tanto no serían comunidades-islas cerradas-aisladas, por el contrario dependiendo de su carácter (redes públicas, semipúblicas o privadas) se reproducirán los conocimientos en aprendizajes nuevos a través del uso intensivo del conocimiento. Lo anterior es en el mejor de los casos, sin embargo como ya señalamos coexiste una serie de restricciones no formales que impiden que el acceso al conocimiento sea del tipo casuística, o que no está libre de interacciones, subjetividades, resistencias, negociaciones, consensos y conflictos entre quienes participan en dichas comunidades.

[16] Entre las comunidades de conocimiento de software libre con trayectoria y permanencia sólida en Internet están: Debian, Ubuntu, Gnome, entre otras.

En las *comunidades del conocimiento* está intrínsecamente ligada la interacción de actores, instituciones y agentes que ahí concurren, los cuales son heterogéneos en saberes: programadores, compradores, internautas, desarrolladores de hardware, doctores, diseñadores, periodistas, etc. Estos agentes, en su interacción social, productiva, técnica y virtual, sea en tiempo real o no, hacen posible toda una gama de procesos de innovación y renovación del conocimiento, que podemos definir como un proceso continúo de intercambio cognitivo embebido en una *constelación de estructuras objetivas, acciones individuales y subjetivas de quienes interaccionan.* En las *comunidades del conocimiento* la *interacción social* y la *constelación* señalada anteriormente, constituyen el eje a través del cual se crea la *fluidez de la información* no sólo en el sector industrial, sino también en servicios. Hatchuel y Weil (1995) señalan que la interacción social entre los agentes está incluida en la definición del *saber hacer* y en cada una de las formas concretas del *saber*. Se pueden reconocer las diferentes formas que asume la interacción social traduciéndose en distintas formas de aprendizaje colectivo e individual que se realizan en las comunidades de conocimiento (Lave y Wnger, 1991 citado por Hatchuel y Weil, 1995). La fluidez de la información se concretiza en aprendizajes y conocimientos que se transforma en bienes tangibles como innovaciones tecnológicas, o bien en intangibles como prototipos, programas de software, etc.

En la *interacción social* el trabajo es creador y no una simple actividad rutinaria, en el cual los procesos de innovación constante transforman la dinámica organizacional y social en la empresa, lo que propicia la existencia de dispositivos, incentivos y reglas de acumulación de diversos tipos de conocimiento, así como su utilización reproducción y creación de nuevos conocimientos. Foray y Lundvall (1996) señalan que el problema de la relación trabajo-conocimiento plantea dos enfoques: a).- se reconoce siempre que en cualquier sociedad, el centro del desarrollo económico descansa en la competencia humana, y por tanto es usual aplicar una perspectiva analítica que coloca al centro del debate el aprendizaje y el conocimiento en cualquier formación económico social histórica; b).- se presta atención a las nuevas características de la economía y hace legítimo señalar un nuevo estadio de la economía y la sociedad, llámese economía basada en el conocimiento ó sociedad de la información y el conocimiento; economía del aprendizaje o sociedad de la información; la economía es más dinámica y directamente enraizada en la producción, distribución y uso del conocimiento que antes. Foray y Lundvall (1996) consideran que la creación y difusión del conocimiento está en

la base y emana de las actividades rutinarias en la vida económica, y suponen que toma la forma de *aprender haciendo, aprender usando y aprender interactuando,* de tal forma que la organización más amplia de la sociedad y de las firmas es lo importante. La propuesta de estar ante una economía basada en el conocimiento ó sociedad del conocimiento, se deriva de las nuevas tendencias en la producción y en el mercado de trabajo donde "los cambios en la estructura del mercado laboral y de la producción presentan como la economía está ampliando su devenir basada en el conocimiento" identificándose transformaciones fundamentales: a).- en cada sector hay una creciente proporción de trabajo calificado y una tendencia al crecimiento del empleo, que es más rápido en aquellos sectores intensivos que demandan trabajo altamente calificado; b).- existe un contenido creciente de conocimiento e información en los productos y procesos.

Para los autores citados, el concepto de *economía del aprendizaje* es más preciso que sociedad del conocimiento, porque no se tienen argumentos que demuestre que el conocimiento es mucho más importante hoy de lo que ha sido hasta ahora. Consideramos que las hipótesis que están detrás del concepto de economía del aprendizaje son: a) más que la tasa de cambio técnico, es la necesidad de rápido aprendizaje la que se ha incrementado en la economía actual; b) lo central para el desempeño económico no es la posesión del conocimiento por agentes y organizaciones en cierto punto del tiempo, sino la capacidad de aprender (y olvidar); c) el aprendizaje es el punto central del proceso de trabajo. Es decir la capacidad de aprender, y especialmente aprender nuevas calificaciones y competencias, son necesarias en el proceso interactivo y social de la nueva economía, lo cual no puede sucederse en una economía pura de mercado (Foray y Lundvall:1996:21) [17].

El concepto de *aprendizaje* permite distinguir entre conocimiento tácito y codificado: el conocimiento codificado es aquel procedimiento mediante el cual se transforma un conjunto de procedimientos del *saber-hacer* en información disponible, que puede ser relativamente fácil transmitirla a través de la infraestructura de la información; esto es que la facilidad de transmitir, verificar, almacenar y reproducir conocimiento significa una reproducción del conocimiento. El conocimiento tácito se refiere al conocimiento que no puede ser fácilmente transferido, porque no ha sido presentado en una forma explícita. Un importante tipo de conocimiento tácito es la calificación: la persona calificada sigue reglas que no están

[17] Foray y Lundvall: 1996 18-21; además Lundvall y Jonson:1994; Casas R., 2001 y 2003.

reconocidas como tales por las personas que le siguen. De acuerdo con Polanyi (1958), la única forma de transferir este conocimiento es a través de un tipo específico de *interacción social* similar a la relación de aprendices, esto es que no puede ser vendido y comprado en el mercado y, su transferencia es extremadamente sensible al contexto social. Una de las características contemporáneas, es que los límites entre conocimiento tácito e implícito cambian continuamente, porque hoy se intenta codificar todo, sin embargo ello no significa que el conocimiento tácito termine, sino que es una relación dinámica y siempre habrá conocimiento implícito. Ahora bien, la codificación es importante, porque reduce los costos del proceso de aprendizaje del conocimiento (de la difusión), el conocimiento adquiere así una forma concreta, objetivada, donde cada vez más adquiere las propiedades de una mercancía, se externa el conocimiento y permite que las firmas puedan acceder a él, a un costo determinado, pero menor al conjunto de *saber-hacer* no codificados (ver cuadro 6).

Cuadro 6

Tipos de conocimiento en la economía del aprendizaje (*saber-hacer*)

Know what: saber qué (conocimiento de qué o cuál), se refiere al conocimiento acerca de los "hechos", este conocimiento sería lo que normalmente llamamos información. Este conocimiento es importante en áreas como leyes y medicina.

Know why: saber porqué (conocimiento del porqué), se refiere a los conocimientos científicos, los principios y leyes de movimiento en la naturaleza, en la humanidad y en la sociedad. Puede ser importante en ciertas áreas: química, electrónica, etc. Su producción y reproducción está organizada en instituciones especializadas, las firmas tienen que interactuar con ellas para accesar a ese tipo de conocimiento.

Know how: saber cómo (conocimiento del cómo), es la capacidad de hacer algo.

Know who: saber quién (conocimiento del quién) se refiere a la mezcla de diferentes tipos de calificación, incluyendo lo que puede ser caracterizado como *la calificación social*. Involucra la información acerca de quién sabe qué y quien conoce cómo hacer qué. Pero especialmente involucra la formación de relaciones sociales especiales con los expertos involucrados, las cuales hacen posible obtener acceso a su conocimiento eficientemente.

Fuente: Elaboración propia en base a Foray y Lundvall: 1996 18-21;

Las actuales tendencias de la *economía del aprendizaje* ó economía del conocimiento, según damos cuenta en el cuadro 6, pueden ser vistas como una respuesta a la necesidad de un manejo más efectivo del conocimiento codificado, dando un nuevo rol al aprendizaje tácito, en especial el conocimiento acumulado a través de las trayectorias de aprendizaje, habilidades y destrezas en los individuos. Lundvall, recupera una propuesta previa sobre los tipos de conocimiento y señala que hay una tendencia hacia una creciente codificación del *Know what*

y Know why; para el caso del conocimiento científico, sólo es codificado aquel conocimiento que no se está produciendo recientemente, donde no hay competencia por alcanzarlo; además argumenta que hay esfuerzos por codificar el *know how*, pero donde es más difícil la codificación del *know who*, aquí las redes sociales son importantes. Otra tendencia que explica Lundvall, es que se está dando un cambio entre lo que se llama conocimiento público y privado, pues la interacción social en redes permite que lo público se fortalezca, pero ahora con códigos que son propios a esas redes, y por ello su importancia. La economía del aprendizaje enfatiza que se está dando una polarización, donde el trabajo menos calificado tiene menor demanda. Esto a su vez se vincula con la globalización y el interactuar de las economías. Sin embargo, señala que si bien la globalización puede mover empleos no calificados hacia los países atrasados, esto no explica por sí misma la caída de la demanda de empleo no calificado en los países desarrollados. Otra explicación es que el cambio tecnológico ha fortalecido la demanda de trabajo calificado; de igual forma el cambio institucional (debilidad de los sindicatos, y conductas de las empresas) determinaron el movimiento hacia trabajo más calificado.

Además de los cambios señalados, no se ha considerado del todo la perspectiva microsocial, por ejemplo la posiciones del sujeto frente al trabajo, la identidad en la interacción social, la construcción del sentido en el contexto de las redes, las comunidades y colectividades: la disposición o no del individuo al colaborar, la confianza o rechazo implícito en forma de sabotaje, descuido intencionado o accidente premeditado; o bien una resistencia abierta y dirigida en la comunidad o red que se este inserto; es decir, consideramos que el consenso y el conflicto, la resistencia y el inter-juego cobran un nuevo rol embebido en los procesos de aprendizaje y generación de conocimiento; asimismo el tipo de conocimiento -tácito y codificado- expresa relaciones distintas y, los grados de confianza y colaboración son importantes –por ejemplo aquél conocimiento que sólo se adquiere en una relación experto/aprendiz, donde la experiencia, interacción social en comunidades y colectividades son centrales para su transmisión configurándose *flujos de aprendizaje* . Esta perspectiva microsocial pretende ser abordada por la economía del aprendizaje, como bien señala Foray y Lundvall (1996:24) que las comunidades y su grado de cohesión social, marcan distintos grados de aprendizaje y la explicaciones de éstos deben desafiar la sostenibilidad y eficiencia de las comunidades: "Primero, la necesidad de flexibilidad y rápida innovación pueden ser

totalmente alcanzadas por organizaciones donde el liderazgo puede delegar responsabilidad a la mayoría de los empleados. Segundo, las organizaciones de manera creciente tienen que abrir sus límites para acceder al conocimiento a través de la interacción con agentes externos. Tercero, la exclusión permanente de partes de la fuerza de trabajo minará la flexibilidad de las firmas que puedan obtener por la contratación y despido de trabajadores calificados".

1.4.- El desarrollo de software: un proceso de producción de símbolos

La industria del software representa diferencias sustanciales con el viejo régimen fordista-taylorista, en el cual el proceso de trabajo estaba regido por el cronometro, por el capataz, por la administración visible del conocimiento en tiempos y movimientos (diseño y ejecución, concepción y trabajo). En cambio en la producción de software a la medida, la producción es inmaterial donde la materia prima es un conjunto de símbolos (lenguajes de programación) que se combinan y reconfiguran en algoritmos compuestos de signos, donde estos signos responden a su vez a problemas conceptuales del cliente (símbolos de requerimientos). El trabajador del software no sólo debe transformar la información del cliente (símbolos o diseños de las necesidades del cliente) en un conjunto de *textos sígnicos* que posiblemente resuelvan el problema planteado, resultado que en términos de conocimiento es una innovación. Es decir, el trabajador del software no sólo participa en el diseño del los algoritmos (signos) sino que posee el control del como y el porque se construyen dichos signos a partir de un conjunto de símbolos (lenguajes de programación). Procedimiento que rompe con los cánones del taylorismo, ya que el proceso de trabajo del software está ceñido a la producción de signos (algoritmos) a partir de otros símbolos (lenguajes de programación), procedimiento que supone participación e intercambio de información y conocimiento entre aquellos que participan. Ahora bien, este intercambio de información y conocimiento está embebido en varios procesos que constriñen al proceso de trabajo: a) implementación de tecnología moderna o no: hardware, lenguajes de programación, tipo de procesadores, tipo de bases de datos, plataformas tecnológicas, etc.;b) tipo y calidad del proceso comunicacional en el trabajo: fluidez y grado de la información, tipo de comunicación como puede ser trabajador-trabajador, trabajador frente a la pantalla, trabajador fuera del horario de trabajo, pero trabajando, etc.; c) el proceso social: limites a la fluidez de la información, resistencias y conflictos en la comunicación y generación de conocimiento, etc. Estos procesos al interior de

la esfera del trabajo rompen con los paradigmas tayloristas-fordistas y, significa repensar el proceso de trabajo en el desarrollo del software.

Para abordar el proceso de trabajo del software se hace necesario, primero detenernos a explicar que es un software y la diferencia consustancial con respecto al hardware; segundo comprender cual es el contexto que da origen a la Ingeniería del Software y su incidencia en el proceso de desarrollo de un programa informático. Tercero, teniendo en cuanta el significado del proceso de desarrollo de un software y la incertidumbre que le encierra, estaremos en condiciones de abordar el problema del proceso de trabajo del software, pero desde la perspectiva de la Sociología del trabajo. De tal forma que en el siguiente capitulo abordaremos los dos primeros aspectos, donde intentaremos responder a la pregunta ¿Los argumentos del control del proceso de trabajo elaborados desde y para la manufactura pueden servir para interpretar los mecanismos del trabajo en el desarrollo de un programa de software? y construir un concepto ampliado de trabajo que integre las nuevas formas de organización en el proceso de trabajo simbólico.

Para Boscherini y Poma (2001) el proceso de producción del software no es una simple producción física de bienes separados del trabajo, por el contrario significa una pro/ducción, donde se trata de producir conocimiento. Para Pressman (2002) la producción de conocimiento en el software, se genera cuando se crea un conjunto de códigos agregados en algoritmos, que a su vez producen un resultado o una solución a un problema previamente planteado por el cliente o usuario; en otras palabras, el software es un programa informático que articula una serie de símbolos que cuando son ejecutados en un dispositivo físico (computadora personal o portátil, palm-top, etc.) produce resultados prácticos según una lista de requerimientos previamente acordados. La característica de la industria de software con respecto a la tradicional manufactura, es que cuando la industria del software "procesa" un programa informático éste sigue existiendo en propiedad de la empresa, aún que se haya vendido al cliente. El cliente sólo accede al uso exclusivo del programa informático, no sólo por el pago de licencia de uso, también por la limitación al acceso de los algoritmos que integran el software. El cliente, entendido como quien adquiere software, sólo obtiene un limitado derecho de uso sobre las tareas y rutinas desplegadas en la interfaz gráfica, ya que en la transacción económica no recibe el derecho de "propiedad" con respecto a la información y conjunto de conocimientos que representan los algoritmos estructurales del software que

adquirió, de tal forma que está limitado al uso y no obtiene autorización para modificarlo, revenderlo, etc. La empresa que desarrolla el software acumula información y conocimientos de los conjuntos de símbolos desarrollados (algoritmos), mismos que continúan acumulados tácitamente, no en la empresa que le desarrollo, sino en el conjunto de programadores que participaron en la creación del software; es decir el conjunto de algoritmos creados forman parte de los conocimientos acumulados del o los programadores que participaron.

Otra característica de esta industria es que el software desarrollado no se estropea, no se deteriora; pero la curva de vida presenta una serie de contingencias: el desarrollo de nuevo hardware influye en la compatibilidad del sistema; la incorporación de nuevo software genera problemas de interoperabilidad. Otras dificultades son la casi inexistencia de documentos que expliquen el procedimiento del cómo y porque se configuraron determinadas rutinas de códigos (algoritmos). Es decir, mientras que en la manufactura la gerencia posee los documentos y procedimientos del proceso de trabajo, en la industria del software es complejo establecer procedimientos que documenten eficientemente el proceso de trabajo. En otras palabras, en la manufactura el producto es generado por un conjunto de trabajadores y éstos pueden darle mantenimiento, actualización y compostura, o bien por otro conjunto de trabajadores diferentes, no necesariamente los mismos. En cambio en un software el mantenimiento y/o actualización se torna complejo si lo hacen otros trabajadores distintos de quienes lo desarrollaron: es complicado ofrecerle mantenimiento y/o actualización al software cuando la documentación del programa no existe y, suponiendo que la documentación existiera, aún así es problemático comprender la lógica algorítmica del como está desarrollado el programa que se le intenta dar mantenimiento. En otras palabras el software no se descompone o estropea en su uso, sino que en caso de errores o fallos, éstos ya estaban ahí desde su adquisición y está presente en todas las copias del mismo.

Capitulo II

Ingeniería del software y trabajadores del conocimiento
en el proceso de producción de símbolos

> Los Desarrolladores de antes eran misteriosos y profundos. No podemos comprender sus pensamientos, por eso, todo lo que haremos, será describir su apariencia: Despierto, como una zorro cruzando el agua. Alerta, como un general en el campo de batalla. Amable, como un anfitrión saludando a sus huéspedes. Sencillo, como bloques de madera sin tallar. Opaco, como pozos negros en cuevas oscuras. ¿Quién puede decir los secretos de sus corazones y mentes?.
>
> El tao de la programación

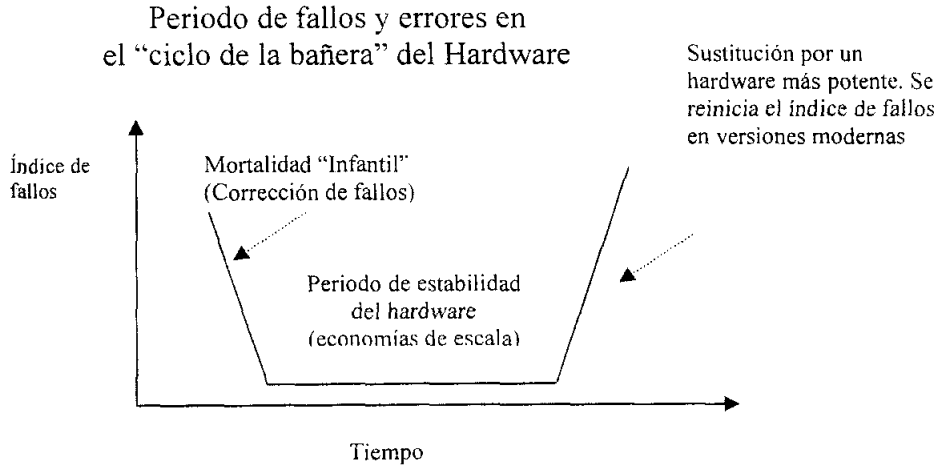
2.1.- Introducción

Durante las primeras décadas de los 40´s a los 60´s los programas de software se suponían un añadido del hardware, ambos se consideraba como uno sólo; la programación de software, se consideraba un acto heroico, el software se contemplaba como una obra de "arte" para el que no existía metodología, se realizaba sin ninguna planificación homogénea, producto de una "mente brillante" aislada e incomprendida. En esta etapa artesanal, para desarrollar un programa de software era necesario que la empresa lo necesitara y ella misma lo creaba ("desarrollo en casa"). En consecuencia, el desarrollo de software tenía muy poca difusión, habitualmente los programadores que desarrollaban software lo hacían porque lo necesitaban y lo creaban para la empresa que laboraban, éstas condiciones hacían que el desarrollo de software fuera caro, aislado y limitaba el desarrollo social del conocimiento del mismo. Hoy día la programación moderna sigue siendo compleja, y, en ocasiones, hasta cierto punto lejana y cara; por ejemplo los programas de inteligencia artificial, software para el análisis y modificaciones del ADN, software especializado en biotecnológica, entre otros. Aunque hay otros tipos de software que se ha socializado el uso, por ejemplo el software comercial (Microsoft, Oracle, Sun, etc.); el software a la medida, el software de libre acceso en Internet (Ubuntu, Gnome, Linux, etc.). Entre ambas eras -la "artesanal" y "moderna"- coexisten "puentes" que permanecen hoy día; por ejemplo, la concentración del *saber hacer* por los programadores; persistencia de las incertidumbres, contingencias y fallos en el software, problemas que la Ingeniería del Software con las herramientas y normas que implementa no ha logrado atenuar del todo la *aflicción del software*. A continuación abordamos el contexto de la industria del software y, la persistencia de fallos y errores en el proceso de trabajo, entendiéndose estos como "*aflicción del software*", problema "opaco como un pozo negro en una cueva obscura" que aún con los avances técnicos de la Ingeniería del Software no se logra predecir con calidad "cero errores", no se garantiza la entrega a tiempo del software.

2.2.- Periodización de la industria del software

¿Que diferenciación existe entre hardware y software?, brevemente diremos que cuando se construye hardware, el proceso creativo humano basado en el análisis, diseño, construcción, pruebas de calidad, etc. se traduce finalmente en un objeto físico; en cambio para el caso del software éste es un elemento lógico, inmaterial, es decir, el software no se fabrica en el sentido clásico de los bienes físicos; el costo del software radica en el diseño, no en el proceso de manufactura como el hardware; no se puede proyectar costos como si fuera un proceso de fabricación común[18]. A diferencia del hardware, el software no se estropea, no se deteriora, no se corrigen del todo los problemas de calidad, no se cumplen fácilmente las metas de entrega a tiempo. En el hardware -según vemos en la grafica 2- se presenta un periodo de tiempo en que los fallos y errores son comunes, situación que es conocida como "mortalidad infantil" de proyectos de hardware, sin embargo, una vez corregidos o cuando al menos se logra "sacar en limpio" dichas fallas -que frecuentemente son atribuidos a defectos en el diseño o en la manufactura-, cuando se corrigen los defectos, se entra en periodo de estabilidad durante un cierto lapso de tiempo. Hoy día la tasa de cambio técnico es dinámica e inversa al costo y tiene dos efectos, por el lado de la demanda efectiva, por un lado el mayor consumo de hardware y. por otro, el abaratamiento "empuja" al desarrollo de la industria del software de todo tipo.

Grafica 2



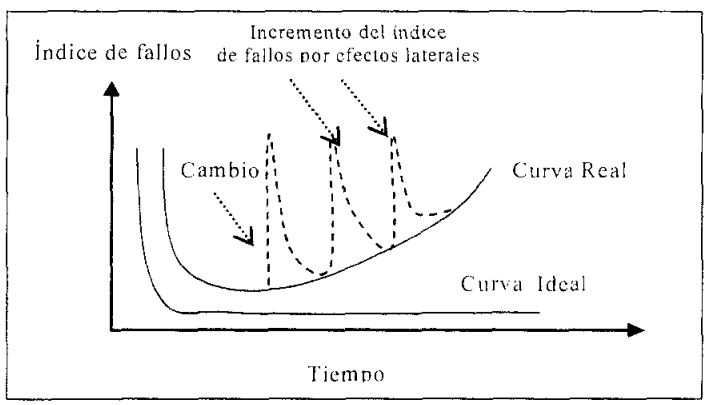
Fuente: elaboración en base a Pressman:2002:6 Figura 1.1

[18] Véase Gonzalo C. Agustín, (1991), Ingeniería del Software: Práctica de programación. Editorial RA-MA, Serie Paradigma. Mark, N. y P. Rigby (1994), Ingeniería del Software, Editorial BT-Gran Bretaña, Megabyte; Meyer, Bertrand (1997) Construcción de Software orientado a objetos. 2 edición, traducción de Katrib M., Miguel; García B., Rafael; Sánchez, Salvador, editorial Prentice-All.

La curva de desarrollo de la industria del hardware ha ingresado en una fase de cierto grado de estabilización, lo cual no ocurre con el ciclo de vida del software[19]. Por ejemplo, en la medida que se desarrolla o se le da mantenimiento al software, éste acumula una serie de cambios que terminan por alterar el sistema de algoritmos (problemas de configuración y operabilidad) y, conforme se acumulan dichos cambios, es probable que se introduzcan nuevos errores (problemas de intraoperabilidad), implicando que la curva de desarrollo se modifique, que junto con los defectos originales no descubiertos o ignorados (aflicción del software) harán que falle durante las primeras etapas de la vida del programa; una vez que estos se corrigen (parches), suponiendo que no se introduzcan nuevos errores, la curva se aplanará, sin embargo, con el tiempo se solicitarán modificaciones (actualizaciones y/o mantenimiento) como resultado de nuevas necesidades no previstas o que surgieron con la implementación de nuevo hardware o software, generándose nuevos picos de fallos (ver grafica 3).

Grafica 3

Índice de fallos en el desarrollo de Software



Fuente: Pressman:2002:6 Figura 1.2

En la grafica 3 damos cuenta del índice de fallos posibles que están ocultos y son contingentes al proceso de trabajo. Mientras que en el hardware, sí se deteriora una pieza, se sustituye por una nueva e incluso se incrementa la potencia del mismo –en caso de una pieza como microchip, memoria más potente, etc.-, en cambio para el software no hay piezas de repuesto disponibles. Cada fallo oculto en el software indica un error en el diseño o en el proceso de trabajo. Por otro lado, el mantenimiento y actualización del software supone una

[19] "Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software" Norma IEEE 1074. Existen heterogéneos ciclos de vida, que la Ingeniería del Software califica como: modelo en cascada, modelo incremental, modelo en espiral, modelo de prototipado, etc.

complejidad considerablemente mayor que el mantenimiento del hardware. Para el caso del software a la medida, se crea acorde a los requerimientos del usuario o cliente, no es que se ensamblen códigos preexistentes, sino que se crean algoritmos específicos a las necesidades del cliente (en este sentido el software a la medida representa innovaciones especificas); en este sentido la figura 3 es una simplificación de los modelos reales de fallos y errores en el proceso de desarrollo de un programa informático que no se rompe, pero se deteriora. Otro argumento de la coexistencia de contingencias, incertidumbres y fallos en el proceso de trabajo, es la falta de socialización del conocimiento, una deficiente acumulación de conocimientos tácitos y limitados flujos de aprendizaje en la programación.

Hoch et. al. (1999) señala que el origen de la industria del software es reciente, puede situarse hacia mediados de la década de los cincuenta (1955-1960) fecha en la cual se fundó la primera empresa de software independiente denominada Computer Usage Company (CUC); en estas fechas los programas de software fueron desarrollados y distribuidos gratuitamente por los fabricantes de hardware (software empotrado) o bien se desarrollaban en áreas especializadas de los clientes (software embebido), quienes ocasionalmente elaborar el sistema operativo que pone en funcionamiento las computadoras (inicios del software a la medida). En esta década se intercambiaba gratuitamente el software desarrollado (comienzo del software privado) entre los grupos de usuarios. Por ejemplo la asociación SHARE de usuarios de IBM, o bien la asociación USE para usuarios de Univac, entre otras asociaciones, intercambiaban y facilitaban el acceso gratuito de programas e información (inicios del software libre) entre los usuarios. Hacia mediados de los sesenta y principios de los setenta, se incrementan las ventas de hardware empresarial, empezando a crearse una demanda insatisfecha por parte de los usuarios que no contaban con recursos o capacidades tecnológicas para producir su propio software o establecer un convenio del tipo SHARE con IBM. Empresas como CUC y otras que proseguían una trayectoria similar, aparecieron en la segunda mitad de los cincuenta y comienzos de los sesenta para cubrir la naciente demanda de software; iniciándose lo que Hoch et. al. (1999) definen como "la era de los servicios profesionales a la medida" que se inicia con las primeras empresas de software que hacían desarrollos a la medida, específicos y adaptados a las necesidades de cada cliente. Empresas que se desempeñaban como subcontratistas de fabricantes de computadoras o de grandes usuarios (tanto del gobierno

como del sector privado) que subcontrataban parte de sus actividades de desarrollo[20]. Esta etapa entre 1950 y 1960 se caracterizó por:

- Robustecimiento tecnológico del hardware. Mejorías en el diseño, así como en funciones; mientras que el software se veía simplemente como un añadido.
- La mayor parte del hardware se dedicaba a la ejecución de un único programa que a su vez se dirigía a una aplicación específica (terminales para una sola tarea)
- La programación de software se consideraba un arte de "andar por casa". No existían métodos, herramientas, técnicas de planificación, control de calidad, etc.
- El software se realizaba prácticamente sin ninguna planificación, aunque empiezan a aparecer algunos indicios de control.
- El software se diseñaba a la medida para cada aplicación y tenía una distribución relativamente pequeña y concentrada en unas cuantas empresas.
- El proceso de trabajo en el desarrollo de programas informáticos era individual "programador solitario" o en colaboración con unas cuantas personas.
- El programador solitario era quien diseñaba, escribía, documentaba, ejecutaba, implementaba, depuraba, daba mantenimiento y actualizaba el software.
- El programador acumulaba y concentraba para sí el proceso del saber-hacer el software. Ante la escasa rotación laboral de este tipo de empleados, las empresas no se interesaban en el *que hacer* y *como hacer* en el proceso de creación del software.
- El entorno de *tipo artesanal* significaba que el desarrollo del software, el conocimiento del *saber hacer*, y *saber que hacer* estaba impregnado a nivel cognitivo en el programador, quien no documentaba suficientemente los procesos, bajo el argumento de "quien va a querer leer esto" o bien "para que documento el proceso, si soy yo el que va ha darle mantenimiento".

Hacia mediados de la década de los sesenta y principios de los setenta, se sucedió una segunda etapa en la industria del software, que se inició en 1964 cuando ADR, una pequeña firma creada en 1959 por un grupo de programadores, comercializó dos mil programas del

[20] Chudnovsky, D. (1986), "Los servicios informáticos y de telecomunicaciones en América Latina: las cuestiones que requieren ser investigadas", CET, Buenos Aires.; Hoch, D., C. Roeding, G. Purkert y S. Lindner (1999), Secrets of Software Success. Managements Insights from 100 Software Firms around the World, Harvard Business School Press, Boston; Campbell-Kelly, M. (1995), "Development and structure of the international software industry, 1950-1990", Business and Economic History, Volumen 24, N° 2, Winter.

software Autoflow, -desarrollado por encargo del fabricante de computadoras RCA Corporation, cliente que abandono el proyecto- competidor del software share de IBM, que se ofrecía embebido gratuitamente en el hardware. La empresa ADR, junto con CUC fueron las primeras empresas en independizar el software del hardware, convirtiéndose en las primeras empresas que tuvieron como negocio un "conjunto de símbolos" (Campbell-Kelly, 1995). El mercado del software se contextualiza en este periodo por una creciente complejidad en el hardware como consecuencia de una acelerada tasa de cambio tecnológico; presencia de escasos lenguajes de programación (Pascal, Fortran y Cobol) que, además de estar limitada la socialización del conocimiento, no existía una comunicación eficiente entre los programadores; además el sistema operativo que se requería para que se ejecutará el software en el hardware, era escaso; limitaciones que complican cada vez más a los usuarios de hardware desarrollar sus propios sistemas de software que necesitaban; labor que estuvo en manos de los fabricantes de hardware, quienes desarrollaban y proveían el software como un añadido (software empotrado) entre las herramientas del hardware que se ofrecía. Aún hoy día es común adquirir una computadora y que ésta sea proveída por un conjunto de programas informáticos, que por lo regular pertenecen a la empresa global Microsoft, proceso monopólico denominado como "cuota Windows", ya que existe software libre (*open source*) , que ofrece similares actividades que Windows.

Otra limitación en el desarrollo de software es que hacia mediados de los años cincuenta los programas se escribían en un lenguaje de programación de bajo nivel, llamado ensamblador[21] el cual proveía un conjunto de instrucciones básicas que debían ser traducidas a lenguaje máquina (código binario) por los compiladores. Un avance importante durante este período fue el progresivo desarrollo de lenguajes de alto nivel que reemplazaron progresivamente al lenguaje ensamblador. Un lenguaje de alto nivel ofrece a los programadores de software una sintaxis más natural y un vocabulario más cercano al ambiente del desarrollo. Los primeros lenguajes de alto nivel fueron: Fortran desarrollado por IBM (1954-1957), Cobol o RPG a principios de los sesenta, lenguajes que facilitaron la reducción de los costos "en casa" (desarrollo en casa) iniciándose la "independencia" entre software y el

[21] Assembler es uno de los primeros lenguajes binarios compuesto por 0 y 1, éste es el lenguaje que "habla" el hardware. Actualmente los compiladores "traducen" los algoritmos escritos en cualquiera de los más de dos mil lenguajes de programación al código máquina que "habla" el hardware".

hardware (Steinmueller, 1996, citado en Chudnovsky, 2001). Entre las principales características de la segunda era del software se pueden señalar:

- Avance de la multiprogramación, sistemas multiusuarios, introduciéndose nuevos conceptos de interacción hombre/máquina (ergonomía del software).
- Las técnicas interactivas abrieron un nuevo mundo de aplicaciones y nuevos niveles de sofisticación del hardware y software.
- Los avances en los dispositivos de almacenamiento en línea, condujeron a la primera generación de sistemas de gestión de base de datos.
- El software comienza como "producto inmaterial embebido" y empieza a independizarse del hardware con el advenimiento de las primeras "casas desarrolladoras de software a la medida".
- Los productos de software comprados al exterior añadieron cientos de miles de nuevas necesidades al software nacional (interoperabilidad, mantenimiento y actualización).
- En su mayor parte, las aplicaciones fuente (software añadido, empotrado y embebido) tenían que ser corregidos cuando se detectaban fallos y/o se modificaban por cambios en los requerimientos del usuario o se adaptaban a un nuevo hardware.

La tercera etapa en el desarrollo de la industria del software, entre los setenta y principios de los ochenta, se favoreció por la conjunción de algunos factores que contribuyeron a la consolidación de la industria: i) Mejora en las capacidades de procesamiento del hardware y una mayor optimización de los circuitos integrados y, difusión del primer microprocesador de Intel en 1971, lo cual redujo el tamaño del hardware e hizo mas eficiente la relación precio/potencia); ii) mayor complejidad de las aplicaciones, donde sobresalió el debate en torno a la calidad, eficiencia, entrega a tiempo, cumplimiento de contratos, etc.; la *aflicción del software* hacia fines de los sesenta, implicó el principio de una nueva disciplina: la Ingeniería de Software; iii) aparición de la serie económica de computadoras personales IBM/360, la primera "familia" de computadoras que podía usar software y periféricos intercambiables, al poco tiempo se convirtió en un estándar de facto en la industria; iv) la decisión de IBM en 1969 de vender separadamente el software del hardware que producía, v) se inicia la independencia del software con respecto al software.

Entre los primeros protagonistas de esta etapa estuvieron las firmas subcontratistas y proveedoras de servicios a la medida, de los años 1950 y 1960, que a través de diversas

estrategias intentaron incursionar en el negocio de productos de software; sin embargo, no todas lograron hacer una transición exitosa, y muchas de ellas atravesaron problemas financieros que culminaron en quiebras, como CUC que enfrentaron pérdidas durante la segunda mitad de los años setenta para entrar en bancarrota en 1986; o bien las fusiones y adquisiciones, como es el caso de SDC, que en 1979 fue adquirida por Burroughs Corporation.

En este periodo coexistió una serie de fusiones, quiebras y adquisiciones entre las primeras empresas desarrolladoras de software a la medida, arribó un tipo de programas de software innovador, que se situó en los complejos productivos industriales, que hoy conocemos como Enterprise Resources Planning (ERP). La empresa SAP fue una de las pioneras y que actualmente es líder en el segmento mundial de los ERP. SAP se instituyó en 1972 en Alemania con la idea de desarrollar un software estandarizado que administrara las distintas fases del proceso de negocios, desde el planeamiento de recursos humanos, administración de la planta, pasando por la logística, hasta el manejo de inventarios. Más adelante, empresas como Oracle (1977) y Baan (1978) se incorporarían a este segmento de mercado. Los sistemas ERP son el producto "estrella" en el mercado de soluciones empresariales. Existen otros productos que forman parte de este segmento como son el software para el manejo de mercancías "suply chain", herramientas de soporte para toma de decisiones, administración y operación de call centers, etc. Esta etapa se caracteriza por:

- Los sistemas de programación distribuidos incrementaron notablemente la complejidad de los sistemas informáticos.
- La creciente demanda de acceso instantáneo a los datos supusieron una fuerte presión sobre los programadores de software.
- Eficiencia, optimización y abaratamiento de los microprocesadores y computadoras.
- Mayor presencia y variedad de software a la medida.
- Mayor tendencia a la estandarización y atomización de los componentes del hardware, iniciada por Michael Dell.
- Mayor diversificación de software, pero sin haber una disminución en costos y sin una presencia de un control eficiente de fallos y errores en el proceso de trabajo.
- Mayor presencia del software libre (*open source* y *free software)* en sectores como el educativo, gobiernos locales, empresas e instituciones de menor tamaño e Internet.
- Se inicia un debate en torno a las comunidades virtuales de software.

Hacia fines de los ochenta y con mayor fuerza en los noventa, se inicia el cuarto periodo en el desarrollo de la industria del software, que se caracteriza por un amplio despliegue de hardware inducido por el uso intensivo de las tecnologías de la información y comunicación; como por el progreso del sector de las telecomunicaciones, televisión, radio, telefonía y por el nuevo sector de las comunicaciones, Internet que se instituye como la columna vertebral de los nuevos renglones tecnológicos como el comercio electrónico, diseño electrónico de paginas Web, diseño virtual, e-educación, e-gobierno, etc.. Despliegue que propulsó una progresiva demanda de software. Ésta creciente demanda de software fue reforzada por la masificación, atomización y estandarización de los componentes del hardware, así como por la consolidación de una plataforma estándar para computadoras personales: el sistema operativo MS-DOS de Microsoft, añadido en las plataformas tecnológicas de las computadoras personales de IBM, que habilitó importantes economías de escala en la producción de software empaquetado para un mercado masivo, por ejemplo el software estándar de Microsoft ó el ERP de SUN.

La cada vez mayor demanda de software, requerido por un número creciente de computadoras y usuarios deseosos de desarrollar nuevas tareas influyó en una especialización creciente de las empresas desarrolladoras de software, generándose con ello nuevos tipos de software y, una creciente división del trabajo entre proveedores de software (Torrisi, 1998). Al respecto, las empresas globales de software como Lotus, Microsoft, Word Perfect, Novell, IBM, entre otras, fomentaron la compatibilidad entre diversos tipos de software en relación a la gran base instalada de productos de hardware, estos es interoperabilidad entre proveedores, como IBM, Compaq, Hewlett-Packard, etc. Acuerdos monopólicos que influyeron en la creciente demanda de software. Paralelamente y, como consecuencia de una estrategia de abaratamiento de costos, el hardware se automatizó y estandarizó (outsocurcing de partes y componentes), lo que significo economías de escala y mayores externalidades que incentivó a los usuarios a adquirir software compatible con las plataformas tecnológicas ya existentes y sistemas operativos más difundidos; proceso que contribuyó no sólo a la consolidación de monopolios del hardware (IBM, Compaq, Hewlett-Packard, etc.) sino también monopolios de software especializados (Microsoft, SUN, Oracle, etc.). Monopolios del software que poseen ventajas comparativas en relación a usuarios y fabricantes de hardware, por ejemplo la aparente premisa de que el software forma parte embebida del hardware, como si fuese un

hecho inevitable que las computadoras operen con el software de Microsoft, premisa conocida como "Cuota Windows"; o bien, que el sinónimo de ERP es SUN. Pactos monopólicos que se traducen en barreras al desarrollo de nuevas aplicaciones y, la consecuencia de un mercado de procesos y productos restringido, limitado, con fuertes restricciones a la entrada, que se erige estructuralmente como un *modo de producción de software privado*, el cual es costoso porque se cobran patentes de uso, el conocimiento no fluye, la información en el desarrollo del software se "encripta", el propio proceso de trabajo creador se encierra en una "caja negra". Situación que se complejiza hacia mediados de los noventa, ante la mayor presencia de empresas de software basado en código libre (*open source*), como Linux, Fiere Fox, Ubuntu. Gnome, etc., empresas que cada vez tienen mayor presencia en el mercado del software, primero en el sector académico, después en empresas virtuales como Goggle, Amazon.com, wikipedia.com etc.; y en países, empresas, gobiernos e instituciones de la Comunidad Económica Europea (libro Blanco), Universidad de Berkeley, empresas como IBM; Nokia etc; Gobiernos como el de Inglaterra, Madrid, el Distrito Federal, etc. Este *modo de producción de software libre* hace frente a las barreras de entrada del software privado. Mas adelante abordaremos éste debate, por el momento es importante insistir en la complejidad de la industria del software hacia mediados de los noventa, que se puede resumir en las siguientes características:

- Presencia de más de dos mil lenguajes de programación.
- Existencia de sistemas expertos y software de inteligencia artificial que se han trasladado del laboratorio a aplicaciones prácticas, cotidianas.
- Persistencia de la aflicción del Software, la mayor sofisticación del hardware ha implicado el desarrollo de software complejo, con más utilerías e interoperabilidad, sin que ello implique un software exento de fallos o con cero errores.
- La demanda actual de software exige un diseño eficiente y, uso óptimo de los recursos internos, es decir mayor intra e inter operabilidad.
- Presencia de otras ingenierías, como la Ingeniería de la Usabilidad, Ingeniería de Requisitos, Arquitectura del Software, Ergonomía del Software, etc.

Los factores importantes para determinar el tipo de aplicación del software son el contenido y la determinación de la información de salida, el primero hace referencia al

significado y la entrada-salida de información; el segundo representa la necesidad de predecir el orden y tiempo de llegada de los datos. Al respecto autores como Meyer (1997), Cuevas (1991), Norris y Rugby (1997), señalan que es complicado definir categorías genéricas para la aplicación del software; sin embargo se pueden categorizar algunos tipos potenciales de aplicación de software en:

- Software de sistemas: Conjunto de programas que han sido escritos para servir a otros programas. Algunos procesan estructuras de información, complejas pero determinadas y otras indeterminadas. Se caracteriza por una fuerte interacción con el hardware, lo utilizan múltiples usuarios, operación concurrente, comparten recursos, gestión de procesos, estructuras de datos complejas y múltiples interfaces externas.
- Software de tiempo real: mide, analiza y controla sucesos del mundo real conforme ocurren diversos elementos. Elementos son la adquisición de datos, recolección de la información recibida para su análisis. El análisis es la transformación de la información adquirida; el control responde al exterior; y la monitorización coordina lo demás para mantener la respuesta en tiempo real. Un sistema así debe responder dentro de un límite estricto de tiempo.
- Software de gestión: reestructura los datos existentes en orden para facilitar las operaciones comerciales o gestionar la toma de decisiones. Tareas: procesamiento de datos y cálculo interactivo.
- Software de ingeniería y científico: caracterizado por algoritmos complejos de números. Actualmente se han tomado características del tiempo real y de sistemas.
- Software empotrado: reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. Ejecuta funciones limitadas y curiosas o significativas y con capacidad de control.
- Software de computadoras personales: el mercado de éste nació en la década de 1980, hay cientos de aplicaciones y este tipo de software sigue representando uno de los diseños más innovadores.
- Software de inteligencia artificial: hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis

Otra forma de agrupar y condensar el desarrollo de software, es el propuesto por IDC (Internacional Data Corporation) en abril de 2003, que puede ser resumido en tres segmentos:

- Software de sistema y utilitarios: incluyen sistemas operativos, lenguajes de programación, herramientas de medición del performance de los sistemas, programas de mantenimiento y seguridad, convertidores, sistemas para el manejo de redes, etc.
- Herramientas de aplicación: incluyen todos los programas que les permiten a los usuarios recuperar, organizar, administrar y manipular datos y bases de datos. Este grupo se divide en cuatro grandes categorías: recuperación y acceso a datos; administración de datos; manipulación de datos; diseño y desarrollo de programas. Incluye sistemas de administración de base de datos, sistemas de soporte e información para la toma de decisiones, planillas de cálculo, herramientas CASE (Computer-Aided Software Engineering), etc.
- Soluciones de aplicación a la medida: son programas diseñados para ofrecer soluciones a problemas propios de una industria, o bien para desempeñar una función específica de los negocios. Este software puede ocuparse de funciones "cross industry" como la contabilidad, manejo de recursos humanos, pay roll, administración de proyectos, procesamiento de texto y otras actividades de oficina; también brindar soluciones específicas para mercados verticales, por ejemplo, bancos y sector financiero, manufactura, salud, exploración y explotación de recursos naturales, etc. Obviamente algunos de estos programas como los procesadores de texto, pueden ser usados también por usuarios particulares (Chudnovsky, López y Melitsko, 2001:15).

2.3.- Riesgo e incertidumbre en el software

Con respecto a la persistencia del riesgo e incertidumbre en el desarrollo el software a pesar de la existencia de la Ingeniería del Software, se ha integrado una segunda corriente de académicos y empresarios europeos (esto no quiere decir que no haya estadounidenses) que critican el método general de la Ingeniería del Software, en el sentido que ésta se ha centrado casi exclusivamente en los atributos y particularidades del software, y ha descuidado al cliente como usuario final, se ha centrado más en temas relacionados con el sistema de operación

interna, con el rendimiento y fiabilidad del sistema; es decir se ha centrado en aquellos factores que pueden ser medidos implícitamente como: numero de errores por cada numero de líneas de programación; tiempos de respuesta y probabilidades de error por cada determinado numero de código, etc.; se avanzo en aspectos "cuantitativos", deterministas, verticales que intentan solucionar la calidad del proceso de trabajo a partir de normas disciplinares y herramientas procedimentales de análisis, diseño, codificación y pruebas rígidos; revisiones técnicas formales -olvidándose de las informales-; estrategia de pruebas multi-escalada; haciendo a un lado el hecho que el programador tiene la potestad del *saber hacer* con respecto a la creación del algoritmo, tipo de pruebas, control de la documentación y comentarios, así como de los cambios realizados. Se olvida la ingeniería del software que, quien comenta, describe o califica el proceso es el programador, paralelamente al proceso de generación del código; tiene la obligación de ajustarse a estándares de calidad y mecanismos de medida en la configuración del código, pero también la libertad de resistirse, boicotear y, en algunos casos sabotear de diversas formas el proceso de creación del código; es decir la Ingeniería del Software olvidó y relegó los aspectos subjetivos de los programadores, ignoró las normas y valores de la subjetividad laboral. Se hicieron a un lado las representaciones, intereses, sentimientos, valoraciones formales e informales que dan sentido al trabajo, por parte de los programadores. Si bien es cierto en el proceso de trabajo el cliente o usuario final es importante, ello no quiere decir que la Ingeniería del software debe construir todo un conjunto de herramental procedimentales como métricas de calidad, metodologías en el proceso, etc.: existencia de mas de cine plataformas tecnológicas, mas de dos mil lenguajes de programación; etc. Así como la llegada de un conjunto de normas disciplinares que se constituyen como nuevas profesiones: arquitectura del software, ingeniería de la usabilidad, Ingeniería de requisitos, entre otras, que parecería ser que repiten el error de la ingeniería del software, concentrarse en el desarrollo "administrativo" del software, ignorando el aspecto creativo, simbólico, cognitivo del cliente mismo, así como de los programadores.

Consideramos que las normas disciplinares y herramientas procedimentales intentan construir una serie de tácticas para hacer eficiente el desarrollo del software, pero partiendo de una transposición de los conceptos, herramientas y normas de la ingeniería tradicional, así como de los procedimientos administrativos de la manufactura; pero ignorando las particularidades subjetivas de los actores a nivel individual, que intervienen en el desarrollo

del software a nivel de la pantalla de la computadora. En otras palabras, los actores individuales que intervienen (cliente, usuarios, gerente, líder, programadores, tester, entre otros) son representantes activos y creativos que han sido relegados, olvidados como individuos claves en el proceso de trabajo; es decir, el cliente es reducido a un evaluador, otorgándosele un trato periférico, tangencial; el programador es desdeñado, despreciado como simple "mono codificador" (sobre todo en las fabricas de software, aunque es menor este sentimiento en el desarrollo del software a la medida); el gerente como un entrevistador con el cliente, el líder de proyecto, como un "interprete" del diseño generado por el gerente. De tal forma que podaríamos decir que la Ingeniería del Software ha sido cuasi-vertical en sus propuestas, considerando marginalmente al cliente y al mismo programador.

Como ya señalamos la Ingeniería del Software y profesiones similares plantean un debate no acabado en torno a examinar distintas estrategias metodológicas que resulten en un desarrollo de software organizado, disciplinado y eficiente, es decir parecería ser que –hasta este momento- la ingeniería del software intenta explorar las posibilidades de "único mejor camino" en el desarrollo del proceso de trabajo del software, segmentación del *saber-hacer* que ha intentado por distinto métodos (espiral, cascada, etc.) o bien implementado métricas de calidad (CMM, ISO, IIEE, etc.); dividir la industria del software para minimizar la incertidumbre o riesgo (fabricas de software, software encapsulado, software a la medida. etc.), así como con el advenimiento de nuevas profesiones (Arquitecto de Software, Ingeniero de Requisitos, etc.). Sin embargo, hoy en día el proceso de trabajo en el software continúa contextualizado en un ámbito de incertidumbre, de riesgo, e incluso en una atmósfera de alto contenido tecnológico con procesos de trabajo del tipo "maestro-aprendiz".

2.4.- La Ingeniería del Software como norma disciplinaria

Diversos especialistas en Ingeniería del Software, como Gonzalo C. Agustín, (1991); Mark, N. y P. Rigby (1994); García, R. y Sánchez, S. (1993) han propuesto un conjunto de herramientas metodológicas que tienen por objetivo construir métodos y métricas de programación estructurada de software; procesos de verificación de calidad; reutilización de programas o módulos; ciclo de vida más eficientes; entornos de desarrollo seguros; etcétera. Sin embargo, hasta el momento el conjunto de soluciones citadas y diseñadas por la Ingeniería del Software y ciencias a fines no ha sido del todo suficiente y eficiente como para ofrecer o proporcionar

una solución que supere la incertidumbre y contingencia en el desarrollo de un programa informático. Esto es que, a más de 35 años que hizo presencia la Ingeniería del Software, no se ha consolidado del todo un conjunto de herramientas procedimentales y normas disciplinares. Quizá, debido en gran parte a las complejidades que subyacen en cada proyecto de software; lo cual torna hasta el momento, imposible de unificar en un conjunto de herramientas y paradigmas de software las inagotables singularidades y excepciones que constituyen la norma habitual del desarrollo de software a la medida. Entre las características que han impulsado el desarrollo de la Ingeniería del Software destacan:

- El incremento potencial del hardware es inverso al costo del mismo. Es decir, mientras que el hardware ha experimentado una tendencia importante a la baja en el costo, la potencia del mismo se incrementa exponencialmente.
- El software ha experimentado una mayor complejidad en los sistemas y el costo del mismo se ha incrementado.
- En la medida que un proyecto de software es complejo y requiere de numerosos módulos e interacciona con diversos tipos de hardware, no sólo el costo es mayor, sino que ha mayor complejidad mayores son las probabilidad de fallos y errores en el sistema, así como incumplimiento en tiempos de entrega.
- El coste de inversión en sistemas informáticos es mayor en software que en hardware.
- El diseño y desarrollo de los programas informáticos, no cumple los tiempos de entrega, se exceden los costos especificados, no hay software libre de fallos y errores.
- Cuanto más complejo es el programa a desarrollar (especialización) más laborioso se torna la comprensión cognitiva de los requisitos a desarrollar.
- Los programas de software son dinámicos, no se deterioran con el tiempo, pero se tornan incompatibles ante el desarrollo de nuevos componentes. Lo cual significa dos cosas: se actualizan o se desechan.
- El problema mayor en un proceso de actualizar o darle mantenimiento a un programa de software, radica en que en la mayoría de los casos no se posee el manual de codificación, lo cual hace imposible su reprogramación. En el mejor de los casos, la empresa que desarrollo el programa que se pretende actualizar ya no existe o bien los programadores que hicieron el software ya no laboran en la empresa.

La "norma" en el proceso de trabajo del software parte de una perspectiva "artesanal", en el sentido que las empresas establecen sus propios métodos de medición de calidad en el proceso y en el producto; hasta aquellas que emplean métodos internacionales de calidad, del tipo CMM, ISO, etc. Por ejemplo, en España, según datos de European Software Institute, ESI (http://www.esi.es/), el 28% de las empresas de software no utilizan ningún modelo de calidad, 62% utiliza el modelo tradicional ISO 9001, 7% SPICE y sólo 3% CMM (Modelo de Capacidad de Madurez). Para el caso de México, según datos del programa nacional del software (PROSOFT) para 2004 -de una muestra de 364 empresas de software localizadas en las zonas metropolitanas de Nuevo León y Distrito Federal-, 42% declaró no utilizar ningún tipo de metodología; 28% respondió que sus procesos de calidad no corresponden a ningún estándar de los señalados (CMM, ISO 9000) y, sólo 6.3% declaró tener o estar en un proceso de evaluación en CMM o ISO 9001. En otras palabras, aún con herramientas de cuarta generación en el desarrollo de software a la medida, el gran ausente es el control de calidad y la aplicación de métodos eficientes en el desarrollo de software.

En una primera aproximación al programador del tipo "artesanal", solitario, aislado y sin metodología de los sesenta; podemos dar cuenta que éste ha sido sustituido hacia fines de los ochenta por un equipo de programadores, especializados en un conjunto de herramientas, métodos y métricas de calidad; procedimientos que conducen a desarrollar un programa informático en forma ordenada, disciplinada y sistemática (objetivo de la Ingeniería del Software). Sin embargo, aún con todo el conjunto de herramientas, métodos, métricas de configuración, etc., las preguntas de la era artesanal del programador del software no han tenido una respuesta satisfactoria:

- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué son tal elevados los costos del desarrollo?
- ¿Por qué no se pueden detectar los errores en el programa antes de entregarlo?
- ¿Por qué resulta difícil comprobar la calidad y eficiencia del proyecto mientras se desarrolla el Software?
- ¿Por qué no se cumplen con las expectativas y tiempos de entrega para los clientes?
- ¿Porque el programador no documenta el proceso del *como y porque* configuró los algoritmos de determinada rutina?

Estas preguntas, con poco mas de 35 años de estar presentes en la Ingeniería del Software, continúan hoy y constituyen parte medular de la *aflicción del Software*, que se traduce en errores, fallos, ineficiencias, improductividades, incumplimientos en tiempos y costos etc.. Singularidades que a través del tiempo han sido sobrellevadas mediante un conjunto de herramientas y normas procedimientales que constituyen el cuerpo normativo de la Ingeniería del Software, definida hacia fines de los ochenta como una "*tecnología multicapa, que integra distintos enfoques de calidad, procesos, métodos y herramientas*" (Pressman, 2002:14). La capa de procesos es la que da cohesión a la Ingeniería del Software, porque integra el área de gestión de proyectos y establece el contexto en el que se aplicarán los métodos requeridos, de los que a su vez se derivarán los modelos, documentos, datos, informes, formularios, diagramas, métricas, etc. (Pressman, 2002:14). El cuerpo teórico, de métodos, normas, procedimientos, métricas, etc. de la Ingeniería del Software, es mucho mas amplio que una ingenua definición, va más allá de lo no escrito; por ejemplo con la dificultad de especificar o aclarar la persistencia de la *aflicción del software*; la imposibilidad de implementar un "sólo mejor camino"; la nulidad de la existencia de una sola metodología, de un único proceso de calidad; la dificultad de garantizar entregas a tiempo, la complicación de resolver con "cero defectos" el software, etc.

La Ingeniería del Software representa una serie de herramientas, procedimientos y normas para regular el trabajo; pero son sumamente heterogéneas y en algunos casos precarias, por ejemplo existen cientos de plataformas tecnológicas, por citar algunas como OS/2; Windows; Windows CE; Palm OS; Unix; Unixbase OS; Linux; Apple Mac OS X 10.1.5; Apple Mac OS X 9.2.2; Sistema Mini Computer; Sistema Mainframe; etc.; complejos lenguajes de programación, que a la fecha suman más de dos mil, como Visual Basic; b/Com; C; C++; Cobol; Xbase; Power builder; SQL Server; Servd WWW; Delphi; HTML; XML Java; Java Script; Active x; Latte; Visual Café; SQL; RPG; Clipper; Progress; Oracle; Basic; Fox Pro; Dbase; Sybase; Informix; DB2; ERP; Macromedia, Firmware; Macro Excel; etc.); complejas bases de datos, como Sybase; Oracle; Informix; DB2; MySQL; Dbase; VMS; MS SQLServer; SQL Base; Ingres; Progress; ODBC; JDBC; Pervasive; Clipper; Excel; etc. aunado a una serie de métodos, métricas y proceso formales variados y distintos que ofrecen seguimiento a la calidad, cumplimiento de los requerimientos del cliente, minimizar la contingencia en el desarrollo de software. Sin embargo, paralelamente coexisten una serie de

prácticas individuales y colectivas que se resisten a la implementación de herramientas procedimentales y normas disciplinares e instauración de regulaciones que rompan con las características artesanales entre los programadores; el carácter artesanal se explica por la combinación que representa la creación de software, por un lado técnicas y procedimientos formales y por otro, un conjunto de habilidades y destrezas que combinan aprendizaje tácito y explícito en el desarrollo lógico de los algoritmos. Esta polémica está presente en la descripción que hace el estándar IEEE[22] al definir a la Ingeniería del Software como "*la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software*" (Pressman:2002:35); enunciación que intenta señalar orden a través e la disciplina y control mediante cuantificar el proceso de trabajo; declaración que busca romper con el grado de dependencia de la producción del software del conjunto de conocimientos, habilidades, destrezas y reflexiones de los programadores; proceso cognitivo que no se sujeta a tiempos y movimientos, disciplinados y/o jerárquicos que se puedan regular desde la gerencia; por el contrario hace referencia a "tiempos y pensamientos" cognitivos generados entre el conjunto de programadores.

La Ingeniería del Software constituye un amplio conjunto de normas disciplinares, herramientas procedimientales –cada uno con complejas y heterogéneas propuestas- que condensan una serie de técnicas y disciplinas que fortalecen el desarrollo óptimo del programa de software; por ejemplo el conjunto de "paradigmas de desarrollo del software" o "ciclos de vida" que la Ingeniería del Software intenta implementar de inicio a fin en el desarrollo de software. La Ingeniería del Software, parte de la hipótesis general de que todo proyecto de software que no cuente con un ciclo de vida o paradigma de desarrollo está condenado al fracaso; razonamiento rígido que concibe el desarrollo de software como un proceso por etapas preestablecidas, como requerimientos, diseño de módulos, codificación del diseño, prueba y "ensamble de módulos"; por último revisión y pruebas de calidad y, bajo el supuesto de que no existe otra forma posible. Y en caso de no seguir las reglas, métodos y procesos normativizados no se puede planificar, estimar los costos ó vigilar el proceso de trabajo; menos aún señalar tiempos de entrega. La Ingeniería del Software intenta establecer que al inicio de un proyecto de software, se debe elegir el ciclo de vida que seguirá el proceso de

[22] IEEE: Standards Collection: Software Engineering, IEEE, Standard 610.12-1990, IEEE, 1993. Además véase Bauer, F. L. 1972, Software Engineering, Information Procesing, 71, North Holland Publishing Co.

trabajo y éste proceso o paradigma representará el procedimiento de las distintas fases, módulos, tareas y/o actividades en el desarrollo del software. Una vez conseguido lo anterior, se está en condiciones de planificar los plazos del proyecto, asignar personas a las distintas tareas, presupuestar los costos, etc.

Consideramos que éstas rigideces estructurales de la Ingeniería del Software, al concentrase en herramientas y métodos, normas y procedimientos para el desarrollo del software, hacen a un lado el diseño centrado en el cliente y al proceso cognitivo, así como el conjunto de interacciones sociales, objetivas y subjetivas que conducen al programador ha codificar determinado algoritmo o bien a que realice una documentación en forma deficiente e incompleta del programa (código ciego); que oculte información (código oculto), etc. De tal forma que, algunos ingenieros del software como Fritz Bauer (1979, citado por Pressman, 2002:14) señalan que la Ingeniería del Software es el establecimiento y uso de principios de ingeniería robusta, orientada a obtener económicamente software que sea confiable y que funcione eficientemente; es decir que la Ingeniería del software se integra por un conjunto de dispositivos claves: métodos, herramientas y procedimientos. Los métodos suministran *el cómo* construir teóricamente el software, comprenden una serie de tareas que incluyen: a) planificación y estimación de proyectos; b) análisis de los requerimientos del sistema y del software; c) diseño de estructuras de datos, arquitectura de programas y procedimientos algorítmicos; d) codificación; e) prueba y mantenimiento; entre otras. Las herramientas suministran un soporte automático o semiautomático para los métodos. Existen herramientas para soportar cada uno de los métodos. Cuando se integran las herramientas de forma que la información creada pueda ser usada por otra herramienta, se establece un sistema para el soporte del desarrollo del software llamado procedimiento. Entre los principales paradigmas más utilizadas por diversos académicos y que intentan ordenar, disciplinar y coordinar un proyecto de software destacan los siguientes: modelo lineal secuencial; modelos de construcción de prototipos; modelo de desarrollo rápido de aplicaciones; modelo evolutivo de procesos; modelo basado en tecnologías de objetos; modelo de métodos formales; técnicas de cuarta generación; tecnología de procesos. Las herramientas, por su parte, suministran un soporte automático o semiautomático para los métodos. Existen herramientas para soportar cada uno de los métodos. Los procedimientos de la ingeniería del software facilita el desarrollo racional y oportuno del software. Los procedimientos se definen como la secuencia

en la que se aplican los métodos, las entregas que se requieren (documentos, informes, formas etc.), los controles que ayudan a asegurar la calidad y coordinar los cambios así como el conjunto de guías (documentos) que facilitan a los gestores del software establecer su desarrollo. Entre los principales paradigmas que intentan ordenar la aflicción del software y que pretenden un desarrollo ordenado, sistemático y disciplinado en el proceso de trabajo del software se identifican los siguientes modelos: Modelo de construcción de prototipos; Modelo de Desarrollo Rápido de Aplicaciones (DRA); Modelos evolutivos de proceso de software; Modelo basado en tecnologías de Objetos; Modelo de métodos formales; Técnicas de cuarta generación (T4G), entre otros. (Pressman, 2002; Gonzalo, 1991; Mark y Rugby, 1994). Todo el contexto anterior se agregara en un solo concepto: herramientas procedimentales, que utilizaremos en el capitulo 7 y 8.

En términos generales, en la industria del software no se cumplen los tiempos de entrega del proyecto, se rebasan los costos programados, el mantenimiento del programa esta rodeado de incertidumbre (entiéndase por mantenimiento a la incorporación de nuevas líneas de código, para "actualizarlo" a las nuevas necesidades del usuario), la actualización constituye un incremento en los costos estimados, o bien que en el mediano plazo se deba adquirir otro software porque en gran parte de los programas de software realizados a la medida, no existen manuales o metodologías que expliquen el como y porqué se configuró determinadas líneas de código o algoritmos y, por último, existe un deficiente control en términos de la calidad, auspiciado por la inexistencia de métricas de calidad estandarizadas en la industria.

2.5.- Ingeniería del software y las singularidades de un trabajo creativo[23]

El proceso de trabajo en el desarrollo del software posee una serie de características que le diferencian de la manufactura, se trata de un proceso lógico, cognitivo, donde la materia prima son imágenes, pensamientos: *El software es un proceso creativo, no se fabrica en un sentido clásico.* Por ejemplo, supongamos que a un conjunto de programadores, el gerente de una empresa les solicita que realicen un software que automatice la nómina de los empleados de la

[23] Este documento se hizo a partir de: Gonzalo C. Agustín, (1991), Ingeniería del Software: Práctica de programación. Editorial RA-MA, Serie Paradigma. Mark, N. y P. Rigby (1994), Ingeniería del Software, Editorial BT-Gran Bretaña. Megabyte; Meyer, Bertrand (1997) Construcción de Software orientado a objetos. 2 edición, traducción de Katrib M., Miguel; Garcia B., Rafael; Sánchez, Salvador, editorial Prentice-All. McClure, C., (1993) CASE. La automatización del Software. Editorial RA-MA, Addison-Wesley Iberoamérica, Traducción de José M., Ortega.

empresa. Dicha solicitud suena muy precisa para el gerente de la empresa, sin embargo, para los programadores tan sólo representa el inicio del análisis del programa informático que se desarrollará. De tal forma que entre el analista del sistema surge una serie de preguntas: ¿Cómo se realizan los pagos por nómina: en forma semanal, quincenal o mensual?; ¿Los pagos dependen de alguna categoría dentro de un tabulador?; ¿Quién y como se determina la categoría y cuáles son las reglas para cambiar de categoría dentro del tabulador?; ¿Qué tipo de deducciones u aportaciones se incluyen en la nómina?; ¿Con qué regularidad se descuentan las deducciones o se pagan las aportaciones?; ¿Se les compensan tiempos extras, como se registran, como se estiman?; si no se pagan compensaciones, o tiempos extras o incentivos, ¿Se querrán pagar en el corto o largo plazo, tendrá estimada la empresa incorporar algún tipo de compensaciones, pretenderá la empresa la posibilidad de pagarla?; ¿Existirán comisiones, bonos de vacaciones o aguinaldos? ¿Cómo son las reglas actuales y cuales las potencialmente futuras?; en lo relativo al salario, ¿Cuál es el sueldo base y de que se integra el salario integrado?; ahora bien, ¿Cómo se realiza un aumento de sueldo?; etcétera. Este conjunto de preguntas, representan las primeras ideas, pensamientos o conjeturas que formarán parte medular del conjunto de requisitos que debe cumplir el desarrollo del software. No es extraño, que el cliente, no tenga en cuenta las diferentes dimensiones de análisis que debe abordar antes de requerir el desarrollo de un programa informático. De ahí la importancia de las primeras entrevistas entre el gerente, el analista de sistemas o programadores que desarrollarán el programa informático. De estas primeras reuniones es importante que el cliente esté de acuerdo con el conjunto de ideas o pensamientos que se convengan. De aquí se deriva un primer documento que se denomina lista de requerimientos, el cual describe los requisitos que debe cumplir el software a desarrollar.

En términos generales se observan algunas particularidades del proceso de trabajo creativo en el desarrollo de software:

1.- Si suponemos que la lista de requisitos del programa informático se convino debidamente con el cliente, de cualquier forma se plante si son todos los requisitos, o se ignoró algún requerimiento fundamental; ¿Cómo interactúan dichos requerimientos?; ¿Cómo agrupar los requerimientos en módulos o sub-módulos para su desarrollo?; ¿Cómo se garantiza que el agrupamiento de los requerimientos en módulos y la interrelación entre éstos sea la apropiada?. Este tipo de preguntas relativas al diseño se acompañan todavía de mas

preguntas técnicas relativas el desarrollo in situ: ¿Cuál es la arquitectura más apropiada para el producto?; ¿El lenguaje de programación que se utilizará es el óptimo, es el que mejor conocen los programadores?; ¿La plataforma tecnológica es la que mejor se ajusta al proyecto?. Otras preguntas relativas al método surgen: ¿Cómo estimar el tiempo suficiente?; ¿Los módulos a desarrollar poseerán una calidad aceptable?; ¿El método de desarrollo es suficientemente flexible?. Los defectos no detectados harán que falle el programa durante las primeras etapas de su implementación. Una vez que se corrigen baja el índice de fallos, mas el software se deteriora con el tiempo. El software sufre cambios, con estos vienen nuevos defectos, el software se va deteriorando debido a los cambios realizados (parches). No hay piezas de repuesto para el software, cada fallo indica un error en el diseño o en el proceso mediante el que se tradujo el diseño a código máquina ejecutable (algoritmos).

2.- El software, a diferencia de los productos manufacturados, requiere de la comprensión del usuario final y, en caso de ausencias o errores es complejo proporcionarle mantenimiento y/o actualización. Si suponemos que se desarrolló el programa informático de nomina empresarial, que se cumplió medianamente con los tiempos de entrega y costos, irrumpe otro conjunto de preguntas: ¿Se explico mediante iconos e imágenes lo suficiente el programa?; ¿El programa es accesible desde el inicio para el usuario?; ¿Están lo suficientemente documentados los procesos como para que el usuario final los comprenda?; ¿Cumplió el programa con los requisitos pactados con el cliente?; ¿En caso de ausencia de requerimientos, se pueden anexar tareas sin que ello implique problemas de configuración entre módulos?; ¿En caso de nuevos requerimientos, existe la suficiente documentación lógica de los algoritmos, como para ser reconfigurados sin que impliquen problemas de interoperabilidad entre módulos?. En el desarrollo de un programa a la medida, para una empresa es complejo subcontratar partes o módulos del sistema, no sólo porque los costos se elevarían, también por el difícil acoplamiento de los componentes de un programa, realizados por distintas personas, dificultad que esta intra-relacionada con los componentes o módulos del sistema; es decir, la operabilidad entre los distintos módulos debe estarse "probando" (testeando) continuamente. En otras palabras, los módulos interaccionan entre ellos, de tal manera que no sabemos si las modificaciones significan que algunas cosas no funcionen.

3.- Lo que se vende no es un producto, sino una licencia de uso, se concede al cliente un uso del programa, más no el acceso al contenido (utilización del conjunto de algoritmos).

A diferencia de un producto manufacturado, en el desarrollo de un programa informático, lo que adquiere el cliente y/o usuario final es un conjunto de iconos que resuelven problemas planteados por el cliente. Sin embargo, cuando el cliente adquiere el programa, se apodera de un conjunto de iconos que se realizan cuando se ejecutan en el hardware del cliente, cuando son utilizados por los empleados finales del cliente, cuando estos iconos son embebidos en el uso cotidiano del usuario final. *4.- La producción de software es, en sí misma una actividad innovadora.* El programa informático desarrollado, resuelve exclusivamente un conjunto de problemas planteados por el cliente. El software a la medida, es un programa que soluciona un conjunto de tareas ya conocidas (Torrisi:1998), sin embargo la innovación radica en el hecho que soluciona problemas específicos, es decir el grado de "originalidad", obviamente, varía con el tipo de software producido y con las tecnologías utilizadas en su desarrollo. En un extremo están las adaptaciones y/o actualizaciones y cambios menores de productos de software ya existentes. En el otro, están nuevos productos que abren mercados inexplorados (el lanzamiento de la hoja de cálculo o las páginas web, por ejemplo) o los programas o servicios creados para un cliente individual (Torrisi:1998; Chudnovsky:2001).

5.- El desarrollo de software sigue siendo aún una actividad con características artesanales[24]. Pese al avance que ha experimentado la llamada Ingeniería del Software[25], todavía siguen subsistiendo problemas de calidad, confiabilidad, cumplimiento de tiempos, etc. en los procesos de desarrollo de software[26]. Esto ha llevado, entre otras cosas, a crear nuevos modelos para la producción de software (en cascada, evolutivo, transformador, en espiral, etc.), nuevas técnicas y herramientas de programación (CASE, programación "orientada a objetos", etc.), investigar modelos de diseño flexibles que permitan la reutilización de módulos (programación orientada a objetos, lenguajes de programación de

[24] "La ingeniería de software se encuentra aún en una etapa de transición entre lo artesanal y lo 'profesional' ... no se ha logrado pasar de la etapa de manufactura casi artesanal a la de fabricación seriada con técnicas y procedimientos establecidos" (Perazzo et al, 1999, p. 18).

[25] Por ingeniería de software se entiende la definición y empleo de principios y métodos de ingeniería con el fin de obtener de manera económica software confiable y capaz de operar sobre máquinas reales, Bauer:1972

[26] Los problemas –y consecuentes costos en términos de dinero y tiempo- generados por la falta de adecuados sistemas de ingeniería de software, que garanticen calidad, confiabilidad y predictibilidad en los programas desarrollados, llevaron en 1984 a establecer en los Estados Unidos el Software Engineering Institute (SEI) en la Universidad Carnegie-Mellon, con financiamiento federal y patrocinio del Departamento de Defensa (principal usuario de software en aquel país). Entre otras iniciativas destinadas a mejorar este estado de cosas, el SEI desarrolló el llamado Capability Maturity Model (CMM). Dicho modelo describe las prácticas básicas asociadas con el desarrollo de software confiable y reusable que pueda ser creado según las restricciones de tiempo y presupuesto originalmente convenidas. Si bien fue desarrollado originalmente para firmas que trabajan con grandes contratos para el gobierno, su aplicación se puede extender a otros tipos de firmas.

cuarta generación, etc.); utilizar herramientas específicas para la administración adecuada de los tiempos de desarrollo en los programas de software (herramientas del tipo UML -Unified Modelling Language-), así como introducir estándares de calidad y gestión propios de esta industria, como el modelo CMM o el SPICE (Software Process Improvement and Capability Determination – promovido por el International Committee on Software Engineering Standards y por el European Software Institute-). Mas adelante ampliaremos este aspecto de la calidad en el proceso.

2.6.- La aflicción del software y la ingeniería del software

Glass (1998); Tiechrow (1989); Pressman (2002); entre otros, ponen en tela de duda el termino "crisis del software" utilizado en la jerga académica de los últimos años el concepto de *aflicción del software*; ya que crisis no da opciones, hace referencia a "blanco o negro", mientras que el termino "aflicción" indica "una causa o desastre muy duradero o que reaparece con frecuencia continuando indefinidamente". La *aflicción del software* no se limita al software que no funciona correctamente, sino que abarca preguntas asociadas al como se hizo el software, por ejemplo: ¿Cómo se va a desarrollar el software?, ¿Cómo mantener el volumen cada vez mayor de software?; ¿Cómo estar actualizado en los procedimientos y normas que demanda el desarrollo de software?. En resumen la aflicción del software hace referencia también al conjunto de problemas, errores y fallos localizado en el desarrollo de un programa informático; también abarca problemas asociados con: ¿Qué técnicas, herramientas y métodos utilizar para desarrollar software eficiente?. Preguntas que se originan porque:

- Se sobrepasan continuamente los costos estimados en el presupuesto inicial del proyecto.
- No se cumple la planificación, se excede en meses, incluso años.
- Se ha mejorado muy poco la productividad y eficiencia de los programadores.
- Los errores y fallos en los programas de software producen en los clientes insatisfacción y falta de confianza (incertidumbre).

Tales problemas son sólo manifestaciones más visibles de otras dificultades del Software:

- No se cuenta con bitácoras, documentos o etiquetas formales donde se explique el como y porque se seleccionaron determinadas combinaciones de algoritmos.

- Sin la documentación respectiva de los algoritmos desarrollados, proveer mantenimiento o actualización del programa informático es casi imposible.
- No se cuenta con un procedimiento formal, sistematizado para recoger los requerimientos o necesidades del cliente. Sin datos confiables como guía, la estimación y planificación es errónea y los resultados son inciertos.
- No existen herramientas formales para calcular la productividad por programador, por tanto no podemos evaluar con precisión la eficiencia de las nuevas herramientas, técnicas o estándares propuestos por la ingeniería del Software.

La insatisfacción del cliente con el programa informático desarrollado se produce frecuentemente porque:

- La documentación existente -cuando la hay- frecuentemente es inconclusa, con indefiniciones de los requisitos desarrollados.
- La comunicación entre cliente y el diseñador y programadores es escasa.
- La calidad del software es normalmente cuestionable.
- Se ha empezado a comprender recientemente la importancia de la prueba sistemática, completa del software.
- Están comenzando a emerger conceptos cuantitativos sólidos sobre la fiabilidad del software y garantías de calidad.
- El software existente puede ser muy difícil de mantener.
- Las tareas de mantenimiento del software se llevan la mayor parte de la inversión.

La *aflicción del Software* representa un conjunto de incertidumbres e insatisfacciones crecientes por parte de los clientes con respecto a la calidad, tiempos de entrega, diferencias entre costo inicial y costo final. Contingencias que contextualizan la aflicción del software que data de más de 35 años (Carnegie, 2000) y, a pesar del incremento sustancial en la tecnología para el desarrollo de software y la creciente complejidad de los proyectos, que por cierto se incrementa a un ritmo mayor que las capacidades para encontrarle solución (Alcalde, 2003). La respuesta tan prometida los últimos años acerca de mejorar la eficiencia, productividad y calidad al aplicar nuevas metodologías y tecnologías de software no llega, no se vislumbra, hecho que ha llevado a las organizaciones a plantearse que el problema fundamental es la

impericia de las mismas organizaciones en dirigir los proceso de software (Carnegie, 2000). Para 1998 la IIEE señaló que 26% de los proyectos fallaron completamente y 46% experimentaron un desbordamiento en planificación y tiempo (Citado por Reel, 1999) y los beneficios de métodos y herramientas no pudieron ser distinguidos en este remolino de proyectos sin métricas, sin metodologías, es decir, es caótico e indisciplinado el desarrollo del Software (Carnegie, 2000). Algunas explicaciones que se han dado con respecto a la aflicción del software son:

- Los problemas asociados con la aflicción del software se han producido por el carácter lógico del software y los errores de los involucrados en el desarrollo.
- El software es un elemento lógico en vez de físico. Por tanto, el éxito se mide por la calidad de una única entidad.
- El software no se rompe. Si se encuentran fallos, existe una probabilidad de que se introduzcan inadvertidamente durante el desarrollo y no se detectaran en la prueba.
- El mantenimiento incluye normalmente la corrección o modificación del diseño.
- El desafío intelectual como trabajo cognitivo que es el desarrollo del software, forma parte de una de las causas de la aflicción del software.
- Otro de los factores es la ausencia de una eficiente comunicación y falta de fluidez en la información de los contenidos del programa entre aquellos implicados en el desarrollo del software: clientes; desarrolladores; tester´s; gestores, etc.
- La operabilidad entre los módulos puede fallar por:
    - ◊ La complejidad y/o características especiales del programa.
    - ◊ Los problemas particulares asociados a cada módulo fueron reinterpretados (lista de requerimientos) erróneamente por los programadores. Cuando ocurre esto, los problemas asociados con la aflicción se multiplican.
    - ◊ Los trabajadores del software han tenido muy poco entrenamiento formal en las nuevas técnicas de desarrollo de software.
    - ◊ Malos hábitos que dan como resultado una pobre calidad y mantenimiento.
    - ◊ Resistencia al cambio por parte de los programadores a implementa nuevos métodos, plataformas tecnológicas o nuevos lenguajes de programación.

Para hacer frente a la aflicción del software, la Ingeniería del Software ha implementado una serie de estrategias metodológicas denominadas modelos de software o paradigmas de Software. Sin embargo, lo complejo y heterogéneo en las distintas rutas cognitivas para el desarrollo de un programa informático, así como la nula posibilidad de un "solo camino" en el desarrollo de software complejizan la aplicación de normas y estándares.

2.7.- La aflicción del software y el contexto de creación artesanal

Entre 1940-1960 el desarrollo de la industria informática se encontraba en un estadio "artesanal" donde la socialización del conocimiento estaba aislado, centrado en una cuantas empresas, sólo unos cuantos expertos conocían los procesos para programar en lenguajes de bajo nivel, como enssamblador o fortran (Pressman:2002). Los lenguajes de bajo nivel son aquellos que están más próximos al lenguaje maquina (código-maquina), son aquellas instrucciones algorítmicas que se procesan directamente en el circuito microprogramable del hardware, utilizando nomenclaturas binarias (0 y 1). Cuando se desarrollan los primeros sistemas operativos (MS-DOS), se disipa lentamente el uso del lenguaje ensamblador y se facilita que las grandes empresas de software innoven la programación al desarrollar *lenguajes de programación de alto nivel*, que son aquellos "más alejados" del lenguaje binario; entre los primeros están el fortran y cobol; pero es gracias al sistema operativo, el cual opera como "interprete" que "traduce" los algoritmos de los lenguajes de alto nivel al lenguaje código máquina. En este periodo los conocedores de lenguaje-máquina trabajaban largas jornadas de trabajo, no documentaban los procesos, no diseñaban prototipos, creaban por "ensayo y error", no escribían las rutinas del algoritmo; etc.; procedimientos que significaron incumplimientos en las metas y objetivos planeados, morosidad en la entrega, no se conocían los procedimientos bajo los cuales se configuraba determinado conjunto de rutinas, no se suministraba mantenimiento a los programas. Una restricción importante en el desarrollo de software en este periodo fue la casi exclusividad del software empotrado, es decir que la programación estaba orientada a una determinada computadora o un conjunto especifico de servidores, lo cual hacia restrictivo el uso o desarrollo de sistemas informáticos de uso genérico. Éstas condiciones condujo a la industria informática a un estadio denominado *aflicción del software*, con presencia no sólo de errores (problemas de configuración hardware-software, incumplimiento de requisitos, desarrollo fuera de tiempo, etc.) también

complicaciones para "conocer" el como y el porque se desarrollaban determinadas líneas de código (problemas de documentación y administración de los recursos humanos). La *aflicción del software* en la industria de la informática hace referencia a una serie de singularidades que se configuran entre 1940 y 1960: Los programas no cumplen los requisitos que solicita el cliente; los programas no se terminan tiempo y rebasan los costos señalados; no existe el suficiente personal capacitado; son insuficientes los programas académicos; etc.

La *aflicción del Software* hace referencia, en términos generales a la primer etapa en el desarrollo del software embebido, el software se consideraba como un adherido al hardware de las computadoras; la potencia del software es oscurecido por el hardware, por ejemplo la utilización del horno de microondas ó un celular invisibiliza la potencia del software, en otras palabras, el uso del software incrementa las funcionalidades del hardware en la medida que el hardware es capaz de dar soporte a un software más complejo. Entre 1970 y 1980, a medida que el software deja de ser considerado como "un añadido" empieza a desincorporarse del hardware, a objetivarse como producto y como servicio, sólo entonces se inicia una compleja separación entre el software y el hardware[27]. En este periodo artesanal la industria del software esta invisibilizada en el hardware, predomina el hecho de que "lo importante es que funcione" el programa desarrollado, el software es considerado como un solo producto hardware/software.

El proceso de trabajo en el desarrollo de software carecía de una planificación específica, de una metodología o procesos definidos; el programador diseñaba los componentes, escribía el código, lo compilaba, ejecutaba y en caso de errores, lo depuraba, lo re-escribía y volvía a ejecutarlo. Este ciclo de vida podemos denominarlo como de *"creación doméstica"* que estaba monopolizada la información y el conocimiento; la innovación y el aprendizaje que impregnaba el proceso de trabajo del programador. En este periodo de *creación doméstica* se caracteriza por la no documentación de los procesos, la no escrituración de algoritmos; etapa que denominamos como "aflicción del Software" en la cual, la forma de trabajar por parte del programador, es conocida como *"programación artesanal"*, donde los desarrolladores de sistemas informáticos detentaban el proceso del saber-hacer del sistema informático (Gartner, 2003; Pressman, 2001).

[27] En términos reales el Software, solo se objetiviza así mismo en el cuerpo físico del hardware, ya que éste solo es un conjunto de símbolos abstractos, inmateriales que se potencian en el hardware.

En la etapa artesanal el acceso al conocimiento de las líneas de código estaba restringido, por limitaciones en la socialización de la información, el aprendizaje y la obstrucción de los flujos de información. No sólo por la *creación doméstica*, también por lo incipiente de la industria misma que limita el flujo de capacidades y conocimientos para comprender las sentencias del código desarrollado por los programadores; lo cual implica que el programador de tipo artesanal, al igual que el artesano del siglo XIX del que habla Tohmpson, Edwards y otros, detentaba un *saber hacer* y este se convertía en *poder* sobre los medios de producción con los cuales interactuaba. Al respecto, un gerente del área de sistemas (DC2), de una de las empresas entrevistadas (CATI), señalaba con un aire de preocupación y enfado, que esa libertad de la cual gozaban los programadores y la concentración de conocimiento en ellos, les daba poder, ya que sólo ellos *sabían-hacer* las modificaciones en las sentencias de línea de código, y el *por que* deberían modificarlas:

> "Bueno, los errores principales de lo que es programación antigua (...) es el nombre de variables. Era un error dejar que el programador decidiera como se iban a llamar las variables (...). Esa libertad era buena para el programador por que se divertía haciéndolo, pero muy mala para el que seguía; el que seguía no entendía absolutamente nada (de lo) que narra el código que estaba viendo, y sobre todo, la otra parte importante -a parte de eso- es que como no había estándares en la programación y manejo de la documentación, pues era cualquier cantidad de problemas para que tanto el usuario final como el siguiente programador entendiera lo que estaba haciendo con esa aplicación (...) le daba poder, por que era el único que podía modificarlo. Era el único que conocía, y eso, tienes que romperlo no puede ser que alguien que te está haciendo el trabajo se convierta en el dueño de lo que te está haciendo. (...) si, pero como se hace, con mejores prácticas[28] todo el mundo lo entiende. Ahora no tienes la libertad de ponerles a las variables el nombre que quieras (Líder CATI DC2).

El periodo artesanal se complica debido a la movilidad y rotación laboral; los ejecutivos, los empresarios y dueños de empresas desarrolladoras; clientes y usuarios finales tenían incertidumbre e inseguridad de que quien desarrollaba el software, no estuviese más para facilitarle mantenimiento o actualización al software anterior que hubiesen desarrollado. La *creación doméstica* del software se consideraba mas personalizada por el programador; el diseño, desarrollo e implementación lo llevaba a cabo el programador contratado para tal fin;

[28] Mejores practicas es un *proceso* acuñado y desarrollados por pymes en contra posición a las métodos y procesos de grandes empresas, que aplican metodologías caras y burocráticas como CMM, ISO, IIEE, etc.

lo cual significaba que la documentación del desarrollo del software muchas de las veces no existía. Existen varias posibilidades del porque no documentaban el proceso los programadores, desde aquellos programadores que les gusta hacer la parte "divertida" que implica el reto de solucionar X implementación, y el resto (documentación) ya no es atractivo:

> " (...) va a haber software donde la documentación pues va a ser mínima o es muy mala o no se entiende o faltan temas, pero normalmente si existe, obviamente pues el proceso de documentación de un software es un proceso digamos tedioso, a lo mejor pues no cualquiera hace... no pues ¿que nos gusta?... pues programar, desarrollar las interfases, pero ya cuando viene la fase de la documentación, sobre todo si no hay alguien al que digamos que le interese mucho hacerla, pues... normalmente se deja aparte o a lo ultimo la documentación (Líder WALS JG 02).

También puede ser por estrategia de "co-dependencia" de la empresa con respecto al programador:

> "La mayoría de los proyectos que hago no existe (*la documentación del diseño)* y si quieren reemplazarme necesitan contratar a alguien con la misma competencia que yo y, a parte pagarle horas extras para que se entere de que es lo que yo estaba haciendo, se meta a mi código... a parte cuando se meta a mi código va a encontrar cientos de fallas que yo no había contemplado..." (Programador DINS 03).

El proceso de trabajo en el desarrollo de software a la medida fue formalizándose hacia mediados de los setenta, década en que hacen presencia las primeras casas de software[29] que ofrecen productos estándar, gracias a la introducción de la multiprogramación, técnicas interactivas, bases de datos, "bibliotecas" de software (comandos y utilerías que se pueden utilizar en varios sistemas informáticos) etc. Es en la década de los ochenta que las aplicaciones informáticas (ya sea compradas al exterior o desarrolladas localmente, o bien las bibliotecas o librerías ya integradas en algunos sistemas informáticos) presentan una serie de restricciones, errores o fallas, al momento de implementarse e interoperar con otros sistemas informáticos; o bien cuando las "casas de software" y las empresas desarrolladoras intentan proveerle mantenimiento[30] y actualización[31] a los sistemas informáticos existentes desde la

---

[29] Casas de software en los setenta, hacia fines de los noventa, se les conoce como "fábricas de software". La diferencia en el concepto es importante, ya que las primeras hacen alusión a un sistema cerrado, limitado por las prácticas en esa morada, en ese domicilio físico. A diferencia de fábrica de software, que busca reconocerse ésta, con las prácticas similares a la de una fabrica, donde existan las jerarquías laborales, división del trabajo, etc.

[30] Mantenimiento: puede ser corregir código defectuoso, eliminar líneas de código obsoletas, potenciar operaciones. etc.

década de fines de los setenta. Errores y fallas que no sólo significaron grandes recursos económicos en corregirlos, también colaboro a estas fallas, la casi nula existencia de documentación suficientemente definida que permitiera comprender el método empleado para el desarrollo de las líneas de código y proveerle mantenimiento y actualización. Es en este contexto en el cual el concepto de *aflicción del Software* cobra relevancia (Gartner, 2003). A este periodo en el cual el desarrollo de sistemas informáticos se hacía de manera artesanal, con pocas o nulas posibilidades de documentación del proceso de programación que había proseguido el programador, se le denomino *aflicción del software,* concepto que los menos trágicos, denominaron *aflicción crónica del software* que hace referencia a los errores en un contexto de éxito de algunos programas:

> "(...) Esa libertad era buena para el programador por que se divertía... pero muy mala para el que seguía, el que seguía no entendía absolutamente nada de lo que narra el código, que estaba viendo y sobre todo, la otra parte importante -a parte de eso- es que, como no había estándares en la programación y manejo de la documentación, pues era cualquier cantidad de problemas, para que tanto el usuario final como el siguiente programador entendiera lo que estaba haciendo con esa aplicación (Líder CATI DC2).

Este periodo artesanal, donde la *creación doméstica* estaba centrada en el programador, esta contextualizada por un proceso de invisibilización del trabajo del programador frente al hardware; el trabajador cognitivo detenta el *saber-hacer* como *poder* en el proceso de trabajo; el flujo de información es obstaculizado por el monopolio de la información y el conocimiento, así como de la innovación y aprendizaje del trabajador; el medio parea desarrollar el software no importaba, el centro de atención estaba en la funcionalidad del hardware, el medio no era el importante –en este caso el software como producto- sino el fin es el incremento de productividad y competitividad implícitos en la implementación de nuevo software, que potencia al hardware donde se instala (Perrini, 1989; Baethge y Oberbeck:1995).

[31] Actualización: cuando el hardware se modifica, es mas potente, puede implicar que el software se actualice en algunas operaciones que no habían sido consideradas, por ejemplo ampliación de la manufactura, área administrativa, nuevas actividades, etc. el mantenimiento y la actualización de los sistemas informáticos son considerados como dos de las áreas prioritarias en la industria del software.

2.7.1.- Superposición entre la esfera del producto y como servicio

La interposición del software como producto y proceso no es del todo exacta, sin embargo consideramos que no es trivial dicho planteamiento, debido a que la respuesta no es sencilla, aún para los expertos en el tema, por ejemplo la experta en calidad del software Hanna Oktaba (2005), señala:

> "...existen por lo menos dos versiones antagónicas de la interpretación de lo que es un proceso (de software). La escuela SEI (Software Engineree Institute) opta por preocuparse en «que hacer durante la realización de un proceso con el fin de obtener resultados deseados», mientras que la escuela ISO/IEC15504 prefiere especificar «que es lo que se propone lograr a través de un proceso»...la discusión final sobre este tema nuevamente se pospuso hasta la siguiente reunión" (Revista SG, Software Guru, Noviembre-Diciembre 2005, página 6)[32]

En la industria del software los expertos no se ponen de acuerdo en su definición. Para los analistas de Gartner (2003) señalan que quizá la industria todavía se encuentra en una fase de consolidación. Al respecto en diferentes entrevistas se señalaba que hace falta una *cultura informática* (Gerentes RF1 y DG1), o simplemente es "inmadurez de la industria misma" (Líder DC2 y RD2). Hasta el momento, diversos investigadores como Chudnovski et.al. (2001), Tikai (2003), entre otros señalan que el *software como producto,* es un bien desarrollado basado en la venta de licencias para su uso dentro de una organización o a nivel individual. En algunos casos la empresa desarrolladora provee algún tipo de servicio asociado al software (actualización de las versiones, soporte técnico, mantenimiento, etc.) que puede estar incluido dentro del contrato de licencia o comercializarse de manera independiente. Hoch et al (1999, citado por Chudnovsky et.al. 2001:4), nos indica que es posible dividir el segmento de productos de software en dos grupos: como soluciones empresariales y, como productos empaquetados de mercado masivo. La distinción entre ambos grupos va más allá del mercado al cual se dirigen (en este sentido, un procesador de texto, por ejemplo, puede comprender tanto al mercado empresarial como al hogar). Una diferencia sustancial entre un producto de mercado masivo y una solución empresarial (servicio) radica en que esta última exige, en mayor o menor medida, acorde a su complejidad, algún grado de personalización o adaptación a los requerimientos específicos de la organización en la cual va a ser implementada. En este último caso, la "puesta en marcha" de la aplicación, es decir su

[32] La información de los paréntesis es agregado nuestro. Cuarta reunión del International process research consortium (IPRC) celebrado en Irlanda en el mes de Agosto del 2005.

instalación y los ajustes necesarios para su correcto funcionamiento, suele implicar una inversión importante en términos de tiempo y dinero.

Chudnovski et.al. (2001), Tikai (2003), ambos autores advierten que el *software como servicio*, es un conjunto de actividades tan diversas como el diseño y desarrollo de soluciones a la medida, implementación y adaptación de productos de terceros, servicios de consultoría, capacitación, instalación y mantenimiento de productos de software, etc. Sin embargo diferenciar en los hechos, el software como producto y como servicio es cada vez más difícil, ya que la frontera de ambos es difusa, porosa, con similitudes y divergencias. Por ejemplo los programas de software industriales más difundidos entre el sector empresarial: Enterprise Resources Planning (ERP) se mezclan ambos procesos, donde del costo total, 30% corresponde al pago del producto y, 70% a servicios profesionales. Para este casto, el número de licencias otorgadas podría ser una medida de desempeño para una empresa de productos, mientras que en el caso de una empresa de servicios la cantidad de horas de implementación asociadas a cada proyecto sería el indicador más relevante. Al respecto, DigiWorld 2006, en su informe 2006, plantea esta frontera difusa entre ambos tipos de Software:

> Los servicios (...) han adquirido un notable peso específico para los fabricantes de software (...) Las aplicaciones informáticas se venden cada vez más como servicio ante la aparición de los modos de distribución basados en Internet como las páginas xSP (Salesforce.com). De forma paralela, el desarrollo del software libre acentúa esta tendencia, ya que los defensores del *open source* obtienen sus beneficios mayoritariamente de los servicios. (DigiWorld, 2006:66 www.enter.es)

DigiWorld, 2006, dedica parte de su análisis a estas fronteras difusas entre producto y servicio del software. Piénsese por ejemplo, en las nuevas tendencias en el desarrollo de sistemas informáticos implementados a través de Internet (programación Web), ya no representa sistemas locales monousuario, se convierte en multi-usuario, ya no hablamos de venta de un producto, en CD, ya que éste se disuelve en Internet.

> "... ahora lo nuevo es la programación en Internet. Entonces eso empieza a diferir un poquito de lo que (...) estaba acostumbrado a manejar sistemas monousuario o locales de una maquina, después tuvimos que hacer sistemas multiusuarios con C y con UNIX o luego hacer los sistemas en red de área local, y luego ya viene la parte de tratar de hacer los sistemas en ambiente en Internet que es un ambiente ya global" (programador TADI MZ3).

Sin lugar a dudas, que este nuevo enfoque global a través del desarrollo de sistemas informáticos con herramientas y soporte en Internet, está redefiniendo las relaciones con el cliente. Las empresas desarrolladoras promueven no sólo como programas de software, sino también como consultorías en TIC o como empresas integradoras de soluciones informáticas. Conceptos que contextualizan el desarrollo del software, primero como producto, después como servicio; es decir, pueden desarrollar software y, en lugar de vender el producto, se adapta, lo cual lo convierte en servicio.

> "(...) hablar de su propio software es muy difícil, sino tiene marca no es su propio software es algún desarrollito que hicieron para alguna empresa y ahí lo tienen. Y al cabo de seis meses es basura, si no tiene una revisión, y un control de revisiones, un control de cambios adecuado se convierte en basura en muy poco tiempo, entonces su propio software no existe eso (Líder CATI DC2.) (...) Es que nunca tuvieron su propio software, no tienen marca, ORACLE es marca, los ERP son marcas, pero si hablamos de empresas desarrolladoras como Hildenbrando o alguna otra (Softek), no tienen marca. Entonces llegan a cubrir huecos nada más. ¿Qué te falta? – preguntan- Yo te lo hago, por ejemplo un sistema de captura de quejas en algo gubernamental, lo hacen y les queda bonito y lo entregan, pero no es un producto (Líder CATI DC2.) (...) Simplemente es un desarrollo de un sola vez, que no recibe el mantenimiento, versiones, todo lo demás para convertirse en producto, le sirve a una empresa nada más" (Líder CATI DC2).

A través del tiempo el sistema informático puede conformarse como producto o servicio. En producto, si se sigue una estrategia de acumulación de procesos a través de modificaciones y mejora, del o los sistemas de software, que conforme el tiempo se estabilicen sus parámetros y se vaya disponiendo de un software reconocido en el mercado, especifico a determinado nicho. El ejemplo más disponible es el de las áreas contables o de nóminas, que aún siendo productos, éstos conllevan cierto grado de adaptación a la medida del cliente:

> "...nos especializamos, -un dato más-, en hacer un ERP que no se modificará por código para cada uno de nuestros clientes, cada vez que un cliente tenía una necesidad diferente la incorporamos como necesidad general... Hoy resolvemos prácticamente, aunque diga allí el 98% (...) prácticamente 100% de los requerimientos a todos nuestros clientes..." (Gerente DYYA RF1).

A diferencia de aquellos sistemas informáticos, que conforme transcurre el tiempo, la empresa sigue proveyendo un software que se modifica por código acorde a las necesidades de los clientes, de tal forma que conforme pasa el tiempo puede tener un mismo servicio repetido 100 veces:

"... tenemos, te podría decir 100 clientes, que a la mejor si hemos desarrollado 100 sistemas o sea 100 versiones del mismo sistema. Pero los clientes que tenemos pues se han mantenido durante muchos años con el mismo servicio, el soporte técnico que a ellos les prestas para ese sistema (líder TADI RS2)"

La empresa DYYA, propone que en lugar de prestar un servicio a la medida del cliente, lo mejor es iniciar con parámetros, que después se pueden implementar con otros clientes, en lugar de estar modificando el código, ya que de hacerlo así, de modificar el código sólo se incurre a repetir errores del sistema, depender de los programadores, caer en incertidumbre de la operabilidad del sistema, en el incumplimiento de la entrega del sistema:

"....Si te vas a la pura programación entonces tiendes a replicar el problema... entonces no lo resuelves y además puede caer en contradicción con otras líneas de código u otras partes del sistema... el otro problema... sobre todo en ERP's si metemos el factor programación durante la implantación, creas incertidumbre en la implantación, porque la implantación del sistema en su organización empieza a depender de terceros, de programadores, de... "*cuándo me entregas tal o cual* cosa de... tal o cual función" y realmente creas incertidumbre en la implantación... otro gran problema es que empiezas a tener versiones diferentes en cada cliente y cuando llega a haber cierta obsolescencia – por ejemplo que cambio la lógica de negocios, cambiaron a alta tecnología, etc.- mover esos clientes a nuevas versiones...ósea... es una tarea titánica ¿Por qué? Porque tienes que estar cociendo *(parchando)* todos los desarrollos que hiciste en cada uno de los diversos clientes...de los diversos clientes, entonces pues muchos realmente tienden a no actualizarse y hacerse obsoletos su inversión tan grande en muy poco tiempo..." (Líder DYYA JM2).

Este debate entre producto y servicio, consideramos que están yuxtapuestos los procesos, por tanto el concepto articulador de *fronteras difusas* intenta señalar que la industria del software transita entre la esfera del producto y la del servicio; pero no es un concepto rígido como los de la industria manufacturera en la cual un producto es artefacto tangible, calibrado, estandarizado; ó en la Industria de los servicios, que se considera aisladamente que un servicio es un acto inmaterial donde median símbolos y caracteres intangibles que no se objetivan en el cliente. Mas adelante, desarrollaremos este debate construyendo y proponiendo conceptos ampliados del trabajo simbólico.

2.7.2.- Ingeniería de la usabilidad y ruptura taylorista-fordista

La ingeniería de la usabilidad[33] hace uso de técnicas de la Sociología, Psicología, entre otras como la hermenéutica para aprehender las necesidades cognitivas del usuario (Londoñon, 2003; Sanz, 2004; Floria, 2000). El enfoque de la usabilidad, se centra en factores que son medidos indirectamente: facilidad de uso por parte del usuario (usabilidad), mantenimiento sencillo y que no implique muchos costos, que cumpla con las expectativas del cliente, que la interfaz, que es la pantalla con la cual va a interaccionar el usuario, sea sencilla de utilizar[34]. La ingeniería de la usabilidad propone una serie de estrategias cualitativas demasiado abstracta como para ser medidas directamente. Para poder estudiarla se descompone habitualmente en los siguientes cinco atributos básicos señalados por Nielsen (1993), Lores J. y González P. (2002) y Lores y Granollers (2004):

- Facilidad de aprendizaje: Cuán fácil es aprender la funcionalidad básica del sistema, como para ser capaz de realizar correctamente la tarea que desea realizar el usuario. Se mide normalmente por el tiempo empleado con el sistema hasta ser capaz de realizar ciertas tareas en menos de un tiempo dado (el tiempo empleado habitualmente por los usuarios expertos). Este atributo es muy importante para usuarios noveles.
- Eficiencia: El número de transacciones por unidad de tiempo que el usuario puede realizar usando el sistema. Lo que se busca es la máxima velocidad de realización de tareas del usuario. Cuanto mayor es la usabilidad de un sistema, más rápido es el usuario al utilizarlo, y el trabajo se realiza con mayor rapidez. Nótese que eficiencia del software en cuanto su velocidad de proceso no implica necesariamente eficiencia del usuario en el sentido en el que aquí se ha descrito.
- Recuerdo en el tiempo: Para usuarios intermitentes (que no utilizan el sistema regularmente) es vital ser capaces de usar el sistema sin tener que aprender cómo funciona partiendo de cero cada vez. Este atributo refleja el recuerdo acerca de cómo

---

[33] La ingeniería de usablidad fue presentada primeramente por Nielsen J.:1993, y en Europa ha sido impulsados por el grupo de investigación GRIHO de la Universidad de Leida, España, Principalmente por los investigadores Lores J. y González P.; (2002), Lores y Granollers: 2004; Véanse las excelentes tesis doctorales: Granollers i Saltiveri, Toni, (2004), MPIu+a. Una metodología que integra la Ingeniería del Software, la Interacción Persona-Ordenador y la Accesibilidad en el contexto de equipos de desarrollo multidisciplinares", Universidad de Leida, Julio de 2004. España. http://griho.net/; Sanz, Marcos (2003), "Metodología de desarrollo de interfaces de usuario gráficas para sistemas de información interactivos con alta usabilidad". Escuela Técnica Superior de Ingenieros de Telecomunicación de Madrid.

[34] En Europa se ha detectado que 28% de los programas de software a la medida no son funcionales, porque el cliente no percibe en la interfaz el conjunto de requisitos que demando (Londoñon, 2003, Sanz, 2004).

funciona el sistema que mantiene el usuario, cuando vuelve a utilizarlo tras un periodo de no utilización.

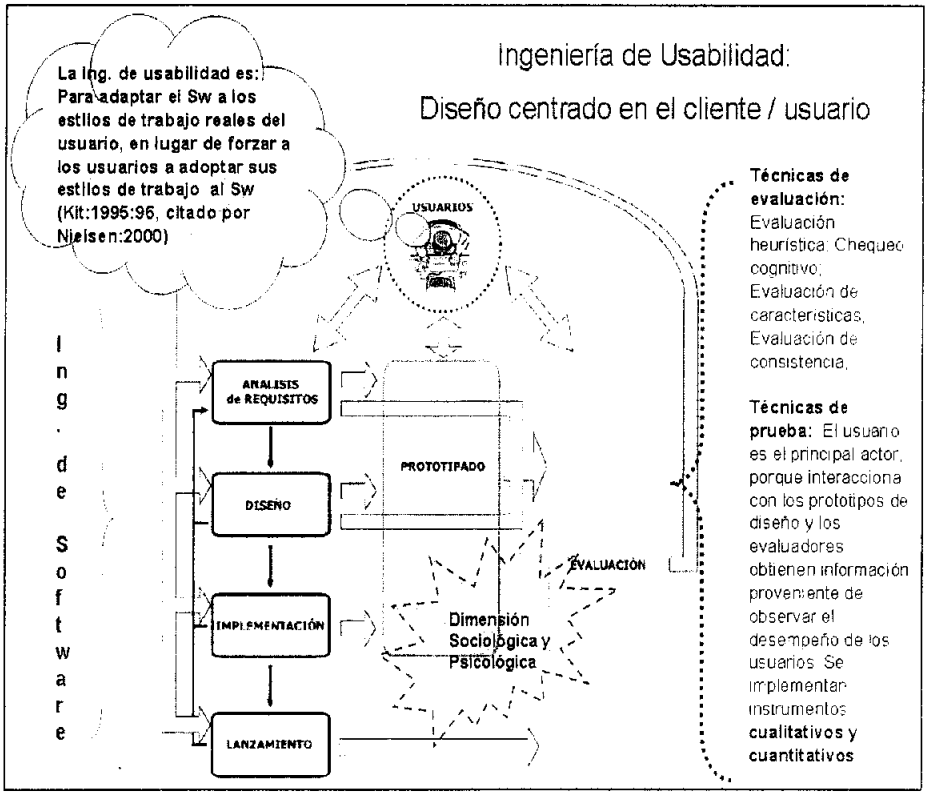
- Tasa de errores: Este atributo contribuye de forma negativa a la usabilidad de un sistema. Se refiere al número de errores cometidos por el usuario mientras realiza una determinada tarea. Un buen nivel de usabilidad implica una tasa de errores baja. Los errores reducen la eficiencia y satisfacción del usuario, y pueden verse como un fracaso en la transmisión al usuario del modo de hacer las cosas con el sistema.
- Satisfacción: Éste es el atributo más subjetivo. Muestra la impresión subjetiva que el usuario obtiene del sistema. Algunos de estos atributos no contribuyen a la usabilidad del sistema en la misma dirección, pudiendo ocurrir que el aumento de uno de ellos tenga como efecto la disminución de otro. Por ejemplo, esto puede ocurrir con la facilidad de aprendizaje y la eficiencia. Es preciso realizar el diseño del sistema cuidadosamente si se desea tanto una alta facilidad de aprendizaje como una alta eficiencia; siendo el uso de aceleradores (combinaciones de teclas que ejecutan operaciones de uso habitual) la solución más común para conjugar ambos atributos de usabilidad.

La complejidad de los computadores y de sus aplicaciones han ahondado la necesidad de desarrollar su "usabilidad" (extendida a todo el campo de la interacción hombre-máquina) y la complejidad de hacerlo, que desborda de lejos lo que se entiende por técnicas informáticas. La ingeniería de la "usabilidad" es multidisciplinar; se nutre de la informática, de la psicología, de la lingüística, de la sociología, de la antropología y del diseño industrial (esquema 4). El diseño está centrado en el cliente, se concibe al cliente como el ente que no sólo opera el sistema, sino que, integra sus metas de trabajo con las funciones implementadas en la aplicación. Para ello, debe emplear y combinar sus propias funciones cognitivas, manejando diversas capas operativas. La más elevada, consiste en encajar su modelo conceptual del trabajo, con su percepción de las funciones de la aplicación informática. En una capa intermedia, el cliente construye los comandos cognitivos correctos, para controlar en cada caso las funciones necesarias de la aplicación informática. Por último, la capa inferior comprende las acciones específicas sobre los dispositivos de entrada al sistema. Se cierra el circuito cuando el cliente interpreta el lenguaje de presentación de la aplicación -normalmente

en forma icónica sobre la pantalla-, representativa del estado en el que se encuentra el procesamiento de la aplicación (Lores y Granollers, 2004).

Esquema 4

Ingeniería de usabilidad centrada en el cliente



Elaboración propia en base a Pressman (2003) y adaptado de la Propuesta de Granolliers, T. 2004

Por último señalamos que el objeto de la Ingeniería de usabilidad es minimizar la sobrecarga cognitiva y perceptiva del cliente en la aplicación informática. La ingeniería de usabilidad utiliza un método de diseño iterativo con prototipado rápido (es imprescindible contar con técnicas de ayuda), cuya infraestructura es el ciclo "análisis-diseño-construcción-evaluación", que se repite varias veces con vistas a enriquecer progresivamente el sistema. La etapa de evaluación del prototipo se confronta con usuarios reales a cada repetición del ciclo, se revela como trascendental para obtener resultados eficientes de una ingeniería.

El proceso de trabajo del software encierra una serie de rupturas con el modelo tradicional del proceso de trabajo clásico de la manufactura: por ejemplo, no hay software libre de fallos, no hay garantía de entrega a tiempo, no se asegura cero fallas; conceptos que

fueron creados a partir de normas disciplinares y estándares en la manufactura, como la división entre concepción y ejecución, intervención de la gerencia en el proceso de trabajo, control de la administración en la ejecución del trabajo a partir de tiempos y movimientos y codificación del *saber-hacer* obrero en documentos diseñados por la empresa, entre otros conceptos torales del taylorismo y el fordismo; que para el caso del proceso de trabajo del software a la medida, encierra una serie de contingencias y singularidades que rompe con el contexto taylorista-fordista. A continuación abordaremos tres distintas polémicas teóricas que intentaremos maticen un esquema de análisis que permita abordar dicho proceso de trabajo.

2.7.3.- Herramientas procedimentales y control en el proceso de trabajo

La preocupación creciente de superar la *aflicción del software* ha llevado al departamento de defensa de Estados Unidos a analizar varios casos relacionados con los problemas de software, revelando que no sólo es necesario el entendimiento del software como producto y su desarrollo como servicio, sino que el problema más grave, no es referente a aspectos técnicos o de conocimiento, sino administrativos y aseguramiento de la calidad (Carnegie:2000). Esto llevó al departamento de la defensa de Estados Unidos a mediados de los ochenta a solicitarle al Instituto de Ingeniería de Software (SEI: Software Engineering Institute), cuya misión es proveer liderazgo en la mejora de calidad de los procesos de desarrollo de software, que interviniera al respecto. Para 1986, el SEI presentó un modelo denominado SW-CMM (Software Capability Maturity Model -Modelo de Madurez de la Capacidad de Desarrollo del Software). Este modelo es el más utilizado en la industria del software, no sólo en Estados Unidos, sino que es promovido en la industria global del software, representando un tipo ideal estándar para el desarrollo del software. El modelo CMM mide la capacidad del proceso para desarrollar un software con calidad, incrementando la certidumbre para terminar los proyectos dentro del costo y tiempo estimado, así como con el cumplimiento de los requisitos solicitados por el cliente. El modelo CMM inicialmente, nació como un cuestionario de fácil aplicación para que las organizaciones evaluaran el grado de madurez en el desarrollo de software, y fue adoptado por la industria militar y civil para evaluar a los proveedores de software de manera estándar y objetiva. La versión inicial del CMM (1.0) se difundió en 1991, fue revisada durante 1991 y 1992 para finalmente dar lugar en 1993 a la versión 1.1; su enfoque está orientado a generar e implantar las mejores prácticas de ingeniería de software como producto principal. El SW-CMM (1.1) está organizado en cinco niveles de madurez, con un total de 18

áreas claves del proceso de desarrollo de software (KPA, Key Process Areas), con 52 objetivos o metas que requieren 316 prácticas comunes. Cada uno de los cinco niveles de madurez están caracterizados por determinadas áreas del proceso de desarrollo del software. de manera que las organizaciones irán avanzando en su grado de madurez dependiendo del número de actividades que se vayan implementando y del grado de consecución de las metas definidas para cada nivel[35]. La meta fundamental, para la mayoría de las organizaciones, es alcanzar un nivel 3 de madurez en CMM. El mecanismo más adecuado para determinar el nivel de madurez actual que posee una organización es realizar una evaluación de la capacidad de los diferentes proyectos de desarrollo de software. Mediante esta evaluación, se podrá determinar si las prácticas realizadas, en cada uno de los proyectos de desarrollo, se adaptan adecuadamente a las actividades descritas en el nivel deseado. Las organizaciones que no adapten esta métrica de calidad seguirán excediendo los tiempos de entrega, ya que no hacen estimaciones realistas del avance real del proyecto, así como de la calidad en el proceso, no cuentan con bases objetivas para diagnosticar calidad y eficiencia en sus productos.

El modelo CMM es de los más difundidos en la literatura de la especialidad, la han adoptado empresas globales como Accenture, AT&T, Boeing, Ericcson, Fuji, Xerox, Hewlett Packard, Hiunday, IBM, Motorota, NASA, NEC, Samsung, Siemens, United Airlines y muchas otras. Paradójicamente para el año 2001 de 1,108 organizaciones evaluadas (sólo 370 fuera de Estados Unidos) a nivel mundial por CMM, 49 estaban certificadas en cualquiera de los cinco niveles que componen a CMM (SEI, 2001,www.sei.cmu.edu). Esta baja certificación se explica, además de los altos costos de capacitación estimados en $12,688 por persona (García, 2000, citado por Márquez, 2003), porque no es un modelo que garantice la calidad de los procesos y productos a través del tiempo, sino que es una "fotografía" que evalúa un estado de madurez en un momento dado (Márquez, 2003:215). Como respuesta a esta deficiente aceptación de CMM, el Software Engineering Institute (SEI) a mediados de los noventa y principios del siglo XXI, acordó integrara todos los modelos de calidad del software en uno sólo: Capabality Maturity Model Integratión (CMMI) (www.sei.cmu.edu/cmmi/.html) integración que culmino hacia el año del 2002. Este nuevo modelo CMMI propone una

[35] Para una mayor información véase: Paulk, M., et.l. (1993) Capability Maturity Model for Software. Software Engineering Institute, Carnegie Mellon University; véase: www.sei.cmu.edu

revisión de la calidad en el desarrollo del software de una manera *continua* y no por niveles como en el anterior (Ver cuadro 7).

Cuadro 7

Niveles del modelo de madurez de la capacidad de desarrollo del software

Nivel 5. Optimizado Orientación a la prevención de defectos. Gestión de cambios tecnológicos. Gestión de la mejora constante del proceso de desarrollo de software.
Nivel 4. Gestionado medido y analizado. Gestión cuantitativa del proceso de desarrollo del Sw. Caracterización de la capacidad del proceso de desarrollo del Sw. Gestión de la calidad del proceso de desarrollo del software.
Nivel 3. Definido (los procesos). Enfoque a la organización de los procesos. Definición del proceso de desarrollo del software y su integración en la organización. Programa de formación y capacitación de las personas. Gestión de integración de los proyectos de desarrollo del sw con la ingeniería de productos de Sw. Ingeniería de productos de Sw. Coordinación entre los diferentes grupos de trabajo. Revisiones periódicas entre iguales
Nivel 2. Repetible pero intuitivo. Gestión de los requisitos del Software. Planificación de los proyectos de desarrollo del Software. Seguimiento y control de los proyectos de desarrollo del Software. Gestión de la subcontratación de los proyectos de desarrollo del Software. Aseguramiento de la calidad de los proyectos de desarrollo del Software. Gestión de la configuración de los proyectos de desarrollo.
Nivel 1. Inicial o a medida Basado en la competencia y acciones individuales de las personas.

Fuente: Construido en base a información de Paulk. M.. et.l. (1993) Capability Maturity Model for Software. Software Engineering Institute, Carnegie Mellon University.

Este modelo CMMI, acuña el concepto de *calidad continua* el cual se basa además de los requerimientos señalados, en nuevos requerimientos enlistados en un método paralelo de evaluación y seguimiento formal del proceso: SCAMPI (*Standard CMMI Appraisal Method for Process provement* www.sci.cmu.edu. Los requerimientos del método SCAMPI contiene una serie de reglas que deberán observarse en la evaluación formal del desarrollo de proyectos de software, este modelo hace necesario e inevitable la asistencia de herramientas computacionales (CASE) en la generación de Software, ya que no se basa en una evaluación de encuesta (CMM 1.0 y 1.1.), sino que se convierte en una evaluación detallada y casi matemática del proceso en la generación del software. Este modelo implica un problema de altos costos, ya que esta diseñado para organizaciones robustas y que poseen un conjunto de desarrolladores de software con un alto grado de compenetración como equipo de trabajo y deja por fuera a los pequeños jugadores novatos y con equipos de trabajo en reciente

configuración.[36] Existen otros modelos de calidad o frameworks como los siguientes: LOGOS-CMM; SPICE (ISO TR2 15504), entre otros. La mayoría de los modelos proponen un área de Aseguramiento de Calidad de Software (SQA, por sus siglas en ingles), sin embargo, la mayoría de estos manuales[37], no han definido con claridad las metodologías que garanticen la calidad del software. Alejandro Bedini señala que existen muchos marcos de trabajo diferenciados, para los cuales no podrá existir o aplicarse un modelo de calidad específico. Para este autor, un marco de trabajo es "estructuras escritas de una idea o conjunto de metas para facilitar a una organización la aplicación de las mismas...permite que todo el personal de una organización se dirija en la misma dirección". El concepto de calidad ha sido desarrollado dentro de la Ingeniería del Software, con el objetivo de mejorar los procesos, brindar pautas para efectuar evaluaciones de las unidades de software, determinar la potencialidad y la calidad de sus procesos, así como el grado de madurez de la organización. Estos "ítems" intentan mejorar los procesos, incrementar la productividad y calidad y disminuir los costos de los proyectos de software. Estas estrategias de mejora de calidad tienen por objetivo superar la *aflicción crónica del software* (Bedini, op.cit.).

2.7.4.- Organización de los equipos de programación

El diseño del software se planifica, destacando el tiempo de finalización y el de ejecución del programa (ejecutable, es decir el interfaz con el cual hace su primer contacto el usuario). Esta planificación debe contemplar los costos que implican los factores de variabilidad y contingencias que influyen en su realización, es decir, establecer un rango de incertidumbre y riesgo; pero hacer esto incrementa el presupuesto presentado, por lo cual muchas veces el líder de proyecto no los estima, pero se incrementa el grado de incertidumbre y el riesgo de errores en la programación. Existen, varios métodos para estimar el grado de incertidumbre y riesgos, así como la cantidad posible de errores por cada mil líneas de código e incluso el rango de riesgo de un programa (Pressman, 2002). Pressman, señala que la gestión exitosa de un proyecto se basa en el contar con personal, producto, proceso y proyecto; orden que no es arbitrario ya que debe contarse con personal calificado, productos eficientes, procesos con

[36] Véase Paulk, Mark C.; et.al. (1993); Capability Maturity Model 1.1; IEEE Software, Vol. 10. No. 4, Julio 1993 pp. 18-27. CMMI-Adoption Information 2003, www.sei.cmu.edu/cmmi/adoption/adoption.html

[37] Alejandro Bedini, "Calidad tradicional y de Software", Universidad Técnica Federico Santa María Industrias, Santiago de Chile, disponible en Internet en abedini@ind.utfsm.cl, acceso 15 de Septiembre de 2007.

calidad y el resultado es un proyecto de éxito. Los elementos, el orden y el contenido son discusiones de los sesenta (Cougar y Zawacky, 1993; Whitaker, 1986; Curtis, 1988). Uno de los debates centrales es disponer con capital humano especializado, tema que es estudiado por el Software Engineree Institute (SEI) que propuso un Modelo de Madurez de la Capacidad de Gestión de Personal (MMCGP), que tiene por objetivo incrementar "la preparación de organizaciones de software para ayudar a motivar, desplegar y retener el talento necesario para mejorar su capacidad de desarrollo de software" (Curtis, 1988). Este modelo comprende las siguientes áreas: reclutamiento, selección, gestión de rendimiento, entrenamiento, retribución, desarrollo profesional, diseño de la organización y del trabajo, desarrollo cultural y espíritu de equipo. Antes de planificar un proyecto debe establecerse los objetivos y el ámbito del producto (se entiende por producto cualquier software que será construido a petición de los clientes), así como considerar soluciones alternativas e identificar las dificultades técnicas y de gestión. Sin esta información no se puede estimar los costos, hacer una valoración efectiva del riesgo, subdividir tareas o una planificación del proyecto que proporcione una indicación fiable del progreso del proyecto. Para establecer los requisitos del software, el cliente y el desarrollador deben reunirse para definir los objetivos del producto y su ámbito, así como las metas generales del proyecto sin considerar el como se lograran. El paradigma o modelo que se practique proporciona la estructura desde la cual se puede establecer una detallada metodología para el desarrollo del software. Diferentes conjuntos de tareas (hitos, productos de trabajo, puntos de garantía de calidad, etc.) permiten a las actividades estructurales adaptarse a las características del proyecto del software y a los requisitos del equipo. Dirigir los proyectos de software a través de una planificación, es la única manera de gestionar la complejidad de los mismos y, tratar de evitar el colapso del proyecto haciendo todo lo posible por compartir entre los gestores del proyecto y los desarrolladores un conjunto de señales de peligros comunes, comprender los factores de éxito críticos que conducen a la gestión correcta y desarrollar entre ambos un enfoque común para planificar, supervisar y controlar el proyecto; pues la sombra del fracaso esta presente: en 1998, los datos de la industria del software indicaron que 26% de los proyectos de software fallaron completamente y, 46% experimentaron un desbordamiento en la planificación y en el costo (Reel, 1999). Todo equipo de programación debe tener una organización interna; pueden identificarse tres estructuras básicas: grupos democráticos; grupos con jefes; grupos bajo jerarquía administrativa.

Capitulo III

Cuestiones teóricas en torno del control de proceso de trabajo:

tiempos y pensamientos en un trabajo cognitivo

Un gerente fue con el maestro Desarrollador y le mostró el documento
de requerimientos para una nueva aplicación. El gerente le preguntó al maestro:
"¿Cuánto tiempo tomará diseñar este sistema si le asigno cinco Desarrolladores al proyecto?"
"Tomará un año", dijo el maestro rápidamente. "¡Pero necesito este sistema inmediatamente o antes!
¿Cuánto tiempo se tardará en diseñar este sistema si le asigno diez Desarrolladores?"
El maestro frunció el ceño. "En este caso, tomará dos años."
"¿Y si le asigno cien Desarrolladores?" El maestro se encogió de hombros
"Entonces el diseño nunca será completado", dijo.
Tao de la programación

3.1.- Introducción

En el presente apartado abordaremos algunas discusiones teóricas en torno a la separación entre concepción y ejecución, entre estandarización del trabajo al estilo de Taylor y Ford y el trabajo flexible y polivalente. Es importante señalar que las distintas corrientes teóricas que han polemizado las circunstancias contextuales de la organización científica del trabajo (OCT), el consenso, la negociación, la resistencia, son fundamentales para comprender el proceso de trabajo de la Ingeniería del Software. Primero abordamos una discusión muy breve en torno a las diferentes escuelas que abordan en la sociología del trabajo, la racionalización de los tiempos y movimientos en la organización científica de Taylor y la división del trabajo en Ford, para continuar con las propuestas de la Sociología Industrial, las condiciones y cualificaciones del trabajo, que analizan el conflicto, la resistencia y el consenso que se conforman a nivel del piso de la producción. Sin embargo, es importante señalar que la lógica interna de dichas teorías tienen como concepto central el control del proceso de trabajo, el cual para el caso del trabajo en programas de software a la medida no es evidente de quien lo ejerce y como se ejerce. El trabajo en el software es creativo, se caracteriza por elaborar *símbolos a través de signos*, mezclando textos algorítmicos que proceden de lenguajes de programación y de un complejo subjetivo que está embebido por contextos culturales, emotivos, estéticos así como de intencionalidades, de sentido del compromiso, además de un conjunto de acumulaciones tácitas de experiencia, habilidades y destrezas personales, así como por interacciones objetivas y subjetivas individuales y por equipos, formales e informales; actividades de resistencia, boicot ó consensos y arreglos sociales o individuales.

3.2.- Racionalización del trabajo o nuevo estadio de producción capitalista

La transformación en la organización y el proceso de trabajo tiene como referente histórico obligatorio los mecanismos de control y apropiación de los conocimientos de los artesanos del siglo XVIII. Este proceso tuvo un punto culminante con la puesta en marcha de la organización científica del trabajo (OCT) de Friederich W. Taylor (1856-1915). La OCT tiene como antecedentes, el hecho que a mediados del siglo XIX:

> "era difícil encontrar mano de obra hábil y disciplinada, y sin los obreros de oficio ninguna manufactura podía instalarse o funcionar; más aún, debían desplazarse hacia donde residían los mismos. La mano de obra industrial calificada era el factor de producción más caro y sus conocimientos se transmitían a sus descendientes, pues ellos tenían conciencia de que si su número era reducido, dominarían a los patrones y los obligarían a pagar altos salarios" (Ure,1845; citado por Neffa, 1994:101).

Los obreros calificados empezaban a organizarse en sindicatos, éstos presionaban a través de las huelgas y del *label*[38]. Los trabajadores de oficio gozaban de autoridad en el proceso de producción, de un dominio no codificado en el *saber-hacer* del trabajador; este *saber-hacer* era fruto de un largo proceso de acumulación del conocimiento empírico generado en la cotidianeidad que se transmitía mediante las relaciones cara a cara en el proceso de trabajo, la circulación de la información se facilitaba en el proceso denominado *aprendices de oficio*, el aprendizaje era encabezado por el artesano mas diestro, que transmitía sus conocimientos a través de acciones informales no sistematizadas en las prácticas cotidianas que se embeben en la *interacción social*. se configuran proceso de aprendizaje situado en los cuales confluyen conocimientos tácitos como el *saber-haciendo* y *saber-aprendiendo*. Si bien es cierto los capataces y mandos medios eran también trabajadores de oficio con experiencia y conocían el proceso, aún para ellos mismo el "status qou" del dominio del proceso de producción por parte del sistemas de obreros calificados era normal. Este *monopolio* del conocimiento técnico del *saber hacer* constituía la base de resistencia y poderío de los trabajadores de oficio en relación con la gerencia. Sabían que en función de este conocimiento no codificado se regulaban las remuneraciones, permanencia del trabajo y control sobre el proceso de

[38] Label: era una especie de "certificado de calidad" que los sindicatos otorgaban a los productos y a su vez, este certificado garantizaba al consumidor final que el patrón "si cumplía" con la clase trabajadora. (era una especie de presión social de los sindicatos sobre las empresas). Neffa, 1994:101

producción. Otro elemento importante a mediados del siglo XIX, lo constituye el agotamiento del régimen de acumulación extensivo que tenia por base las extensión de la jornada de trabajo, contratación de mayor numero de trabajadores, incorporación de más maquinaría universal, etc.; éstas acciones del sistema capitalista correspondían al régimen de acumulación basado en el uso intensivo de la fuerza de trabajo (mayor productividad y eficiencia con mayor desgaste de la fuerza de trabajo), que no obstante significaba un determinado progreso técnico (tecnología dura) simbolizada en la maquinaria multipropósito, un proceso de trabajo estándar y simplificado (tecnología blanda) y, fragmentación del producto en partes y componentes estandarizados (atomización del producto).

Frederich W. Taylor a través de su propuesta de la organización científica del trabajo emprendió la expropiación del *saber-hacer* del obrero a través de una serie de estrategias administrativas, como sistematizar y estandarizar el proceso de trabajo en tiempos y movimientos vigilados por el capataz y explicados en un manual de procedimientos avalados por la gerencia; en otras palabras codifico el *saber hacer* en manuales operativos que mostraban el "único mejor camino" para desempeñar una actividad laboral. Para Taylor uno de los elementos fundamentales en su tesis de la organización científica del trabajo era reducir los tiempos muertos en el proceso de producción. La reducción de éstos tiempos significa un incremento en la producción; manteniendo otras variables constantes, como maquinaria, numero de trabajadores, jornada de trabajo, entre otras; lo cual se traduce como el uso intensivo de la fuerza de trabajo, acción central en este régimen de acumulación del capital. Taylor, señalaba que los trabajadores tienen diferentes formas de hacer un trabajo, el cual no siempre es el más eficiente, para evitarlo la administración científica debe proveer una única mejor manera de hacerlo ("one best way"), esto es que la gerencia homologa los tiempos y movimientos del proceso de producción y, el cumplimiento de dicho "camino" es vigilado por la administración las herramientas del cronometro, la supervisión y los manuales de procedimiento. Taylor señala que la división social del trabajo no implica separación de trabajos intelectuales y trabajos manuales, ya que ambos pueden coexistir en el proceso de trabajo; sino que la división técnica del trabajo, responde a la necesidad -ya señalada por A. Smith- de una especialización que represente una mayor eficiencia

productiva, esto es gracias al desarrollo de habilidades en cada trabajador individualmente y consiste en la segmentación de la producción en serie de operaciones que pueden realizarse aisladamente. Si bien es cierto, estas no son categorías inventadas por Taylor, si le corresponde el merito de orientar el sistema capitalista hacia un nuevo estadio de mayor intensidad y eficiencia en el proceso de trabajo, lo que se traduce en mayor producción por obrero. La base de haber logrado esta mayor producción por obrero reside en la idea que "el contenido básico de trabajo del producto o del proceso es el tiempo mínimo que se necesite teóricamente para obtener una unidad de producción" (Neffa, 1994:192-198).

Las empresas de Ford ya estaban taylorizadas; Ford, sin conocer a Taylor implementó técnicas descritas en el libro de la organización científica del trabajo en 1911. Ford fue precursor de la producción en masa, estos es productos estandarizados y fabricados bajo un esquema especializado. En un primer momento los trabajadores de Ford eran obreros-artesanos que poseían un conjunto de habilidades y destrezas industriales provenientes de sectores como el agrícola, de bicicletas, de armas de fuego, etc.; es decir, poseían un conjunto de conocimientos de los ritmos de la industria y las exigencias de una fábrica. En las empresas de Ford este elemento fue importante para integrarse a los ritmos y exigencias de la cadena de montaje implementadas en el fordismo. La producción en cadena implicaba la intercambiabilidad de partes y componentes dentro de la empresa, para lo cual las maquinas herramientas de propósito universal o múltiples fueron sustituidas por maquinas herramientas especializadas, que produjeran a su vez piezas estandarizadas. El merito de Ford radica en que se organizó un sistema de producción en masa de consumo generalizado. Además de sistematizar el proceso de producción más allá de "pensar en la flojera de los trabajadores," como lo explicaba Taylor, partió de un principio básico: producir a un menor precio un mejor producto. La fábrica de Ford era un sistema coherente, donde el trabajo humano había sido pensado como una prolongación o un apoyo del sistema mecánico (Naville, 1985).

Hacia finales del siglo XVIII tres condiciones particulares se sucedieron para revelar la ventaja de la división del trabajo: un incremento en la habilidad individual de los obreros; economía del tiempo perdido en el paso de una tarea a otra y el desarrollo técnico traducido en máquinas especializadas. Estas condiciones permitieron una

división en "parcelas" discontinuas del proceso de trabajo. Naville (1985) propone que esta división del trabajo ocurrió de diferentes formas y procesos, no sólo en el taller, sino también en la organización administrativa, en la gestión productiva, etc., y, acorde con la división del trabajo, le correspondía una cantidad de trabajo; es decir a una porción determinada de trabajo mecánico (maquinaria-estaciones) corresponde una porción determinada de trabajo humano (Naville, 1985:375). Esta relación se amplía con los progresos tecnológicos que se traducen en una mayor automatización, donde el obrero va perdiendo participación, reduciéndose a sólo "controlar los comandos", por tanto estas parcelas se van volviendo funciones derivadas de la evolución técnica de la maquina.

Harry Braverman, en su obra "Trabajo y Capital monopolista" (1974) critico la división de tareas al estilo de ford y la separación entre concepción y ejecución y el control de la gerencia sobre el trabajo (taylorismo). Braverman estableció que la fuerza de trabajo (apoyándose en conceptos marxistas) es la capacidad humana para realizar una actividad transformadora, y es en esta en la cual descansa la economía capitalista. La economía capitalista requiere de la compra y venta de la fuerza de trabajo y para obtenerla se han generado tres condiciones básicas: i) Los trabajadores son separados de los medios de producción; ii) Los trabajadores son libres (se termino la servidumbre y el esclavismo); iii) Los trabajadores son contratados en el mercado de trabajo por una determinada cantidad monetaria, lo cual los convierte en una unidad de capital (capital variable). Por tanto lo que se traslada al capitalista es la capacidad creadora a través de su fuerza de trabajo, sin embargo esta capacidad creadora no es una cantidad que se pueda medir o convenir antes del momento en que el trabajador esta laborando en determinada jornada de trabajo, debido a que la fuerza de trabajo es potencial, es decir produce mas de lo que consume para su reproducción. La capacidad creativa de la fuerza de trabajo se debe en gran parte a que es moldeable, funcional, orientada a metas, polivalente, inteligente, creativa, etc. Por tal motivo el tiempo de trabajo no retribuido crece exponencialmente frente al trabajo pagado.

Braverman se basó en Taylor para demostrar que el "único mejor camino" resultó en una separación entre concepción y ejecución del proceso de trabajo, que antes estaba implícito en el trabajo, los trabajadores concentraban el *saber hacer*, la OCT dividió el

conocimiento de la ejecución. Es decir, la administración transfirió a la gerencia el *saber-hacer* del proceso de trabajo, en todas y cada una de sus fases. La OCT señala 3 principios: i) Disociación del proceso de trabajo de la pericia de los obreros; ii) Separación de la concepción y ejecución; iii) Uso del monopolio del conocimiento para controlar cada paso del proceso de trabajo y su modo de ejecución. Estos principios facilitaron un incremento en la productividad, un proceso de trabajo parcelado, segmentado en espacios separados; por un lado el diseño, planeación, concepción y por el otro la ejecución; ello implico una división y polarización del proceso de trabajo, desaparición de oficios que, a su vez implicó que se rompieran los lazos entre la población obrera y se destruyó el conocimiento implícito del oficio.

3.3.- Mecanismos de control en el *saber-hacer* del trabajo

En la década de los sesenta y hacia mediados de los setenta cobró notoriedad la discusión teórica en torno al estudio de la relación capital-trabajo desde la perspectiva estructuralista. Las luchas del movimiento obrero fueron el centro de reflexión, la explicación de sus períodos, causas de movilización, participación colectiva, burocratización sindical, etc.: constituyeron un amplio espectro de temas importantes. Una vertiente siguió el camino de Braverman y discusión teórica se centró en exponer los mecanismos del control y resistencia del proceso de trabajo, en especial los límites del talylorismo-fordismo hacia fines de los setenta. Se trataba de dar cuenta del *saber-hacer* del trabajador, de la influencia en el trabajo de las habilidades y destrezas desarrolladas, de la importancia o no de la experiencia acumulada en las prácticas laborales cotidianas; de las relaciones de poder, consensos o resistencias individuales o colectivas en el proceso de trabajo.

Es la sociología industrial, que se caracterizaba en esos años por centrarse en la naturaleza de las tareas específicas y la organización del trabajo en la industria manufacturera que se interesaba principalmente por la experiencia laboral, se concentraba en las ocupaciones manuales y consideraba a la empresa como un sistema cerrado; Gallie (1989) señalaba que los investigadores más influyentes en los sesentas como Joan Woodward en Inglaterra (1965, 1970) y Robert Blauner en Estados Unidos, coincidieron en decir que la tecnología productiva era "*el principal determinante de la naturaleza de las tareas específicas, de la estructura organizativa, de la experiencia de los trabajadores en materia de empleo, y en consecuencia,*

las relaciones sociales entre la dirección de la empresa y los trabajadores" (Gallie, 1989:110). Estas observaciones llevaron a considerar las relaciones estructurales en la empresa y una correlación entre tipos de tecnología y sistemas de control; lo que se traducía en una especie de variable independiente tecnológica que funcionaba como vínculo entre conducta organizativa en las empresas y relaciones laborales en el trabajo (Woodward, 1970). Woodward proponía que *"cuanto más avanzada fuera la tecnología, más fácil sería para la dirección de la empresa diseñar modelos de organización del trabajo que reflejarán más las necesidades sociales que las limitaciones técnicas"* y suponían que se podría conseguir así una mejor calidad de vida, autonomía en el trabajo y armonía social.

Esta postura optimista-estructuralista se enfrentó hacia mediados de los setenta a la tesis de la descualificación, la existencia de una "creciente degradación del trabajo" (Braverman, 1974). Braverman criticó que la sociología industrial inglesa y norteamericana carecieran de una perspectiva sólida para analizar el trabajo. Sin embargo, las criticas a Braverman de esa época consideran que no comprendía que los empresarios puedan desplegar estrategias muy diversas con el fin de controlar la ejecución del trabajo y que el empresario podía optar por minimizar el margen de resistencia del trabajador mediante una estrecha supervisión del trabajo y reducir al mínimo la responsabilidad del trabajador o bien podía optar por darles autoridad, responsabilidad, estatus y dirección, esto mediante la conformación de equipos de trabajo con cierto grado de poder y decisión.

Es la sociología francesa la que recupera la perspectiva de comprender el significado del trabajo para el trabajador dentro y fuera del proceso de trabajo. Para Touraine la sociología del trabajo parte *"del trabajo y no del comportamiento del hombre en el trabajo, de las relaciones reales de los diversos aspectos del trabajo y de los diversos niveles de valorización y no de su impacto sobre el trabajador, de su unificación en el comportamiento del trabajador"* (Castillo, 2003:40). La sociología del trabajo, a diferencia de la sociólogа industrial se caracterizaría por identificar las interrelaciones maquina-entorno-trabajador; condiciones de trabajo, acciones de los individuos que conforman un equipo de trabajo; reconoce las condiciones sociales y técnicas bajo las cuales los trabajadores toman decisiones, adaptan sus actividades reales a tareas especificas; con la consecuente necesidad de utilizar no sólo nuevos conceptos, sino también nuevos instrumentos de recopilación y tratamiento de la información. Esta sociología del trabajo intenta estudiar las estrategias que emplean los

trabajadores, el como adaptan y modifican sus actividades reales a las tareas que le son prescritas; se concentra en las formas de adaptación, resistencia, consenso, normas disciplinarias y de control del trabajo. Sin embargo, al igual que la sociología industrial, la sociología del trabajo se sumó a la creencia que el progreso técnico llevaría al progreso social. Sin lograr desprenderse de prejuicios estructuralistas la sociología laboral señaló que el conjunto de calificaciones que posee el trabajador es resultado del sistema de certificaciones que construyen los empleadores y reconocida por sindicatos *"para establecer una equivalencia entre las operaciones técnicas realizadas por el trabajador y su valor y su reconocimiento social"* (Abramo y Montero, 2003:72).

Esta herencia estructuralista se impregna también de los postulados de la sociología de la cualificación, cuando esta señala que un conjunto de *X* cantidad de cualidades que integra un puesto de trabajo, irremediablemente esta haciendo una codificación de determinadas características que contiene un puesto de trabajo; señalamiento que resulta en una lista de atributos que permite jerarquizar a los individuos y conjunto de tareas que se necesitan para cubrir determinad actividad laboral. Un problema central es como se construye la definición de estas cualidades. Tripier (1995:153) rechaza la visión "universal y natural" de las cualificaciones, y plantea que son las capacidades aprehendidas del individuo a través de diferentes trayectorias de aprendizaje lo que constituye un conjunto de conocimientos. Tripier insiste que, aunque en la actualidad se considere que el desarrollo tecnológico se debe al orden cuasi-natural del progreso técnico, no necesariamente se debe a los avances en la ciencia física, sino que influye la magnitud o avance del proceso social (socialización del conocimiento) del *saber hacer;* por ejemplo, los carpinteros de barcos, de molinos, de viviendas influyeron en la construcción de nuevas fabricas al socializar los rudimentarios conocimientos de gestión a los nacientes empresarios en la revolución industrial (Tripier, 1995:154). Es en la segunda revolución industrial donde la doctrina liberal admite procedimientos de clasificación "societales"[39], sin embargo paradójicamente aún considera esta doctrina que el mercado es regulado de acuerdo con las doctrinas económicas.

[39] Para una mayor ejemplificación de cómo el conocimiento societal de los individuos influyo en el desarrollo industrial, véase el trabajo clásico de Thompson 1989, Jugando a ser trabajadores cualificados. Cultura de fábrica y enorgullecimiento por la cualificación laboral entre los obreros del automóvil de Coventry, en Sociología del Trabajo, nueva época, num.7, pp, 105-140; Lope Andreu y Antonio M. Artiles, (1993), Cambio técnico recualificación, nueva época, num. 19, pp. 69-07.

Las cualificaciones que posee el trabajador son resultado de un proceso social (socialización del conocimiento) y no necesariamente de una categorización construida por el empresario a partir del conjunto de necesidades y destrezas que exigen la maquina-herramienta o por el Estado (sistema de profesionalización). Esta dicotomía se centra en el hecho que para el empresario aceptar una categorización extrínseca no solo es contradictorio a la soberanía de la empresa, sino que implica que el empresario tendría que negociar el reconocimiento de la calidad de la fuerza de trabajo y su costo. Sin embargo la contradicción se atenúa cuando el empresario posee una maquina-herramienta sofisticada e importante en el proceso productivo y acude al mercado externo reconociendo las certificaciones de cualidades que expide el Estado a través, por ejemplo del sistema educativo o sistema de certificación. Lo anterior implica dos procesos, por un lado, la empresa establece controles de todo tipo para habilitar, especializar y cualificar a trabajadores y ofrecer a éstos un sistema de categorización salarial. Por el otro, la empresa se libera de la necesidad de capacitar y lo transmite al Estado. Sin embargo, esta premisa no es del todo segura tratándose de nuevas profesiones en las cuales el conocimiento aún no se socializado y la curva de "aprendizaje social" aún esta en proceso de construcción, por ejemplo, ¿Existe una socialización del conocimiento de las nuevas profesiones como son los ingenieros en dómotica, mecatrónica, software, entre otras?. Para Tripier esta paradoja se ha resuelto a través de tecnificar el proceso de producción sustituyendo fuerza de trabajo. Sin embargo, Kern y Shumann (1988) [40] señalan que los cambios en el sistema productivo como producto de innovaciones tecnológicas, implican una disminución en la división del trabajo, pero paralelamente se promueve una mayor inteligencia productiva de los operarios y la re-profesionalización continua del trabajo (innovación continua). Los autores establecen límites a esta propuesta, sin embargo plantean que se esta en presencia de un momento en el cual ni el mercado, ni el producto permiten una racionalización de la producción al estilo taylorista-fordista, ya que la exigencia de competitividad basada en nuevos productos y ante una demanda que exige diversidad, calidad y disponibilidad inmediata, ha impuesto una profunda reorganización del proceso productivo (Kern y Schumann, 1989:367). Para estos autores "*la propia utilización del capital exige el cambio en la utilización de la mano de obra. Cuanto más acercan las concepciones del producto a la*

[40] Kern, H. y Schumann, 1989, El fin de la división del trabajo: Racionalización en la producción industrial. Madrid, M.T.S.S y Kern, H. y Schumann, 1988, El fin de la división del trabajo; Revista e sociología del Trabajo, num. 2, 1988.

generación de artículos de calidad de alta complejidad...tanto más necesaria es la introducción de nuevos conceptos de producción basados en la idea de recomponer las tareas de manera totalizadora, lo que exige una utilización más amplia de las cualificaciones" (Kern y Schumann, 1989:368).

3.4.- Consensos, juegos y resistencias en el *saber-hacer*

Michael Burawoy (1979) planteó su tesis central de que los trabajadores poseen un determinado poder en el proceso de trabajo, este se da en forma de juego entre los mandos medios y los trabajadores en el piso de la fábrica. Burawoy, así construye una teoría del consentimiento que se genera en el lugar de producción para lo cual se ayuda de categorías marxistas. Burawoy señala que en el capitalismo los trabajadores no pueden transformar por sí solos la naturaleza, ni ganarse la vida en forma independiente. Están privados del acceso a los medios de producción. Éstos tienen que vender su fuerza de trabajo a cambio de un salario, el cual se convierte en su medio de subsistencia. Burawoy se cuestiona ¿Porque los trabajadores regresan al día siguiente?, ¿Porque trabajan? (a diferencia de la sociología industrial que se pregunta por qué no trabajan suficiente). Tres procesos explican la presencia del trabajador: los juegos en el proceso de trabajo, el mercado interno de trabajo y el Estado interno. Con los juegos se refiere al establecimiento de reglas informales para el cumplimiento de metas de producción y su cumplimiento entre trabajadores y algunos mandos de nivel inferior. Estos juegos representan un vínculo entre la racionalidad individual y la del sistema capitalista, porque se busca cumplir las tareas asignadas de producción, así se genera un consenso que lleva a los trabajadores a aceptar y defender las normas y reglas sociales existentes en el proceso de trabajo con la posibilidad de satisfacción subjetiva en el cumplimiento de intereses (que aparecen como propios) lo que genera consenso y se presentan como naturales e inevitables. Al respecto Marx planteaba que los trabajadores están completamente a merced del capitalista, por tanto este puede incrementar la productividad o intensidad. Sin embargo Marx no consideró la organización del consentimiento o la necesidad de lograr un espíritu de colaboración en el proceso de transformación de la fuerza de trabajo en Trabajo. Esto es así porque en el siglo XIX el margen del consentimiento era escaso. Con el transcurso del tiempo y como consecuencia de la lucha de clases el salario se ha desligado del esfuerzo individual,

por tanto es importante complementar las medidas coercitivas (ya señaladas por Marx) con la organización del consentimiento.

Para Burawoy el proceso de trabajo es entendido como mercado interno, es una conjunción concreta de coacción y consentimiento que induce al trabajador a colaborar en la búsqueda del beneficio. Ahora bien, ¿Como se genera este consentimiento?. Burawoy (1979: 202 y ss) cita a varios autores que señalan que los trabajadores aún en el más puro estilo tayloriano "conservan la posibilidad de obtener de su trabajo una satisfacción residual..." (De Mann, 1927, citado por Burawoy, 1979). La intensidad del trabajo no siempre se traduce en sensaciones de desagrado, sino que puede producir ciertas satisfacciones...las condiciones de trabajo dan lugar a privaciones pero éstas pueden generar satisfacciones relativas... (Baldamus, 1961 citado por Burawoy, 1979) con el fin de profundizar en estas satisfacciones relativas en el proceso de trabajo, estas se presentan en formas de juegos lo cual reduce la tensión. Burawoy, debate la creencia que los trabajadores establecen de forma autónoma sus propios sistemas culturales y de producción en oposición a la empresa, señala que ello no es correcta del todo, por el contrario coexisten elementos en forma de juegos que aparecen en el marco de un proceso de lucha y negociación, pero se desarrollan dentro de limites definidos por las remuneraciones y márgenes aceptables de beneficios y consensos.

La dirección administrativa forma parte de éste conjunto de juegos y consensos, conformando una especie de mercado interno, donde a través de capataces –entendidos como agentes- y gerentes que participan en la organización de los juegos y cumplimiento de las reglas no escritas, ayudando activamente a que los trabajadores cumplan con dicho Estado interno, entendiéndose éste como la puesta en marcha de prácticas informales y formales para cumplir con la meta de producción, determinada por el "grupo" de trabajadores, compartiendo el disgusto por las "metas" etc. La satisfacción y deseo de participar en este juego nace tanto de la necesidad del trabajo y de las exigencias del proceso de trabajo en el piso de la fábrica, como de la aparición de "necesidades radicales" o de "una nueva concepción del trabajo". La satisfacción de esa necesidad reproduce la "sumisión voluntaria" (el consentimiento) y también una mayor riqueza material (beneficio-plusvalía). En el mercado interno, el concepto de "arreglárselas" busca describir como se integra el trabajador en el proceso de trabajo como sujeto y no simplemente como miembro de una clase definida por una determinada relación

con los medios de producción. El consenso es generado por el interés de los trabajadores en preservar el mercado interno de trabajo como algo que les beneficia.

Lo que Burawoy (1979) llama estado interno se refiere a la negociación o consenso de intereses sindicato-empresa, trabajadores-gerencia a través de procedimientos formales e informales. El estado interno reorganiza los intereses de trabajadores y gerencia. Las relaciones de producción capitalistas quedan encubiertas porque los trabajadores participan como individuos y no como miembros de una clase. El conflicto es regulado e implantado en el procedimiento de negociación. El estado interno reconoce la existencia de clases, pero la incertidumbre que deja sobre lo que puede ser negociado se resuelve de forma que "*establece la prerrogativa gerencial para dirigir el proceso de trabajo*" (p.120). Las reglas (formales e informales) de estos procesos no generan estabilidad o posibilidad de predicción, sino que ubican la incertidumbre dentro de ciertos límites. Se asientan reglas y se amplían posibilidades de elegir, alcanzando los trabajadores cierto grado de *saber como poder* para resistir o protegerse de la dominación de la gerencia. La empresa "obscurece y asegura el plusvalor a través de la organización, relocalización y represión de la lucha, a través de la constitución o presentación de los intereses de la corporación como los intereses de todos/as, y a través de la promoción del individualismo..." (p.120). A la combinación de conflicto y consenso, propuesta por Burawoy, Edwards (1994) añade los conceptos de antagonismo estructurado y negociación, como explicativos de las relaciones dentro del proceso de trabajo. El primero está instituido en el proceso de trabajo "porque los trabajadores están en posición subordinada: su capacidad de trabajo -su fuerza de trabajo- se transforma en trabajo efectivo bajo la autoridad de la gerencia, y el excedente creado no está bajo su control" (Edwards, 1994:6-7). La negociación se explica a partir de una serie de preguntas: ¿Cómo se organizan, expresan y regulan los aspectos conflictivos del trabajo?; ¿Cómo se crea un orden en el lugar de trabajo?. Para contestarlas debe partirse del debate en torno a la existencia o no de la supuesta separación entre control gerencial y resistencia de los trabajadores que hacen Braverman y Friedman y, en una línea cercana a Burawoy debe considerarse que coexiste una negociación no formal dentro del proceso de trabajo "...tanto como conflicto hay cooperación, porque los gerentes necesitan la complacencia de los trabajadores, mientras que los trabajadores dependen de los empleadores para su trabajo y no pueden afrontar el adoptar una política de resistencia total. El balance entre los aspectos conflictivos y de cooperación en el

trabajo se da a través de la negociación del orden" (Edwards y otros, 1994:7). La negociación es necesaria porque la gerencia opera bajo la presión de contradicciones entre la necesidad de control y la de conseguir la cooperación de los trabajadores por un lado, y la necesidad de mantener la armonía en la empresa y obtener su rentabilidad por el otro. La negociación no es accesoria, sino que es parte integrante de las relaciones entre gerentes y trabajadores, el recurso para regular el trabajo en medio de las demandas contradictorias (Edwards y otros, 1994:8). En cada espacio laboral se resuelve el problema a través de negociaciones formales e informales, cotidianas y ubicadas en el lugar de trabajo, aunque influidas por las tendencias globales y los sistemas nacionales de relaciones laborales (Edwards y otros, 1994:278-79).

3.5.- El trabajador flexible y la psicología laboral

Touraine (1985) señala que el desarrollo en serie de la producción en masa del fordismo implicó la sustitución de obreros calificados por obreros especializados a los cuales se asignan tareas limitadas, repetitivas y rápidamente aprendidas. Cuanto más compleja es la maquinaria y más se mecanizan o automatizan las operaciones mas importancia adquieren los obreros técnicos y los obreros calificados presentan un ritmo decreciente (1985:385). En una especie de tres fases profesionales históricas del obrero. Para Touraine una primera fase se caracterizaría por el predominio de la acción autónoma del obrero calificado. En esta fase, mientras las condiciones económicas, técnicas y organizativas no permitían conocer el proceso completo de producción al detalle por la gerencia, la organización del taller se reducía a la distribución del trabajo entre obreros o equipos capaces de adaptarse a tareas variadas, competentes como para organizar por si mismos su trabajo. En ese sentido, aunque el empresario se encuentra en posición de coaccionar económicamente, el obrero posee una autonomía que se traduce, al menos para los más calificados, en una libertada en relación con la empresa.

La calificación se define como un nivel de conocimientos más que como una facultad de decisión y en muchos casos como capacidad de mando, un principio de organización del trabajo. También existe una carrera obrera, cuyos grados son con frecuencia subrayados por símbolos y ritos. A este sistema se le puede calificar de profesional-especializado. Una segunda fase está representada por el predominio de la organización centralizada del trabajo que va junto al sostenimiento del trabajo de ejecución directo. Es decir, a medida que

desaparecen las condiciones técnicas y económicas de la producción y se hace previsible establecer un plan de producción del cual todos los elementos son supuestamente conocidos y las técnicas y métodos de trabajo sean descritos por la gerencia al estilo de la organización científica del trabajo.

La tercera fase es cuando la empresa se compone en lo sucesivo de puestos de trabajo ocupados por obreros técnicos, donde el aparato técnico de producción es independiente de los obreros. Al interior de este se sucede una transformación radical: la industria moderna de producción a gran escala implica el reemplazo de maquinaria universal o polivalente por máquinas especializadas, esta especialización remite a un tipo de operación bien definida que a su vez permite reagrupar las operaciones elementales (Touraine, 1985:388 y ss). Los obreros especializados están sometidos a una organización centralizada del trabajo; no representan ya un potencial susceptible de múltiples usos, se definen por sus puestos de trabajo y son intercambiables. No se les considera su capacidad de ejecutar una operación manual, sino la aptitud para adaptarse a las condiciones de la producción mecanizada y en gran escala, es esto lo que define al obrero especializado de la industria moderna: su polivalencia y flexibilidad. En este sentido el análisis de los puestos de trabajo no parten de un estudio de las operaciones individuales sino de la definición de los puestos como etapas para adaptar el obrero a la producción y a los instrumentos de trabajo. Los obreros especializados polivalentes son flexibles, son capaces de sustituir en la cadena al obrero enfermo ó ausente, ayudar al nuevo obrero a adquirir los conocimientos con la rapidez necesaria que exige la cadena de producción. Esta visión determinista y casuística de Touraine, es desmitificada por una segunda generación de tayloristas, que consideran que el "único mejor camino" de Taylor es sustituido por enfoques de identidad con la empresa, lealtad y entrega con los compromisos productivos; se sustituye "el único mejor camino" por "la única mejor persona" que es aquella que tiene la "camiseta puesta", que "da todo por la empresa"; para quien la empresa es "parte de su familia" (Watts, 1991). Las propuestas de la psicología industrial redefinen las relaciones entre los actores de la producción que había hecho Taylor, así como sus roles.

> "Como revolución gerencial promovió la reorganización del trabajo alrededor de la disciplina, la obediencia y el culto a la autoridad. Estableció la legitimidad social, en el taller y en la sociedad, de la profesión de los gerentes-ingenieros, que evolucionaron hacia una completa burocracia tecnocrática..." (Watts, 1991:132).

Hacia mediados de la década de los setenta cobra fuerza un movimiento que va mas allá de las filosofías empresariales, es el movimiento que estudia la satisfacción en el trabajo y la calidad de vida (corriente teórica conformada por investigadores del Instituto Tavistock) quienes critican las anteriores concepciones sobre de los factores psicológicos, y señalan la necesidad de crear estructuras apropiadas para que sea posible un clima social deseable, y relaciones interpersonales positivas (Trist, 1993:51). Sobre la psicología en el trabajo afirman que es necesario atender otro tipo de necesidades que muchos trabajadores consideran prioritarias y que se relacionan con la satisfacción en el trabajo y la calidad de vida. Por lo tanto, el trabajo debe ser visto como parte del desarrollo personal, y no como un simple medio para obtener ingresos. Además, en esta época, con cualquier tipo de tecnología, es necesario el compromiso de los trabajadores para obtener un buen desempeño, pero "el involucramiento y el compromiso no pueden imponerse, sólo pueden darse y se dan sólo cuando surgen naturalmente desde los valores intrínsecos de la experiencia laboral" (Trist y Murray, 1993:328), es decir, cuando son tomados en cuenta en la organización misma del trabajo.

El término calidad de vida del trabajo, tiene el propósito de comprender las frustraciones y malestar en el trabajo que generaban baja moral y bajo desempeño. Se intenta superar la concepción taylorista de que los trabajadores son una extensión de la máquina, y se propone la vuelta al centro del trabajo, donde el trabajador es el actor clave en la producción. Se plantea la exigencia de contemplar necesidades como: requerir del trabajador un esfuerzo razonable, un trabajo variable y retador; posibilidad de aprender continuamente, reconocer y gestionar el apoyo social; formalizar premios sociales significativos; es decir hacer sentir y que el trabajador perciba objetiva y subjetivamente que el trabajo conduce a un futuro deseable (Trist y Murray, 1993:331- 588). La propuesta de atender estas necesidades psicológicas llevó al movimiento de calidad de vida del trabajo a plantear la necesidad de realizar transformaciones en la organización del trabajo. Recuperando la concepción de que lo que llaman el proceso de fabricación "incluye contingentes humanos, así como las instalaciones materiales" (Juran, 1990:152).

De las necesidades psicológicas mencionadas por el movimiento de calidad de vida del trabajo retoman las de aprender continuamente, que haya reconocimiento y apoyo social, realizar contribuciones sociales significativas, impulsar satisfacción por el trabajo y superación personal, que resulta de la posibilidad de emplear las capacidades propias, la

confianza en sí mismo y la realización (de emotividad), así como la utilización de la mente y el trabajar por voluntad propia (Ishikawa, 1986:24). Asimismo, afirman, hay influencias culturales que determinan el comportamiento laboral y, en algunos casos, conducen a la resistencia al cambio. Entre ellas están las "actitudes, creencias, hábitos, prácticas, símbolos de estatus, rituales, tabúes" que han sido construidos a lo largo de muchos años y sirven para mantener la organización tal y como está. Por tanto, la planificación para la calidad debe proponerse modificar ese ambiente cultural (Juran, 1990:238). Ishikawa indica que "El método Taylor no reconoce las capacidades ocultas de los empleados. Hace caso omiso del factor humano y trata a los empleados como máquinas". Por ende, el trabajo pierde interés y no produce satisfacción, por lo que no se pueden esperar productos de buena calidad y confiables. Por el contrario problemas como el ausentismo y la rotación expresan debilidades del estilo gerencial y descontento de los trabajadores (Ishikawa, 1986:22).

En síntesis los autores reconocen que la psicología permite identificar acciones que favorezcan a la empresa, al flexibilizar la resistencia de los trabajadores, al procurar su identificación con los objetivos de la empresa y generar, armonía obrero-patronal. La escuela de relaciones humanas propone modificar situaciones que generan insatisfacción y quejas en los trabajadores. La calidad de vida del trabajo considera que las acciones no sólo responden a una racionalidad económica, por lo cual es necesario generar satisfacciones de otro tipo para lograr el compromiso de los trabajadores con la empresa. El control total de calidad se amplía al anterior del proceso de trabajo para obtener, no sólo compromiso, sino la entrega de conocimiento de los trabajadores, sobre detalles del proceso de trabajo (principios toyotistas): y una de sus características es la interiorización de la supervisión y el control. Sin embargo, algunos análisis sobre la aplicación de técnicas de participación laboral han mostrado que conducen a un aumento del control gerencial o una pérdida de la autonomía obrera. Se afirma que hay una pérdida de autonomía por el destino colectivo del trabajo, la autorregulación, la movilidad interna, la mayor ingerencia de los departamentos de métodos y la participación limitada (Chanaro y Perrin, 1991). Hay una pérdida de control para los trabajadores con el aumento en la carga de trabajo, con la tutela más directa o la subordinación al software, con la transferencia de prerrogativas a los servicios funcionales, y con la sujeción a normas que aparecen como técnicas o científicas (Chanaro y Perrin, 1991). Hay un aumento del control gerencial por el oscurecimiento de los intereses de los trabajadores y la pérdida de su identidad

de clase, con orientaciones y limitaciones impuestas por la gerencia y donde quedan excluidos los temas de interés exclusivo de los trabajadores de nivel operativo (Wells, 1987, Trejos, 1998). El control se extiende al ámbito del conocimiento de los trabajadores operativos sobre detalles del proceso de trabajo y el de soluciones a problemas que sólo ellos pueden plantear (Wells, 1987, Trejos, 1998). El control se amplía para introducirse en la mente de los trabajadores, al lograr una participación voluntaria (o aparentemente voluntaria) y en la que se pone en movimiento no sólo la capacidad física, sino también la intelectual, para obtener mejoras que conducen al aumento en la ganancia (Wells, 1987, Trejos, 1998).

3.6.- La subjetividad y la acción laboral como socialización del saber-hacer

Uno de los debates actuales gira en torno al agotamiento o no de los supuestos teóricos tradicionales como los que hemos reseñado anteriormente frente a las nuevas formas de trabajo, las nuevas tecnologías y las nuevas formas de producción. Una dimensión en desarrollo es la de destacar lo aspectos subjetivos del trabajo. La perspectiva de la subjetividad implica aproximarnos a los significados que conllevan el *saber hacer* de los trabajadores, los tipos de control en el proceso de trabajo, la forma de organizar el trabajo, las prácticas laborales, los recursos de poder, negociación, creencias, ideas, imágenes, conflictos, resistencias, experiencias, habilidades, destrezas laborales, que como bien señala Zemelman (1997:21) *"...las ciencias sociales han vuelto crecientemente su mirada hacia los estudios de los valores, las creencias, las ideas, el conocimiento, el sentido común u ordinario que trasciende a los trabajadores pero que está presente en ellos y que resulta decisivo en la comprensión del otro, y en la construcción laboral del mundo. Todo esto es lo que de alguna manera esboza el campo de la subjetividad..."*[41]. Este componente subjetivo de la acción de los trabajadores, Heller (1980) lo entiende *".... como proceso de dar sentido a las propias prácticas que no se agotan en el significado de la práctica misma como resultado, porque el producto existe dos veces, uno como resultado y otro en la subjetividad del actor..."*[42]. Es la perspectiva sociológica de la subjetividad nos puede llevar a comprender mejor las formas de actuar del trabajador dentro del entramado de sentidos y significados en el cual surge cada acción. Aunque la acción implica construcción de significados, no se produce en un vacío de

[41] Zemelman, Hugo; León, Emma, "Sujetos y subjetividad en la construcción metodológica", en Sujetos y subjetividad, No. 6, UAM-X, México 1997, p. 21.
[42] Heller; Agnes, El hombre del renacimiento, península, Barcelona 1980.

sentidos. Una acción puede tener distintos sentidos en diferentes contextos o incluso puede cambiar su sentido en un mismo contexto a través del tiempo. Weber (1984) señala *"...que la acción lleva consigo sentido; ahí está el desafío que con facilidad olvidamos, estudiar el sentido de la acción: La actividad humana se orienta según un sentido que se trata de comprender para hacerla inteligible..."*[43].

La subjetividad supone estudiar la acción laboral, con toda la complejidad de sentidos, asociaciones, aspectos extralaborales y laborales; en particular de generación de conocimiento y de intercambio de información. Zemelman (1987:22) nos dice que *"...la subjetividad nos remite a una amplia gama de aspectos de la vida social (espaciales, económicos, políticos, culturales, laborales, corporales), ritmos temporales y escalas espaciales diferentes, desde los cuales se producen y reproducen redes de relación laboral... a partir de los cuales los trabajadores refuerzan sus vínculos internos y construyen una colectividad laboral que tiende a ser contrastante frente a otras..."*[44]. Es decir, la subjetividad la podemos comprender como un conjunto de representaciones desde la individualidad y colectividad laboral emergente, también anticipa y expresa un conjunto de acciones y reproducciones, de arreglos y conflictos que ayudan a la construcción de nuevos sentidos o refuerzan los ya existentes. La subjetividad no se limita a lo laboral, yuxtapone espacios domésticos, recreativos, sociales, religiosos; así como esferas de conocimiento, valores morales, estética, emociones.

De la Garza, explica que los actores laborales *"...no actúan ni dan significado sólo por su situación en las estructuras, pero para actuar pasan por el proceso de dar sentido y decidir los cursos de la acción..."*[45]. Así la subjetividad no es una estructura que da sentido de uno a uno, sino un proceso que pone en juego estructuras subjetivas parciales en diferentes niveles de abstracción y profundidad que se reconfiguran para la situación y decisión concreta. La subjetividad, en otras palabras, puede reconocer la discontinuidad, la incoherencia y la contradicción; así como la intencionalidad, el sentido, la identidad, la representación de intereses y luchas por el control del proceso de trabajo.

---

[43] Weber, Max, "Concepto de la sociología y del "significado" en la acción social", en Economía y sociedad, Ed. FCE, México 1984, pp. 18-21.
[44] Zemelman; León, op. cit., p 22
[45] Garza Toledo, Enrique de la "El papel del concepto de trabajo en la teoría social del Siglo XXI", en Tratado latinoamericano de sociología del trabajo. Ed. CM, FLCS, UAM y FCE, México 2000, pp 15-33.

3.7.- El trabajo del software embebido en una constelación de relaciones subjetivas

La dificultad para medir la eficacia en el proceso de trabajo del software radica en su propia naturaleza que podemos definir como *"un proceso de trabajo en el cual la materia prima son diferentes tipos de símbolos (diseño de los requerimientos del cliente) y el resultado es un conjunto de signos que se crearon a partir de descripciones, representaciones y significados" (reinterpretación de los requerimientos en algoritmos creados a partir de los lenguajes de programación)*[46]. Esta definición que exponemos se justifica en el proceso de trabajo del programador, donde concurre una serie de proceso que se traslapan:

- Se construye un conjunto de requerimientos del sistema a partir de entrevistas con el cliente. Es un diseño de ideas (símbolos) e imágenes que posee el cliente. ¿El cliente planteó correctamente los requerimientos?; ¿Son todos?; ¿Se interpretaron correctamente las necesidades del cliente?.
- El conjunto de representaciones y significados de los símbolos-ideas planteados por el cliente son re-interpretados en un diseño por los analistas, gerentes ó programadores (según el tamaño y organización de la empresa).
- El diseño de cierta forma representa una objetivación de las necesidades abstractas del cliente donde: ¿Es correcta la interpretación de los requerimientos del cliente por el diseño?; ¿El diseño es lo suficientemente comprensible por el cliente?
- Una vez planteado los requerimientos en el diseño, éstos son codificados a través de los lenguajes de programación; lo cual significa una nueva reinterpretación, que el programador hace de los requerimientos en algoritmos, que podemos definirlos como *un conjunto lógico de signos que tienen coherencia interna*, sin embargo es la composición de éstos algoritmos lógicos que nos lleva nuevas preguntas: ¿Están suficientemente estructurados como para resolver lógicamente el requerimiento para el que fueron creados?; ¿Poseen suficiente lógica interna de operabilidad con otros

[46] Esta definición la propongo a partir de que el cliente señala una serie de ideas -léase el ejemplo del gerente que solicita el desarrollo de un programa informático de una nomina de la empresa-, ideas que se enlistan en una serie de requerimientos y éstos dan principio a un conjunto de signos que el cliente desea que se desarrollen. Este conjunto de signos son los requerimientos plasmado en un diseño. La segunda etapa, es la re-interpretación de éstos signos-diseño por los analistas o programadores, quienes no sólo configuran nuevos iconos (tareas o rutinas en algoritmos) sino que la creación de dichos algoritmos, es a partir de un conjunto de símbolos, donde éstos representan o se construyen a su vez de símbolos determinados por los lenguajes de programación, que son una estructura de símbolos que representan determinadas acciones, por ejemplo, If (si ocurre a entonces es b); add (suma a con b), etc. etc.

algoritmos?; ¿El programador hizo los comentarios suficientes del *como se hizo*?; ¿Los algoritmos son suficientemente claros o requieren de comentarios al margen?.

- Los algoritmos desarrollados, operando en conjunto conforman la operación grafica del software, que una vez aprobado, implica una serie de incertidumbres que están relacionadas con la posibilidad de fallos, errores, incumplimiento de requerimientos, falta de calidad, etc.: ¿Existen los documentos del programa?; ¿Se inspeccionó que se llevaran a cabo los comentarios del programa[47]?; ¿Se examinó la calidad, métodos y herramientas empleadas?.

El conjunto de incertidumbres que se enfrentan en el proceso de trabajo, derivan en parte de una serie de dimensiones subjetivas que se intercalan, conformando un conjunto de *constelación de relaciones subjetivas* que se construyen formal e informalmente; representan intencionalidades y conflictos que pueden ser individuales o colectivos. Dicha constelación se integra por una serie de circunstancias subjetivas inherentes al proceso de trabajo, que se suceden en un conjunto de dimensiones embebidas en espacios claroscuros, opacos, donde podemos distinguir cuando menos tres tipos de subjetividades: subjetividad creativa, subjetividad sígnica[48] y subjetividad que se objetiva fuera del proceso de trabajo:

- La *subjetividad creativa*, tiene que ver con el proceso de construcción de decisiones de los individuos, con limites cognitivos; destrezas y habilidades de abstracción de los individuos; intencionalidad de los actores que interaccionan; practicas cotidianas al

[47] La documentación del programa es diferente a los comentarios del código. El primero hace referencia a dos tipos de textos: La documentación operativa del sistema que se desarrollo. En esta se explican los procedimientos y contenidos del software que se entrega al cliente. Se le conocen como tutoriales. Un segundo texto se refiere a la documentación administrativa del sistema. Hace referencia tanto al proceso de la licitación de requerimientos del cliente, el diseño de los módulos que integran el programa. Aquí se explican aquellos módulos que sean complejos, que no sean claros en su estructura, se explica que hace una clase o un método aplicado. El significado de las variables algorítmicas que integran el programa. Mientras que la documentación, hace referencia a la definición de códigos complejos, no evidentes; se aclaran aquellos bucles complejos (configuración de algoritmos), se declaran los nombres de las variables; se exponen aquellas dudas, incertidumbres, "candados" o procedimientos formales o informales que el programador realiza.

[48] El algoritmo resulta ser un texto de símbolos nomotéticos que a su vez está compuesto de signos que provienen del lenguaje de programación que se haya utilizado (existen mas de 2000 lenguajes); definiremos dicha texto algorítmico como una reflexión dialógica que se objetiva en un proceso "signico" que describen y representan los símbolos-ideas del cliente en algoritmos formulados lógicamente por el programador. Idea reformulada a partir Bajtín (2005, 1982) donde señala que cada texto, visto como enunciado es algo único, individual e irrepetible, en lo cual consiste su sentido. Donde el gesto natural en la representación efectuada por un actor, adquiere una importancia sígnica, por su carácter arbitrario, convencional y sometido a la intención del papel. Bajtín, M. (2005, 1982) Estética de la creación verbal. Editorial Siglo XXI, 2005. México. Pp. 296-297.

interior del proceso de trabajo, como pueden ser arreglos formales e informales, consensos y resistencias, boicot y luchas por el reconocimiento y espacios de poder.

- La *subjetividad sígnica* está representada por los textos sígnicos de los programadores, donde dichos algoritmos implican condiciones de abstracción, esfuerzo cognitivo propio, así como un alto grado de *interacción de pensamientos y movimientos reflexivos* que realiza el programador despóticamente, es decir en el proceso de trabajo no existe una "carpeta técnica de configuración del código", sino que el programador se abstrae, especula y refleja sus pensamiento e imágenes en un texto signico (conjunto de líneas de código que dan por resultado los algoritmos) que le precede un conjunto de habilidades, destrezas personales de conocimiento, flujo de información con otros, ya sea dentro o fuera del espacio laboral.
- *Subjetividad objetivada*, hace referencia al hecho que el programa de software, se crea en el proceso de trabajo a través de los distintos fragmentos (módulos). Una vez que los programadores terminan cada uno las tareas o módulos asignados, se compila y se "corre" por el tester o el responsable de calidad. Sin embargo, el software aún después de ser revisado y aprobado, empacado en el CD, USB o enviado por correo electrónico a las instalaciones del usuario, contiene en si un serie de contingencias ocultas (aflicción del software); pero también permanece invisiabilizado el poder oculto del software hasta que se despliega la interfaz gráfica en la pantalla del usuario final, hasta que el usuario opera y manipula el software, se asume que se *objetiva* a través de los distintos *pensamientos y movimientos* que sugiere el programa de software implementado.

En conjunto éstas subjetividades conforman una *constelación de relaciones subjetivas* que debemos precisar; por ejemplo la *subjetividad creativa*, se recrea y establece a partir del primer contacto, que por lo regular es una entrevista entre cliente-analista/programador; donde el cliente solicita el desarrollo de un software a la medida y enumera una serie de requerimientos o necesidades, esta relación/entrevista esta inmersa en un contexto subjetivo que puede derivar en incertidumbre, por ejemplo, porque las interpretaciones de los requerimientos del cliente pueden ser próximas o asertivas al problema, pero también pueden estar alejadas de las necesidades reales del problema. La naturaleza de la *subjetividad sígnica*,

reside en la configuración reflexiva del conjunto de signos que componen los lenguajes de programación (existen en la actualidad mas de dos mil lenguajes de programación) en el que se decide desarrollar el programa informático sea o no un lenguaje optimo; o bien que, el conjunto de configuraciones algorítmicas –que hemos denominado textos sígnicos– tengan o no cierto grado de cohesión lógica en su estructura interna (errores de configuración y problemas de interoperabilidad entre módulos, falta de experiencia o habilidades del programador, etc.). Hasta este punto, ambas subjetividades permiten señalar que el proceso de trabajo es un conjunto de interacciones que ponen en juego *signos creados a partir de símbolos*, proceso que está embebido de subjetividades que debemos definir, para así abordar las particularidades que acontecen en el proceso de trabajo creativo del software.

Para el caso del desarrollo de un programa de software a la medida no se "fabrica" de manera tradicional, sino que se construye, se elabora mediante técnicas cuasi-artesanales (flujo de información, conocimientos y aprendizajes entre expertos y aprendices; construcción e intercambio de significados) que tiene por resultado un conjunto complejo de *signos creados a partir de símbolos* que no son otra cosa que la transformación de los requerimientos o necesidades del cliente en *textos sígnicos* (algoritmos), algoritmos que solucionan e innovan un proceso solicitado por el cliente o usuario final. La innovación que el cliente posee en sus manos, por llamarlo de alguna manera representa un *proceso de simbolización de significaciones* que se esparcen en la pantalla del usuario final o cliente una vez puesto en operación el software, es hasta este momento en el que podemos señalar que concurre una tercera dimensión subjetiva: *Un proceso subjetivo que se objetiva* independiente al proceso de trabajo que lo originó. Se caracteriza por objetivarse en la pantalla, ya sea en la del cliente/usuario final o en cualquier otra pantalla independiente del cliente; de esta forma el programa de software a la medida posee la característica de objetivarse **n** número de veces. independiente al proceso de trabajo, es decir se independiza de aquellos que le crearon y se objetiva como un producto, con singularidades propias, por ejemplo el costo del segundo programa tiende a cero (el costo de copiarlo en un cd ó enviarlo por correo electrónico) como es el caso de la industria del "software pirata"; se realizan nuevas derivaciones del software a partir del interfaz grafico (clonación de software), y en caso de acceso al código, se modifica y crea nuevo software (re-ingeniería de software), etc.

Podemos decir que la *constelación de relaciones subjetivas* compuesta por las subjetividades ya señaladas (creativa, sígnica y objetivada) no sólo presionan y constriñen las acciones, prácticas individuales o colectivas, así como las interacciones sociales entre programadores; sino que paralelamente, concurren otros procesos intrínsecos al proceso de trabajo que cohesionan y presionan a los desarrolladores del software. Esto es que, si bien es cierto que el desarrollo del software se planifica, se sistematiza, se establecen requerimientos que cumplirse (objetivarse) en el programa (ejecutable para el usuario final), éstos no ocurren del todo, el proceso no es lineal o determinista, por el contrario la ingeniería del software plantea el tipo ideal en el proceso de trabajo, ya que la *aflicción del software* esta presente en la *interacción de pensamientos y movimientos reflexivos* en la creación de algoritmos (texto sígnico); es decir, la Ingeniería del Software no logra establecer plenamente metodologías o métricas de calidad que produzcan software eficiente, justo a tiempo y con cero errores.

Algunos investigadores, señalan que la imposibilidad de generar software libre de fallas y errores, crearlo justo a tiempo –considerado en la presente investigación como *aflicción del software*– obedece a un conjunto de contingencias intrínsecas al desarrollo del software; al respecto Pressman (2002) y Zahran (1998) señalan que el riesgo y la incertidumbre forman parte del proyecto, traduciéndose ello en costos que deben ser estimados. Zahran (1998) señala que el desarrollo de software debe percibirse como una tarea continua de desafíos. Desafíos que conforman parte de riesgo e incertidumbre. Bajo este contexto, académicos y empresarios señalan que para reducir al mínimo el riesgo e incertidumbre en el software, deben implementarse un conjunto de herramientas, métodos y métricas de calidad que conforman parte de la Ingeniería del Software -sobre todo la norteamericana- para así verificar procesos de calidad, gestión de proyectos, análisis formal de la arquitectura del software, conceptos y principios del diseño que configuren software confiable, métricas de prueba, etc. Sin embargo, a más de tres décadas de estar presente la Ingeniería del Software, el proceso de trabajo continúa embebido en riesgos e incertidumbres.

3.8.- El flujo de información en el proceso de trabajo simbólico: una perspectiva general

El proceso de trabajo artesanal de la década de los cincuenta y sesenta, parecería ser superado por la era "moderna" del software, a través de la puesta en marcha de complejas herramientas procedimientales de la Ingeniería del Software, sin embargo ello no es así, la incertidumbre es

una constante. Si bien es cierto en los aspectos formales parece haber una fragmentación de tareas, donde se identifican una serie de fases, que podemos señalar en cuatro grandes áreas: conceptualización; formalización, procesamiento de datos e implementación:

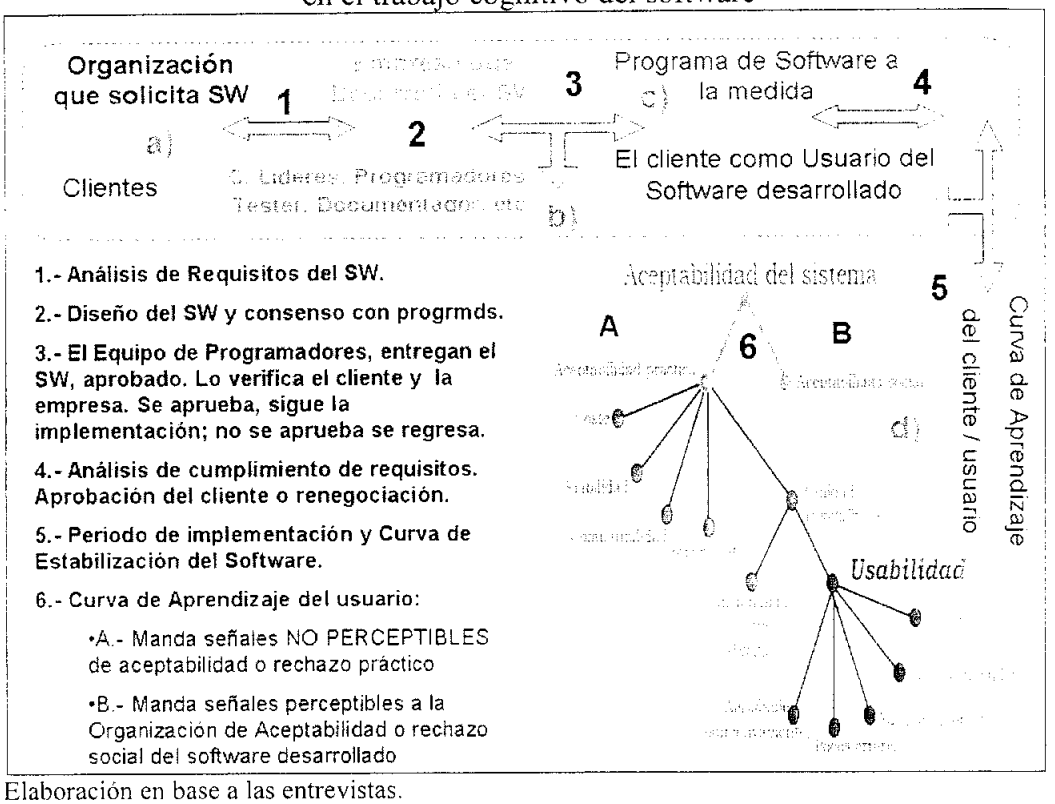
a) La *conceptualización* es el diseño de concepto-grafías de los distintos requerimientos que solicita el cliente/usuario que desempeñe el programa a desarrollar; se hace referencia a describir un conjunto de símbolos, que son representaciones de los requerimientos o necesidades del cliente/usuario; para ello se programan una serie de entrevistas donde se negocia y consensa las representaciones y significados de las necesidades del cliente a través de un diseño final de requerimientos. Este conjunto de símbolos, ocasionalmente -dependiendo de la formalidad de la empresa, tamaño del proyecto ó de las trayectorias administrativas- se conviene una firma de un contrato o cláusulas de requerimientos;
b) La *formalización* de los requerimientos del usuario, son examinadas, aprehendidas y modificadas por el analista, el arquitecto o el programador mas diestro en un diseño modularizado (depende de la complejidad de los requerimientos). El análisis orgánico de los requerimientos deviene en un proceso de conceptualización del problema y resulta en una formalización simbólica de los problemas a resolver.
c) El *procesamiento de datos*, esta circunscrita a proseguir una serie de consideraciones lógicas entre los requerimientos formalizados en un diseño o, concertados problemas a resolver. El diseño es un conjunto de símbolos que se subdividen en módulos, que poseen cohesión lógica; y éstos a su vez se fragmentan en tareas; las tareas contiene información que debe ser trasformada en aplicaciones; las aplicaciones son un conjunto de líneas de código (signos) que fueron creadas a partir de un lenguaje de programación (símbolos reconocidos socialmente), las líneas de código en conjunto representan textos sígnicos (algoritmos) que en conjunto resuelven el problema planteado por el requerimiento.
d) La *Implementación*, hace referencia al *proceso de simbolización de significados*, es la solución de los requerimientos planteados por el cliente (dependiendo del cliente, ya que podría ser otra empresa que hace las veces de "asesor de informática", ó podría ser un cliente contactado a través de Internet, etc.), e implementados en las instalaciones del cliente. En este punto, algunas establecen acuerdos de pos-venta, como asesorías en la capacitación de los usuarios finales, mantenimiento durante un periodo de tiempo, etc. ello dependerá de la organización y administración de la empresa desarrolladora.

Estas distintas fases laborales o fragmentación de tareas no son estáticas en el software a la medida, no son lineales, no son formales, son heterogéneas y varían de acuerdo al tamaño de la empresa, tipo de software que se desarrolla, cantidad de programadores disponibles, entre otros. Sin embargo es común que quien desempeña una función en un proyecto, se modifica en el siguiente; quien es analista, puede ser programador o líder; los puestos de trabajo son transitorios, parece estar relacionado con una serie de factores subjetivos como las habilidades, destrezas, experiencia, etc.; y objetivos, como que el programador posea acumulación de conocimientos para determinado proyecto. Las fases poseen fronteras difusas, ambiguas; no son rígidas, no están delimitadas, determinadas o estructuradas; por el contrario el transito entre una y otra es habitual; los limites entre programador, gerente, analista y desarrollador están invisibilizados y, son otras características subjetivas las que separan, como la experiencia, las habilidades y destrezas personales, arreglos sociales, consensos, negociaciones formales e informales, etc. que se construyen mediante determinados flujos de aprendizaje, no formales, flexibles, dinámicos y con alto contenido social.

Según damos cuenta en el esquema 5, intentamos concretar cual es el *flujo de información y aprendizajes* que se construye al interior del *proceso de simbolización de significaciones* en el desarrollo de software a la medida; este proceso se propone a partir de los cuatro incisos anteriores y, su posible distribución en el proceso de trabajo. Intentamos, situar cuales son las interacciones sociales que se suceden al interior del proceso de trabajo del software, las cuales no son lineales o casuísticas; ya que, como describimos en el apartado anterior, el hecho de que la eficiencia del software este supeditado, como veremos mas adelante, no sólo a la *aceptación técnica* del software por parte del cliente, también debe existir una *aceptación social* del software por parte de los usuarios finales; consideramos, que esta complejidad invisibiliza la potencialidad misma del proceso de trabajo de los programadores de software. Señalamos lo anterior, porque como hemos expresado el software *es un conjunto de signos creados a partir de símbolos* que por si mismo no se objetiva en el trabajo, sino que se objetiva en las instalaciones del cliente; lo cual nos lleva a plantear que hacia el interior del proceso de trabajo coexiste una doble paradoja, por un lado para objetivar la potencialidad el software se requiere de un hardware con la suficiente capacidad técnica para desplegar la potencialidad implícita del software, de otra manera un software que exige muchos recursos del hardware, sólo bloqueara el sistema, es decir se quedará "colgado" del

hardware; pero por otro lado, si el hardware es compatible se potencia una cualidad implícita del software, que se materializa la potencia del software en la medida que se "estabiliza" el mismo, es decir que coexiste una aceptación social y técnica del software por la empresa que lo adquirió (ver esquema 5).

Esquema 5
Proceso de simbolización y significaciones
en el trabajo cognitivo del software



Elaboración en base a las entrevistas.

Estas fases del proceso de trabajo se yuxtaponen, se traslapan, se integran en la pantalla del programador. Es decir, las complejidades y dificultades metodológicas para aprehender el conjunto de requerimientos en la *conceptualización* del problema, no se suceden en forma aislada, el diseño requiere del conocimiento, no sólo del analista también de las sugerencias del programador. En el *procesamiento de datos* hay un constante ir y venir entre el diseñador y el programador, coexisten una serie de consensos formales e informales en esta interacción cotidiana en el desarrollo del programa; también interviene la tradición organizacional de la empresa, si implementa acercamientos continuos con el cliente para valorar los avances y negociar los avances. Ahora bien, considerando que las fronteras entre las fases del trabajo en el desarrollo del software se desdibujan, no es menos cierto que cobran

fuerza una serie de independencias individuales al interior del propio proceso de trabajo, como es la toma de decisiones para transformar los requerimientos contenidas en el diseño en un conjunto de algoritmos (*subjetividad sígnica*) que corresponde a una toma de decisión individual, reflexiva que incluye una serie de subjetividades implícitas como la voluntad e intencionalidad del programador o programadores en la coherencia lógica de la cadenas de algoritmos que conforma el módulo (arreglárselas para saber resolver); decisiones individuales de documentar o no, parcial o totalmente el proceso, de explicar u ocultar detalles o autolimitar la descripción de la lógica cognitiva de aquellos algoritmos complejos; las *interacciones de pensamientos y movimientos reflexivos* están impregnadas de sentidos, fines, emociones, juegos, arreglos, consensos, negociaciones, etc. Es decir, el proceso de trabajo esta embebido en una *constelación de relaciones subjetiva;* donde ésta *constelación* está representada por un conjunto de arreglos, consensos, conflictos, resistencias, relaciones de poder, ya sean individuales o colectivos que se inscriben, producen y reproducen formal e informalmente en las interacciones sociales cotidianas al interior como al exterior del proceso de trabajo. Éstas representan la oportunidad de concebir, ejecutar y generar relaciones de poder y paralelamente se recrean las tareas requeridas en un contexto de *aceptación técnica* y *aceptación social.*

La *aceptación técnica* del programa de software hace referencia al hecho de que el cliente reconoce que el programa contiene los requerimientos que se plasmaron en la lista de requerimientos, esta de acuerdo en los dispositivos gráficos del software. La *aceptación social* hace referencia a la *curva de aprendizaje* del usuario final y a la *curva de estabilización* del software en las instalaciones del usuario final; del cruce entre ambas curvas, resultara una *aceptación social* del software implementado en las instalaciones del usuario final. Sin embargo, también implica lo contrario, es decir el rechazo, la negación, el conflicto, la resistencia, etc. aquí entran en un inter-juego de negociaciones y consensos las significaciones, las percepciones, sentidos, gustos, preferencias, entre los usuarios finales frente al conjunto de símbolos que se despliegan en la pantalla (interfaz gráfica).

Distintas normas disciplinares intentan explicar este proceso, desde la Ergonomía del software y la Ingeniería de la Usabilidad, hasta la Ingeniería del Software. Es decir, que el software desarrollado no sólo debe ser usable en cuanto al contenido, sino que se enfrenta a un

conjunto de resistencias culturales y boicot en cuanto a la implementación, porque se fracturan cotos de poder culturizados de poder entre los usuarios finales.

> "No es nada más que sea lógico y que salga cómo salga un formato por ejemplo, tienes que hacerlo... que diseñarlo para que sea una buena presentación, aparte del buen funcionamiento, son las dos cosas. Primero el buen funcionamiento pero también buena presentación, amigable al usuario porque si vas a poner un proceso de saturación que nadie va a entender, aunque funcione muy bien, no lo van a saber manejar no les va a servir (Programadore, DASA CB3).

3.8.1.- Curva de estabilización del software

La *curva de estabilización* del software hace referencia al tiempo requerido para que se lleven a cabo una serie de revisiones prácticas, desde aquellas que verifican que tan "amigable" es el interfaz grafico del sistema, pasando por el de operabilidad con el hardware existente, y el de interoperabilidad con otros sistemas informáticos (en el caso de existir); por ejemplo si se "conecta" a una red externa, a Internet, Intranet, si existe comercio electrónico, etc. etc.,

> "...Ves pantallas, ves interfaces de usuario, etc. que están concebidas pensando en el dato, no en el usuario; pensando en la información, no en el proceso. Entonces, llega el usuario se sienta enfrente de la computadora y para registrar un tipo de transacción tiene que navegar por dos, tres, cinco o siente pantallas para registrar una sola cosa. Segundo, el lenguaje que se utiliza en los sistemas de información cae con una gran facilidad, por lo menos se aleja mucho normalmente del lenguaje de negocios que está acostumbrado a utilizar el usuario... Ya van dos. Tercero, no asumen una realidad que es natural del ser humano, el ser humano se equivoca..." (Líder DYYA RF1)

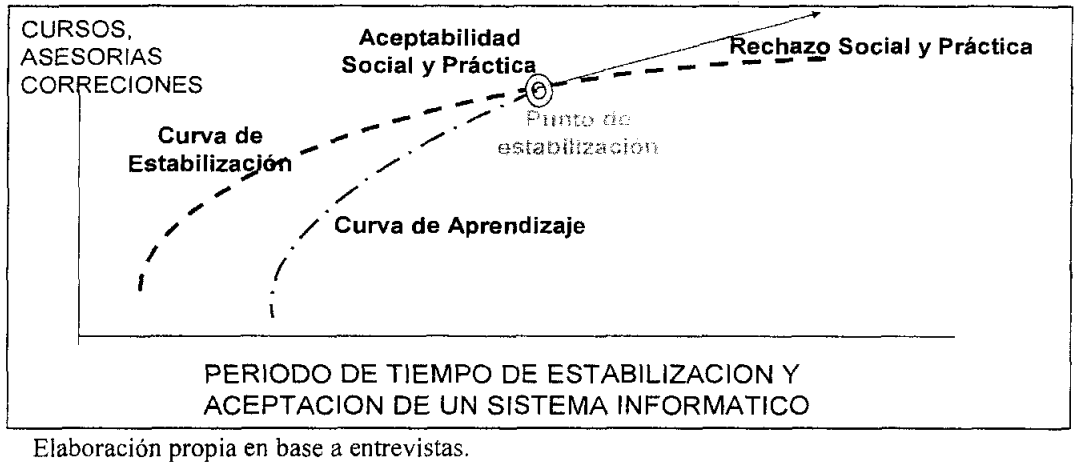
3.8.2.- Curva de aprendizaje del software

El diseño grafico, también denominado interfaz grafica del software, significa una serie de dispositivos como presentación de pantallas, iconos, colores, trazos y figuras que extienden información en la pantalla. Las condiciones de estos dispositivos han sido abordadas por las distintas normas disciplinarias como la Ingeniería de la Usabilidad y por la Ergonomía del Software. La Ingeniería de la Usabilidad, intenta prever el abanico de posibilidades externas al proceso de trabajo del software que afecten la calidad, desempeño y eficiencia del software. La Ingeniería de la usabilidad señala que 30% de los errores en los sistemas informáticos no radica en los algoritmos desarrollados, sino en el conjunto de dispositivos contextuales, formatos, colores y exposición icnográfica que no son del todo claros ó comprensibles para los usuarios finales. Consideremos entonces que puede existir un periodo de tiempo en el cual el

usuario final, aprehende, conoce y desarrolla habilidades para operar el nuevo Software. A este periodo le denominamos *curva de aprendizaje*, periodo en el cual el usuario final aprehende los tiempos y pensamientos requeridos para el manejo y transformación de la información disponible en las pantallas en conocimiento y éste en aprendizajes de los dispositivos (puede estar incluido el cliente; ver esquema 6).

Esquema 6

Estabilización técnica y aceptabilidad social en el software a la medida



Elaboración propia en base a entrevistas.

Paralelamente, a la etapa de aprendizaje del software, esta el hecho que se esta a prueba el software con el cliente, donde puede haber problemas de operabilidad con el hardware, errores de configuración del software al momento de instalarlo en las computadoras del cliente. Sin embargo, también es cierto, que existe una resistencia por parte del usuario final, resistencia que se puede traducir en boicot al uso del software, en señalamiento de que no se "comprende" los iconos que se despliegan en la pantalla, etc. Es decir, consideramos que existe una curva de estabilización del software, no solo del tipo técnica, sino del tipo de aceptación o rechazo social del software (Gonzalo, 1991; Marck y Rigby, 1994; García y Sánchez, 1994).

Ambas curvas, tanto la de aprendizaje como la de estabilización pueden converger antes, sin embargo, dependerá del grado de aceptabilidad social o rechazo en la práctica del software al interior de la empresa, que prolongara el punto de estabilización, o en su defecto ello se puede traducir en un rechazo al sistema por parte del cliente. Esta *contingencia atípica* en la industria, es con el fin de señalar el grado de intervención y dependencia que el proceso de trabajo del software se supedita al uso eficiente por el lado del cliente y usuarios finales. Es

decir, si bien es cierto en el proceso de trabajo del software se culmina el programa mismo, este solo objetiva su potencialidad cuando esta operando en las instalaciones del cliente, cuando es aceptado técnica y socialmente el software desarrollado a la medida.

El periodo de estabilización del software, a su vez está inmerso en dos "transacciones sincrónicas" objetivas y subjetivas. Podemos decir, que la *curva de estabilización* del software es objetiva, tangible, en el sentido que hace referencia al periodo de compensar y corregir posibles fallas o errores de "compatibilidad" con otros sistemas informáticos o con el hardware existente. Por otro lado también existe una serie de arreglos y negociaciones de índole subjetivo, intangible, que tiene que ver con el periodo de tiempo en que los usuarios finales aceptan, rechazan, aprehenden o comprenden el software implementado; es decir existe una "*curva de aprendizaje del sistema informático*"; por tanto tenemos que el desarrollo del software no se circunscribe sólo al periodo que comprende al proceso de generación signos a través de símbolos que dan origen al software al interior del proceso de simbolización de significaciones, sino que tiene que ver con dos procesos simultáneos y sincrónicos: la aceptación técnica y social del software implementado en las instalaciones del cliente.

3.9.- El proceso de trabajo cognitivo. Una mirada desde los actores

La ingeniería del software ha centrado y concentrado sus análisis en formalizar métodos y herramientas para el procedimiento, organización y desarrollo sistemático en el desarrollo de software[49]. Por ejemplo, el estándar IEEE Std 610.12-1990[50] especifica que la ingeniería del software se define como "la aplicación de un enfoque sistemático, disciplinado y cuantificable del desarrollo, operación y mantenimiento del software", preocupación que ha llevado a que el diseño grafico del sistema, el soporte y asesoría al usuario final pos venta haya sido relegado. En otras palabras, la ingeniería del software ha orientado sus esfuerzos al "proceso de desarrollo sistemático y disciplinado" haciendo a un lado las prácticas cotidianas, el flujo de conocimientos tácitos; que según dimos cuenta en las entrevistas -como veremos más adelante- los programadores y lideres se orientan más por las experiencias y destrezas individuales en proyectos similares ya desarrollados con anterioridad, ya que suponen

[49] Véase Jakob Nielsen's Alertbox. Disponible http://www.useit.com/alertbox/20030310.html; Nielsen, J.; Mack, R.L. (1994). Usability Inspection Methods. John Wiley & Sons, New York, NY.; Granollers, T.; Perdrix, F.; Lorés, J.: Grupo de Investigación GRIHO. Universidad de Lleida, España.;

[50] IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE. 1990.

implícitamente que si resulto el *como se hizo* en un proyecto anterior no tiene porque ser distinto en el siguiente proyecto. Sin embargo, está suposición, no es del todo verídica, ya que cada cliente, plantea una serie de problemáticas heterogéneas.

> "...comprender que es lo que quiere el cliente, ya sea que el líder nos lo diga o el cliente directamente nos diga que es lo que quiere. Como programador nos facilita un poco la vida por que lo hacemos una vez...pero si por ejemplo el líder nos pasa ese requerimiento con fallas, que no este bien definido, entonces lo hacemos pero ya después nos lo regresan "No, sabes que, es que así no, necesito esto y esto (...) debe estar seguro, debe tener toda la información necesaria, debe saber lo que en realidad quiere el cliente, para que uno lo pueda hacer bien" (Programador MEKI CM3).

En las entrevistas los programadores señalaban que es importante detectar posibles "vacíos de información", identificar potenciales conflictos de intereses, conflictos que pueden ser tanto con los usuarios, como con el entorno de hardware y software con el cual inter-operara el sistema a desarrollar

> "En ocasiones NO se comprende... o luego por ejemplo muchos clientes entre ellos luego no se ponen de acuerdo, has de cuenta que el de sistemas y el de ventas o el gerente de mercadotecnia piden cosas que no se ponen de acuerdo, el de sistema dice "No, hay que hacerlo así"...el de Producción "Oye mira este es el software"... "Ah, es que mira esto no funciona así, o no me gusta" (Programador 19 MEKI CM3).

Ahora bien, el conjunto de requerimientos no sólo se corresponden a aquellas demandas establecidas por el cliente, sino que existen una serie de factores externos e internos[51] que

[51] Requisitos de comunicaciones del sistema: Son de carácter técnico relativos a las comunicaciones que deberá soportar el sistema software a desarrollar. Por ejemplo: el sistema deberá utilizar el protocolo TCP/IP para las comunicaciones con otros sistemas. Requisitos de interfaz de usuario: Se especifica las características que deberá tener el sistema en su comunicación con el usuario. Por ejemplo: la interfaz de usuario deberá ser consistente con los estándares definidos en IBM's Common User Access. Se debe ser cuidadoso con este tipo de requisitos, ya que en esta fase de desarrollo todavía no se conocen bien las dificultades que pueden surgir a la hora de diseñar e implementar las interfaces, por esto no es conveniente entrar en detalles demasiado específicos. Requisitos de fiabilidad: deben establecer los factores que se requieren para la fiabilidad del software en tiempo de más uso del sistema. Se mide la probabilidad del sistema de producir una respuesta satisfactoria a las demandas del usuario. Por ejemplo: la tasa de fallos del sistema no podrá ser superior a 2 fallos por semana. Requisitos de entorno de desarrollo: Se especifica si el sistema debe desarrollarse con un producto específico. Por ejemplo: el sistema deberá desarrollarse con Oracle 7 como servidor y clientes Visual Basic 4. Requisitos de portabilidad: Definen qué características deberá tener el software para que sea fácil utilizarlo en otra máquina o bajo otro sistema operativo. Por ejemplo: el sistema deberá funcionar en sistemas operativos Windows 95, 98 y Windows NT 4.0, siendo además posible el acceso a través de Internet usando cualquier navegador compatible con HTML 3.0. Amador Durán T, y Beatriz Bernárdez J. Metodología para la Elicitación de Requisitos de Sistemas Software, Versión 2.1, Departamento de Lenguajes y Sistemas Informáticos, Facultad de Informática y Estadística, Sevilla, España, Octubre de 2000.

deben ser considerados al momento de proponer el conjunto de documentos que integran los requisitos del sistema. Consideramos que este proceso es heurístico del tipo cliente-empresa-cliente, generándose una serie de versiones del documento de requisitos, sin embargo, supongamos que ya se firmo el conjunto de requisitos, el cual constituye el documento base mediante el cual se creara el diseño que el arquitecto de software, ingeniero o programador que este involucrado en las entrevistas tomara en consideración para iniciar la parte de diseño y configuración del proceso de trabajo. Este punto, es crucial, porque de aquí dependerá[52]: el cumplimiento de requisitos; cumplimiento de fechas de entrega; propuesta de modularización del proyecto; asignación de recursos humanos, ciclo de vida; costos por horas-hombre; presupuesto del proyecto. Este proceso, es complejo, depende de muchos factores. Un líder de proyecto, experto en desarrollo, nos comentaba que si el cliente posee un cierto grado de cultura informática disminuyen las fallas, lo cual no quiere decir que desaparezcan:

> "...depende mucho de la cultura del cliente. Sí el cliente esta en algún nivel de cultura informática se minimizan los errores pero no se eliminan. Sí tienen menor cultura informática puedes llegar a tomar decisiones que afectan directamente la posibilidad de generar software correspondiente" (Líder CATI DC2).

La lista de requisitos, representa un conjunto de significaciones cognitivas, que proponemos es un conjunto de símbolos que se negocian entre el cliente y el que lo entrevista, éstos se traducirán en el diseño final, una especie de matriz de requerimientos, datos o requisitos que especificaran las características de funcionalidad y operabilidad del software.

> "Se firma un contrato y una cotización para aclarar: tu me estas pidiendo esto, tu solución es esta, te cuesta tanto, nos vamos a tardar tanto, y me tienes que pagar esto, a y te voy a capacitar por tantas horas, a tantas personas y, aparte te voy a dar soporte por tanto tiempo para que lo pruebes y si hay anomalías te las arreglo, solo que si hay nuevos requerimientos determinamos si están incluidos o no en el contrato (Programador TADI MZ3).

[52] Véase Brooks, 1987; Toval et al., 2001 y Pressman, 2002, quienes asocian la correcta especificación de los requisitos del producto de software con la satisfacción de las necesidades y expectativas del cliente. Brooks, F. P. No Silver bullet. Essence and accidents of software engineering. IEEE Computer, Vol. 20, No. 4, pp. 10-19. 1987; Toval, A., Nicolás, J. y Moros, B. Siren: Un proceso de Ingeniería de requisitos basado en reutilización. Jornadas de ingeniería de Requisitos Aplicada, Sevilla, pp 129-143. 2001; Pressman, R. S. Ingeniería del Software: Un enfoque práctico. Quinta Edición, McGraw Hill, Madrid. 601 p. 2002. La Ingeniería de Requisitos explica las técnicas de entrevistas, la identificación de conflictos, etc.

Como una segunda estrategia para corroborar que los requerimientos son los adecuados para el cliente, se diseña un prototipo rápido, ya sea un esquema "vacío" es decir, se presentan pantallas sin contenido, sin datos, solo a nivel de esquemas. Y, en caso que haya un repositorio de módulos equivalentes a los que solicita el cliente, se presenta un prototipo de diseño rápido.[53] Sin embargo, no es lineal, como ya señalamos, sino heurístico, casuístico:

> "Esto es parte de la naturaleza humana... un programador por cuidadoso que sea, escribe todas sus líneas de código, lleva a cabo todas las pruebas que se le ocurrieron , sin embargo, las pruebas que no se le ocurrieron pues no las hizo... llega el momento de la aplicación, pasa todas las etapas betas[54], con todas las pruebas que se le ocurrieron a la compañía que desarrollo el sistema y al momento de salir a producción (etapa de implementación con el cliente) aparecen todos los casos de uso que a esta compañía no se le ocurrieron probar y resulta que muchos casos de uso no los habían contemplado, y resulta un error (Programador DINS 03)

También se pueden detectar problemas de inter-operabilidad:

> "Nada se entiende con nada, no embonan por más compatibilidad que haya. ¿Por qué? Porque va requerir un tiempo (...) de estabilizarse para empezar a operar de manera más o menos adecuada. Y hay un sinnúmero de variables. Así de complejo como es una solución de este tipo. (...) Ya no pensemos toda había en la siguiente fase que es poner en marcha el ERP, o poner en marcha las otras aplicaciones" (Líder DYYA RF1)

Se asume que hay problemas indisolubles al sistema, por tal motivo se les da garantía:

> "incluso cuando ya se entrego al cliente, el mismo cliente va encontrar errores y, por eso aquí se manejan un año de garantía (...) nosotros les damos el soporte constante, constante, y... a corregir errores" (P11 DASA CB3. Programador/Líder)

[53] Aquí el modo de desarrollo de software libre, posiblemente le lleve una ventaja al sw privado. En el sentido que en Internet existen muchos sistemas libres de los cuales se puede disponer sin costo, salvo el que implica poseer los conocimientos de implementación y manipulación del código.
[54] Una etapa beta es un una unidad de prueba, de análisis, que no se libera.

Capitulo IV

Comunidades simbólicas de trabajadores cognitivos en la industria del software a la medida

Un gerente fue con sus Desarrolladores y les dijo: "Con respecto a sus horas de trabajo: Ustedes van a tener que venir a las nueve de la mañana e irse a las cinco de la tarde." Con esto, todos se enojaron y varios renunciaron en el momento. Entonces el gerente dijo: "Está bien, en ese caso ustedes pueden fijar sus propias horas de trabajo, siempre y cuando terminen sus proyectos a tiempo." Los Desarrolladores, ahora satisfechos, comenzaron a llegar a mediodía y trabajar hasta la madrugada.
Tao de la Programación

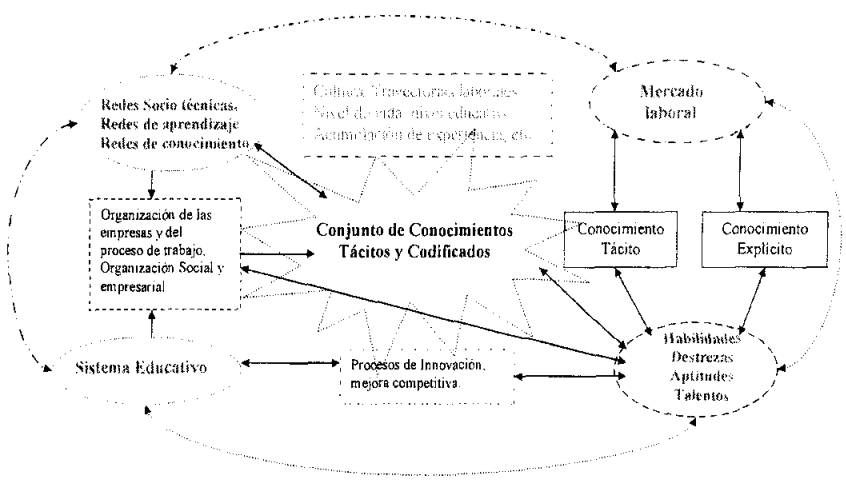
4.1. Introducción

En el presenta apartado abordaremos un debate teórico en torno a las características principales del capitalismo moderno, como el debate en torno a la presencia o no del ascenso de la denominado *economía del conocimiento* la cual implica la no centralidad del trabajo como generador de valor (Castells e Himanen, 2002), el componente clave de esta "nueva" sociedad es un componente abstracto como la información y el conocimiento (Hardt y Negri, 2000). Otros enfoques agregan la interacción social entre agentes del tipo red (Yoguel, 2003); así como la tendencia de nuevas formas de organización del trabajo que reducen la dicotomía del trabajo intelectual (concepción) y del manual (ejecución) (Luna, 2003); separación que, algunos autores proponen que hoy día es articulada por el trabajo inmaterial, que éste se constituye como carburante de las nuevas formas de trabajo del capitalismo moderno (Lazzarato y Negri, 2001; Hardt y Negri, 2000). Sin embargo, consideramos que éstas propuestas obscurecen el trabajo frente al conocimiento o bien la nueva alineación subjetiva del trabajador frente al trabajo inmaterial ésta inmersa en una serie de confusiones. Proponemos que hacia fines del siglo XX cobran relevancia cierto tipo de industrias que producen bienes simbólicos como el consumo de espectáculos y entretenimiento, el comercio electrónico, la publicidad virtual, el desarrollo de software, entre otros ejemplos; donde el proceso de producción y consumo, oferta y demanda no están separados del todo, por el contrario las esferas de producción y consumo, demanda y oferta están muchas veces traslapadas, yuxtapuestas en el acto mismo. Este tipo de producción simbólica rompe con varias particularidades del ciclo productivo de la economía industrial clásica. Por ultimo proponemos una serie de características del proceso de trabajo en el desarrollo del software como trabajo cognitivo, un tipo de producción simbólica.

4.2.- Nuevas formas de producción y crisis del esquema taylorista

Cuando se aborda el tema del trabajo en el marco del "uso" extendido de las tecnologías de la información y comunicación (TIC), se acentúa que el conocimiento y la información representan el componente clave de la *economía del conocimiento* (Rifkin, 2004; Castells, 1991, 2001; Barman, 2006; etc.). Además, uno de los argumentos más insistentes en el análisis sociológico, es el discurso de la interacción social entre agentes del tipo red. Para Yoguel y Fuchs (2003), Casalet (2002, 2006) y Casas (2001) las empresas líderes en la *economía del conocimiento* se organizan a través de sistemas socio-institucionales, bajo un enfoque que es la construcción social de una nueva institucionalidad entre agentes que intervienen en la industria a través de una mayor cooperación en distintos niveles o escalas, macro, meso y micro. A este sistema se agrega el nuevo enfoque del proceso innovador, según el cual los agentes sociales-locales también aprenden y generan conocimiento a partir de sus prácticas productivas y la recombinación del conocimiento codificado, tácito y codificable al interior de las organizaciones, de las redes y sistemas territoriales de las que forman parte. Se revalorizan así las innovaciones incrementales, derivadas no sólo de actividades formales de investigación desarrollo e innovación (I+D+i), sino también por procesos como *aprender haciendo; aprender usando* y *aprender interactuando* (Yoguel y Fuchs, 2003); Pavitt, Nonaka y Takeuchi, 2001).

Esquema 7
Sistema Socio-institucional



Fuente: Elaboración propia en base a Borello et.al. 2005, CEPAL número 87, pág. 138

Este tipo de análisis en red, según vemos en el esquema 7, es considerado como estructural-evolucionista, se deriva de una serie de premisas neoclásicas implícitas, en las que los individuos se ven reducidos a una conducta social en red, asociados en términos de recompensas motivadas por la interacción de sus lazos fuertes y débiles, en una relación no muy clara entre elección racional y conductismo psicológico (De la Garza, 2006:33). Esta corriente teórica iniciada por Granovetter, hace a un lado los problemas de poder y resistencia entre individuos (destacados ampliamente por M. Foucault y Edwards). Este tipo de enfoque institucional y de la teoría de la empresa no consideran los diferentes conflictos, resistencias, consensos y enfrentamientos que se suceden en el piso de la empresa, así como los juegos, arreglos sociales formales e informales que acontecen en la interacción entre individuos (destacados por Burawoy); el enfoque de redes se orienta más a las nuevas relaciones industriales entre agentes, propuesta por la teoría del Managment o las corrientes consideradas como los nuevos tayloristas (neotaylorismo). Otras vertientes señalan que los cambios recientes en la organización del trabajo y en las relaciones laborales de fin de siglo, son mediados por el uso de nuevas tecnologías complejas, modulares y de rápido cambio técnico que transforman la organización socio-técnica de las viejas formas del fordismo-taylorismo.

Las actividades productivas han variado en contenido y en formas de organización. La producción estandarizada, poco a poco va cediendo terreno, se transforman algunos sectores productivos y mercados a favor de una producción especializada, flexible, polivalente, con menores tiempos y costos de producción (flexi-trabajo, flexi-produccion, etc. OIT, 2004). Los principios de competitividad y productividad evolucionan asociando las nociones de valor agregado, novedad y calidad del producto; recomposición permanente de tareas, flexibilidad organizacional, aprendizaje, creatividad e innovación (Hardt y Negri, 2002; OIT, 2000; De la Garza, 2006a). Cambios que hacia fines de siglo XX se constituyen como crisis del esquema taylorista, del obrero "disciplinado" antes que "educado"; fin del trabajo simple y rutinario, desprovisto de una dimensión reflexiva y a la vez creativa que comenzó a perder vigencia desde la crisis de los setenta, recobrando auge temporalmente ante el cambio de una productividad basada en la organización funcional de la producción, fundamentada en una organización tecnológica provista por una nueva economía del conocimiento de fin de siglo.

La propuesta de la *economía del conocimiento* (Villavicencio, 2005, 2006; Lucena, 2006; Castells, 1998, 2001) parecería ser que representa cambios en las relaciones laborales

tendientes a reducir la brecha que separo el trabajo intelectual (concepción) del manual (ejecución), ¿Acaso estamos en presencia de un naciente restablecimiento del *saber-hacer* del trabajador?, interrogación que ha llevado a algunos teóricos a proponer conceptos nuevos basados en el conocimiento (Neffa, 2006). Otros autores van más allá, como Castells (1998, 2001) quien señala que en esta nueva *economía del conocimiento* el trabajo se transforma en *trabajo autoprogramable*, el cual está dado por la facultad de redefinir las propias capacidades individuales ante los cambios tecnológicos. El punto neurálgico de esta forma de trabajo es el conocimiento como carburante (Tofler, 1998) "*Lo que importa, más que unas cualificaciones, es una capacidad general educativa de cultura general, de capacidad de asociación, de saber cuáles son las cualificaciones que necesitas para las tareas que tienes que hacer, dónde buscarlas, cómo aprenderlas y cómo aplicarlas. Para entendernos, un nivel intelectual general, lo cual implica toda una redefinición del sistema de educación: la capacidad social de hacer pasarelas entre el trabajo y la educación*" (Castells y Esping-Andersen, 1999).

En la misma línea teórica M. Lazzarato y A. Negri (2001) señalan que las características del trabajo en la *economía del conocimiento* son las posibilidades de redefinir las capacidades del trabajo mismo y la toma de decisiones por parte de los obreros, acentuándose una mayor disposición subjetiva de los trabajadores, que denominan *interfase*[55]. Para los autores, la disposición subjetiva del obrero significa un nuevo tipo de alineación que los autores entienden como amoldamiento subjetivo del obrero a través de su concepto de *interfase*. Para nosotros éste concepto denota no solo falta de creatividad conceptual, sino insuficiencia o agotamiento de las categorías clásicas del trabajo industrial para abordar las distintas configuraciones que adquiere la producción inmaterial de fin de siglo. Lazzarato y Negri (2001) proponen que el *trabajo inmaterial* se entienda como una significación que logra enlazar más acabadamente la nueva modalidad de amoldamiento subjetivo del obrero que se instaura bajo las nuevas condiciones de trabajo post-industriales.

Consideramos que nuevamente los autores no son capaces de superar el lastre taylorista-fordista al considerar al trabajador como un ente externo al proceso de trabajo, como una extensión de la maquina, haciendo a un lado las intenciones, sentidos y representaciones propias del significado del trabajo por el trabajador; se intenta nuevamente aprisionar al

[55] Los autores proponen el concepto de interfase para hacer referencia a distintas funciones en la organización de equipos de trabajo y distintos niveles de jerarquía laboral; interfase que es posible porque la propia subjetividad del trabajador ya ha sido moldeada, Lazzarato y Negri, 2001.

trabajador como un sujeto *alineado subjetivamente* a las nuevas condiciones que impone el estadio capitalista de producción inmaterial, rechazando nuevamente –como en su tiempo lo hizo el taylorismo y el fordismo– las constelaciones de relaciones subjetivas que caracterizan a los individuos dentro y fuera del proceso de trabajo.

La noción semántica de *trabajo inmaterial* propuesto por Lazzarato y Negri (2001) estaría dado por la actividad productiva donde se hace imposible distinguir trabajo y satisfacción subjetiva del gusto por el trabajo; en la actividad productiva actual la importancia recae en el saber social general, sea éste producto del desarrollo científico o de la interacción social. Este saber social general ha ido modificando el ciclo de la producción, redefiniendo la relación producción/consumo; es decir, por un lado la producción deja de ser fabricación de grandes cantidades de objetos estandarizados, en serie para convertirse en un proceso que, desde el diseño del producto considere las preferencias del consumidor (fin del fordismo taylorismo); por otro lado en el ámbito del consumo existe todo un esfuerzo por activar las necesidades, sentido del gustos, apreciación estética de los consumidores, ello través de un proceso de comunicación social entre producción y consumo, que figurado por el trabajo inmaterial, ya se considere éste como producción de objetos inmateriales (producción audiovisual, software, o de otros productos de comunicación) o como modalidad predominante del capitalismo moderno.

Consideramos que en la producción inmaterial el proceso de comunicación e información no simplemente acerca a consumidores y productores, sino que forma parte de la producción misma, estableciéndose nuevas relaciones sociales de producción, donde la materia prima -desde una perspectiva más amplia- son las condiciones subjetivas de los consumidores. La subjetividad se considera no como instrumento de control social o como reproducción de las relaciones de mercado, sino que es condición de posibilidad para la forma de producción inmaterial (Lazzarato y Negri, 2001; Hardt y Negri, 2003). De esta manera, el capitalismo ha tomado todas las dimensiones de la vida "*el trabajo inmaterial produce al mismo tiempo subjetividad y valor económico tan sólo demuestra en qué medida la producción capitalista ha asaltado a la totalidad de la vida y ha roto todas las oposiciones entre economía, poder y saber*"[56].

[56] Lazzarato, M. y A. Negri, *Trabajo inmaterial. Formas de vida y producción de subjetividad*. DP&A Ed. Río de Janeiro, 2001 Disponible: http://www.rebelion.org/libros/TrabajoInmateria011202.pdf. Acceso 05/09/07

Esta fijación casuística de los autores con respecto a la supuesta plasticidad de las subjetividades de los individuos (interfase), es resultado de una serie de confusiones, la primera tiene que ver con el supuesto estadio de la nueva *economía del conocimiento* (también denominada *revolución de la inteligencia*)[57] que schumpeterianos y evolucionistas comprenden como una convergencia entre industria y servicios, mediada por un mayor uso de información y conocimiento; Zarifian le nombra *producción industrial de servicios* (Zarifian, 1999); bajo esta premisa los autores construyen el concepto de producción inmaterial, donde la materia prima corresponde a una nueva alineación subjetiva del obrero y tiene por objetivo principal, un sujeto consumidor/comunicador (Lazzarato y Negri, 2001). De esta forma los autores limitan lo inmaterial a la subjetividad de los individuos en un marco racionalista, como si la subjetividad no constituyera una constelación de interés y sentidos subjetivos del trabajador.

Una segunda confusión de los autores en cuanto a la producción inmaterial que proponen, es que homogenizan la producción industrial con la de servicios, señalando que toda *producción industrial de servicios* es *producción inmaterial*[58]. Los autores incurren en una tercera confusión al igualar ambos términos (manufactura, servicios e inmaterial) sustentando sus argumentos en la progresiva convergencia entre la esfera de producción y la esfera del consumo, traslape que significa mayor necesidad de aprovisionar a los clientes de servicios a la "medida" como consecuencia de una creciente convergencia entre dichas esferas. Relación que demanda un mayor uso de información y conocimientos en su vertiente tácito, codificado y codificable.

Las confusiones anteriores son materia prima para explicar el conjunto de características que componen una nueva alineación del *generall intelect* del obrero en la producción inmaterial del siglo XXI. Consideramos que este conjunto de confusiones condujeron a Hardt y Negri (2003) entre otros investigadores a señalar que la producción está sustentada en la hegemonía del trabajo inmaterial en el siglo XXI. Lo que implica -según los

---

[57] "Revolución de la Inteligencia". En la actualidad, 85% de todos los científicos que han vivido a lo largo de toda la historia están vivos y cuentan con herramientas más avanzadas y mayor creatividad. Ello ha conducido a que la tasa de cambio científico y tecnológico sea más rápida que en el pasado. Actualmente el conocimiento científico se duplica aproximadamente cada 5 años. Las áreas donde están surgiendo más innovaciones tecnológicas son energía nuclear, informática, robótica, biotecnología, telecomunicaciones y ciencias del espacio. http://es.wikipedia.org

[58] Zarifian, Philippe. El modelo de competencia y los sistemas productivos. Montevideo: Cinterfor, 1999. Papeles de la oficina técnica, OIT.

autores- un cambio pasivo, aceptable, inconmensurable frente a las nuevas formas de producción del trabajo inmaterial -al viejo estilo de Kuhn. Ésta propuesta conceptual de trabajo inmaterial no escapa a la tentación del trabajo en redes de Granovetter, ya que este tipo de trabajo inmaterial es en forma de red, premiando la colaboración y la comunicación entre agentes. Sin embargo, en la presente no abordaremos el análisis de la organización institucional de los agentes como la empresa, los empresarios, sistema educativo, asociaciones civiles, entre otros. Consideramos que las confusiones citadas invisibilizan y obscurecen el agotamiento y transformación del concepto mismo del trabajo fordista-taylorista, se invisibiliza la crisis del modelo tradicional de producción capitalista. También se ignora que la producción inmaterial no forma parte de toda la producción porque se ignoran los otros espacios laborales característicos de la producción de fin siglo: el trabajo precario, el trabajo a domicilio, el trabajo por cuenta propia, el trabajo no estructurado, etc. aspectos por demás significativos que tampoco abordaremos en la presente.

4.3.- Producción simbólica y subjetividad

Si bien es cierto el concepto de trabajo en su acepción más básica puede percibirse como "...la transformación de un objeto de trabajo como resultado de la actividad humana" (Erikson, 1990), De la Garza complementa esta definición señalando que "*ésta actividad no es aislada sino que implica cierta interacción con otros hombres, como resultado de la misma, el hombre mismo se transforma*" (De la Garza, 2003:3)[59]. Lo que condujo a esta reflexión del concepto de trabajo al autor, es la característica del proceso de trabajo del capitalismo moderno: la mayor presencia de objetos simbólicos y procesos de trabajo de creación de símbolos; por ejemplo el sector salud, turismo, publicidad virtual, industria grafica en Internet, espectáculos virtuales y no virtuales, industria del software, tele-trabajo, call-center, entre otros. El autor establece que otra forma de agrupar la producción, consistiría en intentar una clasificación del trabajo en las dimensiones "objetiva y subjetiva con un producto objetivado". Al respecto De la Garza (2007:8-10) explica que en la producción inmaterial es complejo separar las dimensiones objetivas y subjetivas, que no es posible del todo distinguir y separar el momento en que se concreta el producto y en el instante en que se consume el producto; en muchos de los casos "la objetivación se da de manera automática en otro sujeto, el cliente o usuario y no

[59] De la Garza, Enrique,2003 Problemas clásicos y actuales de la crisis del trabajo.

en un objeto separado de los dos. Es decir, se puede hablar de una objetivación en la subjetividad que, por tanto, no resulta solamente del trabajo del productor sino también del aporte del consumidor" (De la Garza, 2007:8).

Partiendo de lo anterior, consideramos que hacia fines del siglo XX cobran relevancia cierto tipo de industrias que producen bienes simbólicos que pueden objetivarse o no; por ejemplo el consumo del espectáculo y el entretenimiento, la producción de una obra de teatro o consumo de una film en el cine, el consumidor final sólo percibe subjetividades de satisfacción, descontento o simple placer. Aquí, como en otros servicios por ejemplo el asistir con el médico, en el restaurante etc., el proceso de producción no se concreta a una separación material de la producción y consumo, no se delimitan las fronteras entre proceso y consumo, entre demanda del producto y oferta del mismo; sino que es en el acto mismo del consumo donde se potencia y precisa el servicio mismo. De la Garza, señala a este tipo de producción simbólica definida como aquel servicio-producto que "*no existe separado de la propia actividad de producir y que de manera ideal comprime las fases económicas tradicionales de producción, circulación y consumo en un solo acto*". La objetivación no se separa del consumo y el producto es exclusivamente simbólico, de tal forma que su componente material resulta secundario. De esta manera "se complejizan, así, las relaciones sociales de producción al hacer intervenir a un tercer sujeto de manera inmediata en el proceso de producción junto al trabajador y el patrón, cuando es trabajo asalariado" (De la Garza mimeo, 2007:7).

También puede haber una producción simbólica objetivada que puede almacenarse y revenderse posterior a la etapa de producción, como es el desarrollo de software, el diseño de páginas web, entre otros. El proceso de producción simbólica rompe con varias rigideces del ciclo productivo de la economía industrial clásica:

- *Producción eminentemente de símbolos*, que pueden ser del tipo cognitivo, morales, estéticos, emotivos, etc.; que consiguen objetivarse o no.
- En la producción simbólica concurre una yuxtaposición del ciclo *Producción-Circulación-Consumo* comprimiéndose en un solo acto.
- Nuevas *relaciones sociales triádicas* en el piso de producción, ya que las tradicionales relaciones diádicas: trabajador-patrón, se ven alteradas (como en la producción de software a la medida) por la presencia inmediata del cliente al principio del proceso de trabajo, como hacia el final del mismo (consumidor final).

- Es complejo separar las fronteras difusas entre las *dimensiones objetiva y subjetiva* y el acto mismo de la creación del objeto. De la Garza, propone que "la objetivación se da de manera automática en otro sujeto, el cliente o usuario y no en un objeto separado de los dos." (De la Garza mimeo, 2007:4). Pero también, puede haber una producción puramente de símbolos y éstos estar objetivados o no. Por ejemplo el trabajo del software, desarrollo de paginas Web, diseño de juegos electrónicos, especulación financiera, entre otros.
- Mientras que en la manufactura tradicional el producto es un objeto externo al trabajador; el patrón lo ofrece al mercado y, dependiendo de oferta y demanda se consume y se origina una ganancia (teoría neoclásica). En la producción de bienes simbólicos, esto no ocurre, *el consumo es casi inmediato a la producción* y éste consumo depende de una serie de objetividades de quien consume el producto. Aunque en la producción de símbolos objetivados (como el software y las paginas Web, por citar algunos) las fases de producción y consumo vuelven a traslaparse.
- Si bien es cierto la actividad laboral sigue siendo de interacción a veces entre sujetos cara a cara y en otras ocasiones no es así, lo cual no necesariamente significa que es una condición en el proceso de trabajo inmaterial, en especial en el desarrollo de software que puede ser pantalla-pantalla; o fuera del proceso de trabajo (tiempos de vida).

En este punto podemos presuponer que la actual fase de producción capitalista la producción simbólica resulta relevante sea porque lo que se produce son solo símbolos (por ejemplo los programas informáticos como el software, los juegos electrónicos, etc.) o por la importancia en la producción material de la subjetividad del cliente.

Intrínsecamente al concepto de producción simbólica coexisten diferentes tipos de trabajos simbólicos:

- *Trabajo cognitivo*, se entendería por aquellas actividades en las cuales predomina el aspecto intelectual, reflexivo. Por ejemplo, la actividad académica, el periodismo, la abogacía, el diseñador gráfico, el diseñador virtual, etc.
- *Trabajo emotivo*, estaría compuesta por diversas actividades en las cuales los sentimientos, las emociones, las pasiones, etc. están presentes en mayor medida, (Bulton:2006, Bigotsky;2004 [1934], Girard:1975), por ejemplo las empleadas de

mostrador, los sobrecargos en aviación; empleados bancarios, profesionistas de radio, prensa y televisión, empleados de call center, etc.

- *Trabajo estético*, formarían parte aquel conjunto de actividades que se transmiten a través de signos abstractos como el sentimiento, las formas, expresiones o composiciones de notas musicales, libretos de film, literatura, etc. por ejemplo, los compositores e interpretes, los dramaturgos, escritores y literatos, arte fotográfico, pintura, escultura, danza, etc.
- Entre otro tipo de trabajos simbólicos.

Ahora bien no toda producción simbólica es de producción y consumo comprimidos en un solo acto. Es importante distinguir dos espacios muy generales: Trabajo simbólico subjetivado y trabajo simbólico objetivado:

- *Trabajo simbólico subjetivado*: Se comprende por un tipo de trabajo que se subjetiva en el consumidor, donde se yuxtapone y comprime la producción-circulación y consumo en un solo acto. El consumidor es el tercer actor que participa –en menor medida- en la concreción del producto y éste se consume en el acto mismo que se produce, se concreta en la subjetividad de quien le consume. Por ejemplo, los restaurantes con venta a la carta, el consumo de un concierto musical en vivo, el médico, el académico, etc. Donde el producto se concreta a través de las representaciones estéticas, sentidos del sabor, del gusto, olor, color etc.
- *Trabajo simbólico que se objetiva*: hace referencia a un tipo de trabajo simbólico que se objetiva fuera del consumidor, de tal forma que la esfera de producción-circulación-consumo no es suficiente para explicar la presencia de un tercer actor (el consumidor) al interior del proceso de trabajo. Por ejemplo el diseñador de páginas Web, el desarrollo de software, entre otros, donde si bien es cierto el cliente participa como productor e incluso como diseñador del o las características del producto éstos se objetivan fuera del cliente. Por ejemplo, el diseño de una pagina Web y la publicidad virtual (formas, esquemas, colores, etc.) ó los programas de software a la medida del cliente, si bien se entregan al cliente en un CD o USB o en cualquier otro medio electrónico o virtual, éstos programas se "virtualizan" en otros espacios que no son del cliente (por ejemplo la piratería).

4.4.- Producción simbólica y trabajo cognitivo

La cada vez mayor presencia de productos simbólicos que se objetivan independientemente del consumidor, como la industria del software, el diseño virtual, operaciones financieras (se objetivan en perdidas o ganancias), etc. nos plantea la necesidad de redefinir conceptualmente por un lado que entenderemos por producción material y que por producción simbólica y, por otro lado, precisar el concepto de producción cognitiva como sub tipo de la producción simbólica. En torno a lo primero, la disociación tradicional entre producción material y simbólica esta mas relacionada al modelo de producción fordista-taylorista. En el actual proceso capitalista cobra relevancia la producción inmaterial, de ahí la importancia de establecer una definición ampliada de producción simbólica. Si bien es cierto en el fordismo-taylorismo la separación entre concepción y ejecución, la división del proceso de trabajo y el control de la gerencia en por el *saber hacer*, definió lo material, lo físico como condición concreta de un producto separado del sujeto (Coriat, 1988) y se hizo a un lado las dimensiones subjetivas de los trabajadores, como son los intereses, sentidos, intenciones, motivaciones, deseos frente al trabajo y fuera de éste por parte del trabajador. Al respecto Marx ya había señalado que la separación entre lo material y lo simbólico no es del todo claro, argumentándolo en su clásico ejemplo de las abejas constructoras o la araña que teje complejos sistemas de redes, donde ambos procesos son superados por mucho, por el mas pésimo de los albañiles, en el simple hecho que el trabajador concibe de manera abstracta (pensamientos) la tarea que ejecutará posteriormente (Marx:1973, 130-131); esta cita de Marx, la podemos ampliar, señalando que el albañil, concibe el trabajo que realizara pero sujeto ésta abstracciones a una serie de constelaciones subjetivas: como la cultura, experiencia, habilidades, destrezas, intereses, sentido del trabajo, etc.

El esquema positivista de concebir que por un lado está la mente y por otro la materia, es criticado por Godolier (1987), para quien la separación mente-materia no existe, por el contrario señala que la cultura del trabajo es el hilo de intersección entre símbolo y materia; Para Burawoy (1989) la cultura del trabajo está compuesta por el conjunto de conductas, procedimientos formales e informales, significados y hábitos de organización en las que están inmersos las relaciones laborales; para Gintis (1983) la cultura laboral como la cultura empresarial son producto de la interacción entre sujetos inmersos en las relaciones laborales, y, éstas están constituidas tanto por elementos cognitivos (caracterizaciones de las

ocupaciones y saberes laborales) como por elementos no cognitivos (grupos de referencia normativos)[60]. Para Reygadas (2000) la dimensión simbólica y la productiva se intersecan en la cultura laboral. La separación entre concepción y ejecución de Taylor y Ford, encuentra sus limites en estos contextos, donde se debe analizar la influencia de lo simbólico en lo material y viceversa, dimensiones que no están disociadas, por el contrario se vinculan, se entrelazan y se reconfiguran mutuamente; ya que cultura incluye signos, símbolos que se refieren a conocimientos, informaciones, valoraciones, emociones, sentimientos, ilusiones, utopías, etc. y, éstas no existen externas al individuo, por el contrario son expresión de una acumulación de normas sociales (Varela:1994 pp. 3-4). En este sentido, proponemos que habría que distinguir la incidencia que tienen las características normativas, acumuladas como códigos sociales, por ejemplo los valores, las representaciones, las ideas, las normas y hábitos contextuales que estructuran la acción del individuo, y la estructura misma de la sociedad (religión, política, arte, etc.). La simbiosis entre los físico y lo simbólico, lo material e inmaterial están subjetivadas en el campo de las interacciones individuales y colectivas de los agentes, instituciones y actores que intervienen en la producción, en lo social, etc.

Para el caso de los trabajadores cognitivos del software, se trata finalmente de una producción simbólica objetivada, donde la objetivación posee varias fases o etapas, sin que ello signifique división del trabajo. Una primer fase es la resolución de problemas planteadas por el cliente, quien colabora junto con el programador o analista el diseño de los requerimientos a resolver (utilización de metodologías complejas); una segunda fase es que el programador transforma los requerimientos en algoritmos lógicos que resuelven parte del problema planteado por el cliente. Una tercera es la implementación y revisión de operabilidad del software que se objetiva, no tanto en un dispositivo portátil como CD o USB, sino que es a través del hardware del cliente donde se potencia el conjunto de dispositivos internos, invisibilizados y una vez implementado el software se amplifican sus nuevas funciones, reconfigurando el espacio donde se implemento.

Podemos establecer algunas características del trabajo cognitivo en la producción de software a la medida:

---

[60] Godolier Maurice (1989), Lo ideal y lo material, Madrid, Taurus Humanidades. Gintis. Herbert (1983) "La naturaleza del intercambio laboral y la teoría de la producción capitalista; en Tohara , L. (Comp.) El mercado de trabajo: teorías y aplicaciones. Madrid, Alianza Universidad, 1986, p. 176; Además léase Krotz, Esteban (1993). la cultura Objetivada. México, Universidad Autónoma Metropolitana – Iztapalapa:

- En el desarrollo existe un relación triádica clientes, productores y trabajadores;
- El producto cognitivo se objetiva fuera del productor y el cliente;
- Está compuestos por signos y símbolos.
- Existe una presión cognitiva en todo el proceso.
- El cliente señala una lista de requisitos –a veces directamente, en otras indirectamente- que debe cumplir el producto solicitado;
- El productor presiona cognitivamente a los trabajadores, que sean creativos y resuelvan de la mejor manera los requisitos que solicita el cliente;
- Las habilidades y destrezas, la colaboración y el juego, son mediados por conflictos y resistencias por el poder de decidir cual es el "mejor camino" en el desarrollo de los símbolos.
- La incertidumbre de cumplir o no con los requisitos esta presente.
- El cliente es un sujeto activo en el diseño y requisitos del producto.
- El producto objetivado representa una porción significativa de conocimiento, el cual se enfrenta a dos dimensiones: convertirse en conocimiento codificado (manuales de procedimiento) o bien disolverse, fragmentarse, acumularse entre los individuos como conocimiento tácito.
- El producto se puede almacenar virtualmente, copiarse, optimizarse el uso, etc.
- El costo de reproducción tiende a cero. Es mínimo comparado con el primer producto desarrollado.

4.5.- Comunidades cognitivas de trabajadores

Abordar el análisis del trabajo cognitivo considerando el corpus teórico tradicional de la manufactura es limitado porque fueron diseñadas partir de estructuras y calificaciones entendidas como conocimiento rígido, salario como instrumento mediador entre trabajo y no trabajo, producción entendida como transformación de materia a través de tecnología y fuerza de trabajo, etc. estructuras que De la Garza (2002, 2006a y 2006b) señala se han modificado, se enfrentan a limites de orden interpretativo. Por ejemplo el concepto clásico de Trabajo, este se comprime en definiciones rígidas empleadas por la economía neoclásica, que ha llevado al concepto Trabajo a un reduccionismo del concepto mismo, sea que se le considere como sinónimo entre trabajo-salario (el salario como unidad de medida de la fuerza de trabajo) o que

se considere como una acción transformadora de materias primas materiales medida en horas; dichas ideas no solo reducen el concepto trabajo a una variable dura, sino que además ignoraran o dejan de lado una serie de dimensiones subjetivas, sentidos y acciones que intervienen en el trabajo. Atendido a este señalamiento, consideramos que para el caso de la producción de software a la medida, las exigencias son de índole cognitivo y encierran en si misma una serie de retos que rebasan lo cognitivo, por ejemplo ¿como generar condiciones para la cooperación y colaboración entre trabajadores, recursos humanos y técnicos?; ¿Como generar entre los trabajadores capacidad de respuesta ágil?; ¿Como romper con las resistencias al cambio, con las rigideces culturales de los trabajadores?.

Si bien es cierto, la producción cognitiva de software se soporta en una mayor fluidez de información y conocimiento, es importante explicar el ajuste del mercado interno de los trabajadores en el proceso de desarrollo del software. Es fundamental exponer la configuración del proceso de trabajo, debemos señalar cuales con las acciones individuales y colectivas, con sus componentes objetivos y subjetivos en la organización formal e informal de los trabajadores del software. Es importante abordar cuales son las diversas estrategias flexibles, ágiles, colaborativas, de resistencia, de boicot o enfrentamiento a nivel de escritorio o pantalla del programador. Consideramos que para el caso del trabajador cognitivo del software está configurándose un tipo de trabajo cognitivo donde se realizan rutinas y tareas complejas, superiores a la división social del trabajo taylorista; donde los programadores de software no sólo modifican, sino que transforman el proceso de trabajo; la materia prima se compone de símbolos e iconos. Proceso que implica trabajo analítico, organizativo y de control; entonces ya no se trata de obreros sino de trabajadores cualificados. Estaríamos entonces ante una necesidad de redefinir los conceptos tradicionales de organización del trabajo. La explicación de conceptos y categorías laborales, implica una reestructuración cualitativa en contenidos conceptuales de quienes conforman o integran el proceso de trabajo. En otras palabras, estaríamos frente a nuevas formas de organización del trabajo donde habría que concebir un concepto ampliado, comprehensivo, que abarcara a sujetos no integrados al proceso de producción; por ejemplo los clientes, el personal de mercadotecnia, etc.

De la Garza (2006a) señala que el concepto de trabajo ha cambiado no sólo en cuanto a la disposición del sujeto obrero frente al objeto trabajo, sino como consecuencia de las transformaciones de fin de siglo en el ámbito del uso y disposición de conocimiento e

información ha llevado a una mayor presencia de productos simbólicos. Uno de los componentes de ésta producción simbólica radica en el uso intensivo del intelecto - Marx denominó *Generall Intelect* (Marx:1991)-, mas que las capacidades físicas medidas en *tiempos y movimientos*, se requiere de ingenio, habilidades, destrezas cognitivas, relacionales, emotivas, aspectos que introducen plenamente la subjetividad del trabajador frente al trabajo. Para el caso del proceso cognitivo del software, éste es un producto que se objetiva en la computadora, que se puede almacenar y objetivar fuera del consumidor (a diferencia de otros productos simbólicos que se objetivan en el consumo del mismo). Un producto que no se objetiva y no se almacena es por ejemplo la obra de teatro o el espectáculo musical (siempre y cuando no se grabe en cintas magnéticas). La característica principal del trabajo cognitivo del software supone una redefinición de la actividad laboral a nivel de interacción cotidiana entre los actores que participan, como el programador, el cliente y el gerente, interacción que no necesariamente sucede en un contexto fordista, supeditada a ordenes del capataz o cumplir tiempos y movimientos de las maquinas; por tanto cabría transitar hacia un concepto de construcción social de trabajo ampliado que se sucede al interior de comunidades simbólicas de trabajo (De la Garza, 2006a:16). Presuponemos que en las comunidades cognitivas del trabajo de Software implican una forma de organización laboral, no Taylorista. Donde en las relaciones laborales de esta forma de trabajo concurren una serie de contingencias, diferentes del proceso de la manufactura. No nos referimos sólo a la participación del cliente, sino también a la yuxtaposición de las fases del proceso de producción y la objetivación del bien desarrollado fuera del consumidor y no en el proceso mismo de producción.

Las *comunidades cognitivas de trabajadores de software* a la medida[61] es un concepto mucho más amplio que busca explicar la constelación de relaciones subjetivas que se suceden en la interacción dentro y fuera del proceso de trabajo. Busca explicar las intencionalidades del programador al documentar o no el proceso cognitivo en el que incurre para reflexionar el conjunto lógico de algoritmos que escribe en los denominados bucles de código. Así mismo, buscamos describir y complejizar las apreciaciones o posiciones presentes en las luchas por el

[61] La empresa desarrolladora de Software, es aquella que ofrece el servicio de Desarrollo de Software acorde las necesidades de la empresa, o bien ofrece un Producto desarrollado en casa que se adapta a las necesidades del cliente. La Empresa Consultora de Tecnologías de la Información, es aquella que ofrece asesoría en el tipo de TIC que debe adquirir el cliente. Y la Integradora de Soluciones Informáticas, ofrece además del asesoría en TIC, una reestructuración al interior de los proceso de organización y administración del cliente. (reflexión en base a entrevistas de Clientes como RF1, JM2, JH0).

control, resistencias y conflictos presentes y ausentes en el proceso de trabajo cognitivo del software. Las *comunidades cognitivas de trabajadores de software* están comprendidas por un conjunto de trabajadores que establecen relaciones sociales amplias, extendidas hacia el interior como el exterior del proceso de trabajo; no estructuradas o rígidas, limitadas o que rebasan al ámbito laboral; una comunidad con estructuras que le constriñen y con grados de libertad en la toma de decisiones que implica acciones subjetivas e interacciones individuales o colectivas. Interacciones que pueden ser cara a cara, pantalla a pantalla, en tiempo real o virtual, etc.

Las *comunidades cognitivas de trabajadores de software*, están constreñidas por una serie de estructuras que forman parte del proceso de trabajo:

- Existencia de varios tipos de Software: Software empotrado, Software empaquetado, Software a la medida, donde las características en el desarrollo del trabajo es variado. Pero, se mantiene la libertad del programador en la toma de decisión en la configuración de las líneas de código que dan origen a los algoritmos.
- Los algoritmos, es un conjunto de signos y caracteres que tienen su origen en los lenguajes de programación que rebasan al programador (existen mas de 2000 leguajes)
- Los algoritmos es el aspecto lógico formal que crean el software, éstos a su vez están embebidos por diferenciados grados de incertidumbre (dependerá de la complejidad del algoritmo desarrollado.
- La incertidumbre de un algoritmo esta en relación directa con condiciones objetivas y subjetivas, como son el grado de conocimiento del lenguaje de programación utilizado, experiencia medida en cantidad de programas desarrolladas en el lenguaje que se utiliza; comprensión metódica del problema planteado y "traducción" correcta de la instrucción en un algoritmo, la habilidad y destreza cognitiva en la resolución de problemas con el lenguaje de programación que se pretende utilizar. En otras palabras es el procedimiento reflexivo para escribir el requerimiento en un código lógico y coherente.
- El proceso de trabajo en el desarrollo de un programa informático no culmina en la elaboración del último código, sino que éste debe testarse, comprobarse que no haya problemas de operabilidad interna e interoperabilidad con otros programas de software. Debe implementarse en el ambiente (hardware y software) del cliente y ser compatible y claro para los usuarios finales.

- La incertidumbre en un software, puede ser interna y externa. La primera por incompatibilidad con el ambiente informático del cliente o por errores internos, la externa puede ser posible por resistencia (boicot, mal uso, borrar sintrucciones, etc.) del usuarios final aprender nuevas tareas, o bien porque se rompen arreglos tácitos al proceso de trabajo.
- En el software a la medida el cliente forma parte del proceso de trabajo.
- Una vez que es desarrollado, el costo de copiarlo tiende a cero, es equivalente al costo del CD en que se resguarde. Como bien inmaterial esta objetivado en un software que puede venderse a otros clientes.
- Una vez que el software se implemento en las instalaciones del cliente, puede actualizarse a versiones mas robustas (nuevas funciones), con un costo adicional, pero mucho menor al costo inicial.
- El software potencia, sistematiza, organiza, reestructura la organización del cliente. Dependiendo del tipo y robustez (cantidad de funciones) del software, se modificara la organización interna de la empresa contratante.

Ahora bien estas condiciones son laxas, ambiguas, interpuestas, es decir en el proceso de producción del software a la medida, el cliente participa en los requisitos o tareas que ejecutara el software, participa en las pruebas de cumplimiento de rutinas o tareas asignadas, participa en la verificación final del software, verificación que no se lleva a cabo en la empresa del proveedor, sino en las instalaciones del cliente, es decir el proceso de desarrollo de un software no culmina en la entrega del CD que contiene el software, es un compromiso del proveedor implementarlo y estabilizarlo en el ambiente informático del cliente, que haya operabilidad y este libre de conflictos de inter-operabilidad con los distintos programas del cliente, que no haya incompatibilidad entre el software instalado y el hardware del cliente, proceso que según los gerentes entrevistados puede llevar de semanas hasta meses, dependiendo de la complejidad del producto. También depende si el cliente es al mismo tiempo el usuario final del software desarrollado o bien se trata de un cliente intermediario (consultorías de nuevas tecnologías).

Los usuarios finales serian todos aquellos que utilizarían cotidianamente el software desarrollado para la empresa, junto con el cliente -que puede ser el director del área de

sistemas de la empresa que contrato el desarrollo del software, incluso una empresa intermediaria o una empresa de software que haya subcontratado el desarrollo de software- convienen un periodo de tiempo en el cual se estabiliza el sistema, se ajusta a las necesidades del cliente, -quien señalara si se cumplieron o no los requisitos previamente acordados-. Una vez que se acepto el software desarrollado y que la empresa reconoce que no existen problemas de compatibilidad, que los requerimientos del programa son los solicitados; algunos clientes optan por solicitar una póliza de garantía y mantenimiento del software, lo cual implica que el desarrollo del software se extiende mas allá del proceso de gestión de las líneas de código que componen el software. Esto es así, porque el software no se descompone, no se merma con el uso, pero se torna obsoleto, incompatible con otros sistemas informáticos conforme se transcurre el tiempo y se instalan otros programas (problemas de compatibilidad), lo cual significa que hay que actualizar o darles mantenimiento (Entrevistas).

Consideramos que no es posible intentar abordar las relaciones laborales de las *comunidades cognitivas de trabajadores de software* a partir de las categorías tradicionales del fordismo-taylorismo; poraque las c*omunidades cognitivas* se circunscriben a una forma de organización laboral diferenciada a la Taylorista. En las relaciones laborales de esta forma de trabajo no taylorista concurren una serie de contingencias, opuestas al proceso de la manufactura. No nos referimos sólo a la participación del cliente, yuxtaposición de las fases del proceso de producción, objetivación del trabajo en manos del usuario final, sino también al proceso mismo de producción. A continuación, describiremos este proceso de producción yuxtapuesto, si bien en termino formales aquí lo desagregamos por etapas, en la realidad es complejo y amorfo, las fronteras donde termina cada una de estas fases.

4.6.- Dimensiones subjetivas en el trabajo cognitivo

Intentamos situar cuales son las interacciones que se suceden al interior del proceso de desarrollo del software, las cuales no son lineales o casuísticas; como describimos en el apartado anterior, el hecho que la eficiencia del software este supeditado en parte a la implementación y aceptación técnica y social por parte de los usuarios finales, nos lleva a considerar que esta complejidad invisibiliza el proceso de trabajo de los programadores de software. El proceso de trabajo en el desarrollo de un programa informático podríamos definirlo como *"un proceso de trabajo en el cual la materia prima son diferentes tipos de*

símbolos y el resultado es un conjunto de signos que se crearon a partir de descripciones, representaciones y significados".

Las primeras dos fases de creación del software, por lo regular se consideran en una sola, dependiendo del tipo de empresa y una vez que se convinieron los requisitos entre cliente y proveedor, sobreviene la etapa del diseño, que es elaborada por el Arquitecto de Software, Ingeniero de Software, o bien por un programador que este al frente del área de diseño (dependerá del tipo de software, tipo de empresa, y organización de los equipos de trabajo), es cuando podemos decir que se inicia el proceso de trabajo del programador de software. Las dos siguientes fases es el punto en el cual, la lista de requerimientos se transforman en tareas especificas que los programadores transforman en algoritmos que conforman el módulo, dando por resultado una tarea o comando que comprende un módulo. Proceso que no es espontáneo, por el contrario, esta inmerso en una serie de acciones colectivas, individuales, virtuales, emocionales, de conocimiento formal e informal, experticia, habilidades cognitivas, destrezas o "timing cognitivo",[62] etc. Independientemente del método que se utilice en el desarrollo y presentación del diseño, el desarrollo de un sistema exige que quienes participan se comprometan con los objetivos del proyecto. En este proceso de identificación para con los tiempos y tareas del proyecto no solo deben tomarse en cuenta las habilidades profesionales y la fiabilidad cognitiva del programador, también debe poseer destrezas personales para comunicarse, tener un carácter disciplinado, y atributos importantes para el intercambio de ideas, ya que un proyecto de software requiere de un intenso intercambio de información, que en muchas ocasiones se desarrolla bajo condiciones de incertidumbre.

Ahora bien, las fases aquí señaladas son construidas con fines de explicación, porque en el proceso de trabajo, las fronteras entre una y otra se desdibujan, interactúan entre una y otra. El cliente puede intervenir o ser llamado en algunos coyunturas de la formalización o procesamiento de datos, o bien el programador negocia las instrucciones del proyecto con el diseñador o vicerversa. La toma de decisiones en la procesamiento de datos si bien es cierto es individual, no está exento de investigar fuera del proceso de trabajo, de interactuar con otros programadores frente a frente o pantalla a pantalla, en tiempo real o virtual. Sin embargo, persiste la toma de decisión individual en la estructuración o coherencia final del código. Así

[62] Entenderemos por esos momentos de "timing cognitivo" de los programadores para resolver un problema. Lo contrario sería lo que en el argot de programación se conoce como "me cicle" que hace referencia a que el programador NO resuelve un problema.

como la decisión individual de la calidad en la explicación del procedimiento reflexivo del código (comentarios al margen del código complejo, que no sea muy claro), y el desarrollo o no de documentación, eficiente o con ausencias en el procedimiento cognitivo del algoritmo (acotaciones del tipo administrativo para darle mantenimiento posterior al o los códigos - algoritmos). Sin embargo, es importante acentuar que las fases conceptualización/formalización y, por otro lado, el procesamiento de datos/Implementación no es rígida dicha separación, sin que las fronteras son difusas, no están marcadas o estructuradas, por el contrario el transito de los agentes entre éstas esferas es común, están invisibilizadas y embebidas en una *constelación de relaciones subjetivas* que son representadas por un conjunto de arreglos, consensos; conflictos, resistencias, relaciones de poder, ya sean individuales o colectivos que se inscriben, producen y reproducen formal e informalmente en las interacciones cotidianas al interior como al exterior del proceso de trabajo.

Debemos de estar pendientes, si en el sector servicios se implementan medidas de estandarización (Macdonalización, Walmartización, entre otras) y en que sentido, se aleja o acerca a los tradicionales procesos de una mayor intervención de la gerencia, ya sea a través de tecnología blanda (organización del trabajo en el piso de producción, que para el caso de los programadores de software, sería una organización a nivel de pantalla de la computadora) o tecnología dura (hardware) se esperaría mayor independencia entre las partes que integran el proceso de trabajo, una mayor separación de las características de la fuerza de trabajo individual, una mayor separación entre las fases que integran el proceso de producción, con el fin último -siguiendo a Braverman- que la descualificación se diese en la pantalla del programador de Software.

Por ejemplo, para algunos la solución de la aflicción del software se resolverá en la medida que se implementen herramientas y métodos que resulten en una mayor estandarización y control en el proceso de producción del software, es decir que la descualificación en este trabajo, posiblemente esté basada en una "administración científica de los tiempos y pensamientos" que pretende la Ingeniería del Software y las demás ingenierías que giran alrededor, como la Arquitectura del Software, Teoría Cliente-Ordenador, Teoría de la Usabilidad, Teoría de la Ergonomía del Software, etc. Sin embargo, existe un conjunto de límites a las intenciones de estandarizar la producción del software:

- Participación proactiva del cliente en distintas fases del proceso de trabajo.
- Incertidumbre en el proceso.
- Ejecución de las LDC (Líneas de código) acorde a las Destrezas, habilidades y, un conjunto de reglas informales entre los integrantes de los equipos de trabajo.
- Tomas de decisión in situ, en torno a que método es el más adecuado en la ejecución de rutinas y subrutinas de las LDC que integran un módulo.
- La toma de decisiones in situ combina un conjunto de arreglos formales e informales. Por ejemplo, el nivel de experticia de quien propone una solución, las destrezas y habilidades cognitivas de quien propone, es decir, no necesariamente es experto, pero posee un talento en el discernimiento de la solución requerida.
- En el proceso de trabajo del software la comunicación del como y el porque se desarrollo determinadas rutinas de LDC del sistema, está basado también en una doble operación: Por un lado, el aspecto formal de la documentación del algoritmo desarrollado (conocimiento codificado); sin embargo, esta descripción carece de una serie de explicaciones detalladas, reflexivas. Así, coexiste una transmisión de información de contenidos lógico-deductivos, cognitivos, los cuales no se transmiten hacia el final de la elaboración de las LDC, como es el caso en la documentación del algoritmo, sino que se va trasmitiendo en el desarrollo mismo del código. Esto ocurre entre el Programador Junior y el programador Senior. Entre el programador responsable de un modulo y el líder o administrador del proyecto.
- En la producción de software sigue habiendo una "flexibilidad cognitiva" y la formación de "arreglos sociales de participación"

Por último señalamos que existe una idea generalizada, como ya señalamos, que las nuevas tecnologías de la información y comunicación se caracterizan por una dinámica de cambio tecnológico, con ciclos de producción relativamente cortos y rápida obsolescencia, como consecuencia de la rapidez del cambio técnico. Premisa que para el caso de la industria del software consideremos inexacta por dos procesos intrínsecos a las TIC e invisibilizados en el uso cotidiano de las mismas: a).- *rigidez técnica* en la reorganización de los programas informáticos del software en ciclos cortos de cambio tecnológico; b).- *consentimiento en la socialización del aprendizaje*, es decir presencia de un ciclo relativamente largo para

familiarizarse en el uso y manejo del software. En otras palabras, cuando se implementa un nuevo software, el usuario final, se resiste en el uso y aprendizaje del nuevo software.

Los estudios del cambio tecnológico se concentraron en señalar el cambio en el hardware (mayor potencia, miniaturización, menores costos, etc.); dejando a un lado las incertidumbres en el desarrollo de nuevos programas informáticos (Caso de Microsoft con Windows Vista, Boeing con su nueva plataforma; etc.) y la resistencia cuasi-natural de los usuarios finales en el aprendizaje de nuevas rutinas. Los procesos intrínsecos señalados, como *rigidez técnica* e *consentimiento en la socialización del aprendizaje* es importante, porque es una condición estructural que constriñe el proceso de trabajo del software. Al respecto Sproull y Hofmeister (1982, citado por Novara, F.1989:70)[63] aclaran que en el desarrollo de programas de software (proyectos informáticos) existe una brecha importante entre quien diseña el software (proyectista ó analista de requerimientos) y el usuario (cliente y/o usuario final). Distancia que se explica porque el analista subvalora las actitudes conflictivas de los usuarios finales en el uso del nuevo software (resistencia, boicot, en el uso del nuevo programa informático) y las incertidumbres en el desarrollo del programa (¿Es usable? ¿Cumplió con los requerimientos iniciales?), es decir, no se planifican las pruebas en casa del usuario, las probabilidades de no cumplir con los requerimientos e incluso no se programa el fracaso del proyecto. Al final, cuando el proyecto fracasaba, el culpable era el usuario final. Es decir, el proceso de trabajo en el desarrollo del software ha venido ignorando que la implementación de un programa de software hecho a la medida de las necesidades del cliente implica un conjunto de cambios administrativos y productivos; Implicaciones que la Ingeniería del Software denomina "*orgware*", entendido como el efecto en la organización empresarial y, "*peopleware*", al efecto en la reordenación del personal y "*ergonomía del software*" al impacto cognitivo en el usuario final que es el que utiliza finalmente las nuevas aplicaciones desarrolladas. Implicaciones que aún hoy en día se debaten en torno a la creación de nuevas profesiones como la Ciencia del Software, Arquitectura del Software, Ingeniería de Requerimientos, etc. En esta misma línea, Bjorn-Andersen y otros (1979) y Mumford y

[63] Sproull, L. y Hofmeister, K. 1982, Implementation: A cognitive Approach. Working paper, Mellon University, 1982. citado en Novara, F., tecnología de la información: concepción, introducción e implantación: un enfoque multidimensional, pp. 35-84, en Castillo, J.J. (editor) 1989, La ergonomía en la introducción de nuevas tecnologías en la empresa. Colección informes numero 8, Ministerio de Trabajo y Seguridad Social, Madrid, España.

Sackman (1979), citado por Novara, (1989:70)[64], ya habían encontrado esta distancia entre la empresa desarrolladora y el usuario-cliente, principalmente en términos de confianza, porque los usuarios encontraban que el programa desarrollado no cumplía con las expectativas esperadas (incumplimiento de requerimientos o falta de comunicación de los analista con respecto a las limitaciones del programa desarrollado), las consecuencias de esta insatisfacción era el mal empleo del programa (boicot), hasta la resistencia en el uso por parte del usuario.

La participación directa del cliente en el proceso de trabajo, la importancia reflexiva y cognitiva del programador, así como la toma de decisión e intencionalidades subjetivas en la creación de los algoritmos, aunado al hecho que el software es un *conjunto de símbolos compuesto por signos*, nos lleva a considerar que los conceptos tradicionales de producción y circulación, concepción y ejecución, tiempos y movimientos, existencia del "único mejor camino", entre otras significaciones principales de la organización científica del trabajo, son insuficientes para abordar el proceso de trabajo cognitivo del software a la media. Proponemos que coexisten varias polémicas en el proceso de trabajo del software:

- Conformación de un conjunto de *constelación de relaciones subjetivas* que se construyen formal e informalmente; representan intencionalidades y conflictos que pueden ser individuales o colectivos. Estas son inherentes al proceso de trabajo: subjetividad creativa, subjetividad sígnica y subjetividad que se objetiva.
- Alto grado de integración del trabajo social en la pantalla del programador, que limita la racionalización de tareas al estilo taylorista. Aún con la fragmentación de tareas en el proceso de trabajo del software: Conceptualización (gerente-cliente); formalización (analista/arquitecto/desarrollador); Procesamiento de datos (Programadores, Tester, Documentador) e implementación (programadores, gerente, vendedor).
- *rigidez técnica* en la reorganización de los programas informáticos del software en ciclos cortos de cambio tecnológico;
- *consentimiento en la socialización del aprendizaje*, es decir presencia de un ciclo relativamente largo para familiarizarse en el uso y manejo del software. En otras

[64] Bjorn-Andersen, N. et.al. The impact of system change in organizations. Sigthoff & Noordhoof, Ámsterdam, 1978; Mumford, E. y Sackman, H. (Eds) Human choice and computers. North Holland, Ámsterdam, 1979, citado en Novara, F., Tecnología de la información: concepción, introducción e implantación: un enfoque multidimensional, pp. 35-84, en Castillo, J.J. (editor) 1989, La ergonomía en la introducción de nuevas tecnologías en la empresa. Colección informes numero 8, MTSS, Madrid, España.

palabras, cuando se implementa un nuevo software, el usuario final se opone, se resiste en el uso y aprendizaje del nuevo software.

- *Flexibilidad cognitiva:* Entendiendo como aquellas acciones objetivas y subjetivas del tipo reflexivas, de rutina o cognitivas que llevan a cabo los agentes que tomas decisiones individuales o colectivas.
- *Arreglo social de la participación*: comprendemos aquellos intercambios sociales frente a frente, pantalla a pantalla en tiempo real o virtual, de experiencias, habilidades y destrezas que permiten al programador solucionar, negociar o consensar una posible solución algorítmica a los problemas planteados por el cliente final.
- *Timmig Cognitivo:* Comprendemos una capacidad cognitiva que no esta en función de experiencias, estudios formales o intercambio de conocimiento. Es la excepcionalidad en la solución de problemas ("no todos los programadores pensamos igual") que en las hechos se traduce como una "caja negra cognitiva" donde se soluciono un problema X, pero no se conoce el proceso por escrito.

Parte I I

Gobernabilidad y estrategia nacional en la creación de capacidades locales en el sector de tecnologías de la información y comunicación (TIC)

Capitulo V

Configurando los espacios socio-técnicos de una incipiente industria de software en México

5.1.- Introducción

En los capítulos anteriores señalamos que la "*crisis del software*" esta contextualizada por espacios de aprendizaje incipientes, una ingeniería del software reciente que aún no consolida sus modelos teóricos y metodologías de trabajo; así como por un conjunto de conocimientos técnicos y de recursos humanos prácticamente recientes. Bajo esta perspectiva, en el presente apartado, daremos cuenta de las condiciones generales de las tecnologías de la información en México, es decir el acceso a Internet, ruteadores, PIB informático, etc.; con estos datos buscamos señalar las condiciones e infraestructura tecnológica en México.

Analizaremos el contexto nacional de la industria del software, las regiones ganadores, las políticas de gestión y promoción gubernamental en el desarrollo de la incipiente infraestructura local orientada a generar espacios de producción de software. Destaca en este apartado, la localización y creación tardía de capacidades locales de las empresas de software en el Valle de México.

La incorporación rezagada de las empresas de software, la falta de organización interna del sector. Las empresas de software no se instituían como un agente de cambio, no habían cuajado un conjunto de iniciativas para que se emprendiera una interacción en los espacios de producción y participar en recursos económicos de PROSFOT; no se tenía acceso a recursos humanos mediante convenios con universidades; no se habían conformado vínculos con el sistema educativo local, con empresas de mayor tamaño, etc.

Examinaremos la intervención de agentes sociales como el sistema educativo, asociaciones particulares, políticas gubernamentales, apoyos financieros, etc. en la conformación de espacios recientes de producción de software.

5.2.- TIC: un sector emergente en un contexto de división digital

El carácter reciente de la industria del software junto a la tardía incorporación de América Latina en general y México en especial, a los flujos internacionales de las tecnologías de la información y comunicación (TIC) implica una serie de desventajas que se traducen en un *incipiente sistema de interacciones y prácticas* sociales, empresariales, institucionales, individuales, agentes no gubernamentales, etc. que no han consolidado las sinergias locales ó procesos interactivos que permitan generar sistemas de innovación. Al respecto diversos investigadores como Freeman (1987); Lundvall (1992); Nelson (1993) entre otros, han señalado que: a) los sistemas nacionales de innovaciones son una especie de "nudos articulados" de empresas y sistemas productivos; b) configuraciones en redes e interacciones sistémicas que fluyen a través de las redes; y c) las conductas a nivel microeconómico que están ceñidas a relaciones sociales de las redes, reglas y restricciones políticas. Estos tres niveles distintos, ofrecen diferentes marcos de análisis para elaborar y promover estrategias de política sectorial, que den cuenta de la división digital en las TIC´s.

Cuadro 8

Gastos en la industria de TIC en México, Estados Unidos y el mundo (Millones de dólares). 1997 y 2001

	1997			2001		
Concepto/Año	México	Estados Unidos	Mundial	México	Estados Unidos	Mundial
Hardware	2,267	138,611	336,435	3,316	136,051	376,119
Software	428	54,010	104,659	597	96,556	196,237
Servicios	1,025	124,013	265,705	1,865	199,203	425,660
Total	3720	318,631	708,796	5,778	433,811	1,000,017
Software / PIB	0.11%	0.65%	0.39%	0.10%	0.94%	0.61%
TI / PIB	1.5%	5.1%	3.6%	1.4%	5.3%	4.3%
TIC / PIB	3.6%	7.7%	6.2%	3.2%	7.9%	7.6%

Fuente: Digital Planet: *The Global Information Economy*. WITSA. Febrero de 2000 y 2002.

Diversos datos dan cuenta de la acceso limitado de las TIC en México, que como usuario del flujo internacional de las tecnologías de la información y comunicación (TIC) en 2001 se sitúa en el lugar 50, pero el gasto en este rubro apenas si representó 3.2% del mercado nacional de TIC, mercado 75 veces más pequeño que el de Estados Unidos; en cuanto a la cuota de mercado de la industria del software, representó 0.10% con respecto al PIB y 3.2% con respecto al mercado de TIC (ver cuadro 8). Mientras que el mercado del software significó un crecimiento del 39.4% en 2001 con respecto a 1997; Estados Unidos creció 78.8% y 87.5% el mercado mundial de software.

El lento crecimiento en la industria del software mexicano obedece a una serie de contextos estructurales, partiendo de la incorporación tardía de las TIC en la región, pasando por los altos costos de las TIC, monopolios regionales que restringen el acceso, y una incipiente industria del software que es similar en toda la región latinoamericana. Por ejemplo, la posición de México no es tan diferente a la del resto de países en latinoamericana; según el informe de Mayer & Bunge (2004) realizado en 16 países latinoamericanos, de los cuales se eligió una muestra de 682 empresas de software y servicios relacionados[65]. Este informe nos permite observar que en América Latina la industria del software es reciente, conformándose hacia fines de los ochenta. De esta muestra, 62 (9%) empresas correspondió a México, donde observamos que 89.2% de éstas empresas (62 empresas es la muestra en México) se conformaron hace menos de 15 años; 82% son principalmente *casas de software* subsidiarias de Microsoft, IBM, Oracle; 43% de los clientes poseen menos de 49 computadoras; 90% de las empresas poseen una cartera menor a 120 clientes y, en su mayoría son profesionistas como abogados, doctores, microempresarios, etc.

En otras palabras, las empresas de software, tanto en México como en el resto de América Latina, presenta un comportamiento homogéneo (ver cuadro 9). Un porcentaje considerable de empresas son emergentes, del tipo micro y pequeñas empresas que atiende a profesionistas mas que a empresas; 86.2% de las empresas consideran sustancial desarrollar software a la medida; 82% creen que el software desarrollado debe ser implementado por ellos. Esta estructura porcentual da cuenta del carácter artesanal de las empresas de la región, el tipo de mercado individual-profesional que atienden; existencia de una casi nula especialización en el desarrollo de software, por ejemplo 81.6% de las empresas mexicanas ofrecen software empaquetado, que no fue desarrollado en casa, es decir operan como distribuidoras de los grandes corporativos, como Microsoft, Oracle, etc. Al respecto, un gerente de empresa comentaba que "el problema es que no nos hemos especializado en el tipo de software que debemos desarrollar... hacemos de todo, asesorías, implementaciones,..." (Gerente DYYA RF1).

[65] implementación, mantenimiento, actualización, comercialización de software, etc. La industria de software en México, al igual que en otros países, puede elegir entre cuatro alternativas principales de desarrollo, (lo cual no quiere decir que no hay más): Software empaquetado y Software a la medida, para el mercado local o extranjero. Heeks, 1999 y Pressman, 2002.

Cuadro 9

Indicadores de la industria del software en México y América Latina. 2003

	México	América Latina
Grado de Importancia en la empresa del desarrollo de Software (Muy importante)	86.2 %	92.8 %
Grado de importancia en la empresa en la implementación del Software que desarrolla (Muy Importante)	82.0 %	85.3 %
Porcentaje de empresas con 15 años de antigüedad (a 2003)	89.2 %	72.0%
Alianza estratégica con Microsoft	46.2 %	31.4 %
Alianza estratégica con IBM	16.9 %	12.8 %
Alianza estratégica con Oracle	18.5 %	22.0 %
Número de Clientes por cantidad de computadoras en su empresa (menos de 24 PC)	27.6 %	20.5 %
Número de clientes por cantidad de computadoras en su empresa (De 25 – 49 PC)	15.5 %	27.5 %
Número de clientes por cantidad de computadoras en su empresa (De 50 - 249 PC)	34.5 %	30.2 %
Número de clientes por cantidad de computadoras en su empresa (De 250 - 500 PC)	12.0 %	12.1 %
Clientes corporativos con facturación menor a 15 millones de dólares (hasta 120 clientes)	80.8 %	89.6 %
Clientes corporativos con facturación menor a 15 millones de dólares (de 121 a 499 clientes)	11.5%	7.3 %
Clientes individuales, profesionistas (Abogados, Doctores, etc.) (hasta 120 clientes)	90.0 %	88.8%
Clientes individuales, profesionistas (Abogados, Doctores, etc.) (mas de 121 clientes)	10.0 %	6.3%

Fuente: Elaboración en base al informe electrónico MBI - Mayer & Bunge Informática S/C Ltda. Documento en PDF: Panorama de la industria latinoamericana del Software. 2004. http://www.mbi.com.br/populacao.htm). Acceso 14.08.06

Como ya señalamos en el capitulo anterior, las comunidades empresariales productoras de software en México y en la región latina en general, no se han fortalecido, está en proceso de consolidación, las capacidades locales de infraestructura, capital humano, redes empresariales e institucionales que propicien un espacio industrial firme; por ejemplo, el 92.8% de las empresas en América Latina y 82.6% en México, declaran como muy importante desarrollar software e involucrarse en la implementación en las instalaciones del cliente, pero estas expresiones parten del hecho que las empresas poseen un perfil de *casas distribuidoras de software*, en lugar de ser empresas desarrolladoras a la medida, que son las que se involucran en la implementación. El carácter incipiente y polarizado de las empresas de software en la región, que para el caso de México, 89.2% tiene menos de 15 años de haberse constituido; las políticas gubernamentales y las condiciones de financiamiento para empresas de alto riesgo no terminan por cristalizarse; debido, entre otras cosas, que éstas empresas que manipulan símbolos no poseen, salvo escasísimas excepciones, activos fijos que las hagan sujeto de financiamiento, sus activos son talentos en capital humano y, por ende, el principal rubro de su gasto corriente está relacionado con el pago de nómina, que a su vez se ve afectado por el ciclo de los proyectos que desarrollan y por los ciclos de la economía financiera. Por último diremos que los datos de Mayer & Bunge (2004) coinciden con el

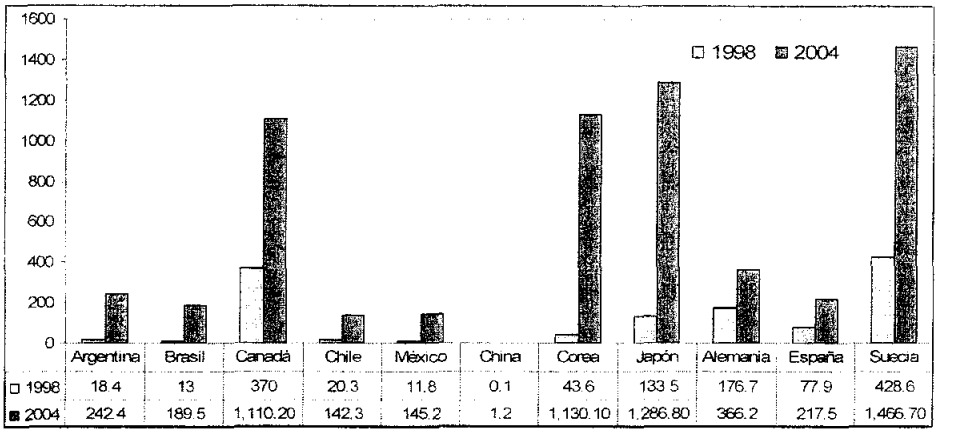
análisis que hace la Secretaría de Economía (2005)[66] de la industria del software en México, realizado en 2004, en una muestra de 800 empresas que desarrollan software, señala que 91% de los programadores de software tienen menos de 15 años de antigüedad en el puesto de trabajo y 86% culmino sus estudios profesionales en 1990.

El contar con una infraestructura tecnológica competente en TIC promueve la formación de un conjunto de agentes y actores sociales (empresas, instituciones universitarias, políticas tecnológicas, etc.) que potencian el desarrollo de procesos de generación, circulación y apropiación de información, constituyendo un sistema complejo en el cual la suficiente y correcta utilización de las TIC actúa como un dispositivo que facilita la circulación del conocimiento y el desarrollo de competencias endógenas (Lundvall: 2003; Cimoli: y Correa:2003; Casalet:2003; entre otros); en este sentido, es importante examinar el acceso digital en la región latinoamericana que, según vemos en el cuadro 4, para el caso de los servidores de Internet en México (http://www.inegi.gob.mx) en 1998 había 11.8 servidores de Internet (Hosts) por cada 10 mil habitantes, incrementándose en 2004 a 145.2; crecimiento menor al de Brasil y Argentina, pero la verdadera brecha digital es con respecto a los países desarrollados como Estados Unidos, Japón, Canadá, Suecia, etc. (véase gráfica 4 y 5).

La disposición de conectores a Internet en México (Hosts) sigue siendo limitado con respecto a países latinoamericanos en condiciones similares, ya no comparemos con respecto a Estado Unidos o Suecia; obsérvese lo reciente de la incorporación de ésta infraestructura en la región hacia fines de la década de los noventa, destacando la ampliación de la brecha digital con los países desarrollados (Canadá, Suecia, Japón Alemania, Estados Unidos, etc.) que se incrementó en 2004 por arriba de las mil servidores por cada diez mil habitantes; en cambio en Latinoamérica apenas si llegó en promedio a los 180 servidores, brecha digital que nos recuerda el fallido proceso de industrialización sustitutivo de importaciones implementado en la región para cerrar la brecha tecnológica de los sesenta y setenta (véase gráfica 4 y 5). Otra de las circunstancias estructurales que frenan la expansión del acceso a las TIC en México, además de la resistencia cultural al uso de las tecnologías (brecha generacional), es el carácter monopólico del acceso a Internet, telefonía móvil, entre otros elementos estructurales como el uso de patentes, falta de laboratorios de ciencia y tecnología, falta de innovación radical, etc.

[66] Secretaría de Economía, 2004, Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México; Universidad Autónoma Metropolitana, Unidad Xochimilco, disponible a través de suscripción en el portal www.software.com; Acceso 06/2006.

Gráfica 4
Servidores de Internet (Hosts) por cada 10 mil habitantes
Países seleccionados. 1998 y 2004



	Argentina	Brasil	Canadá	Chile	México	China	Corea	Japón	Alemania	España	Suecia
1998	18.4	13	370	20.3	11.8	0.1	43.6	133.5	176.7	77.9	428.6
2004	242.4	189.5	1,110.20	142.3	145.2	1.2	1,130.10	1,286.80	366.2	217.5	1,466.70

Fuente: Elaboración propia en base a INEGI. http://www.inegi.gob.mx. Datos de Estados Unidos son: 1,128.20 en 1998 y 6,645.20 en 2004. hosts (servidores de Internet), todos aquellos equipos conectados a la red, estos pueden ser servidores, PC's, impresoras, etc. todos ellos con dirección IP única.

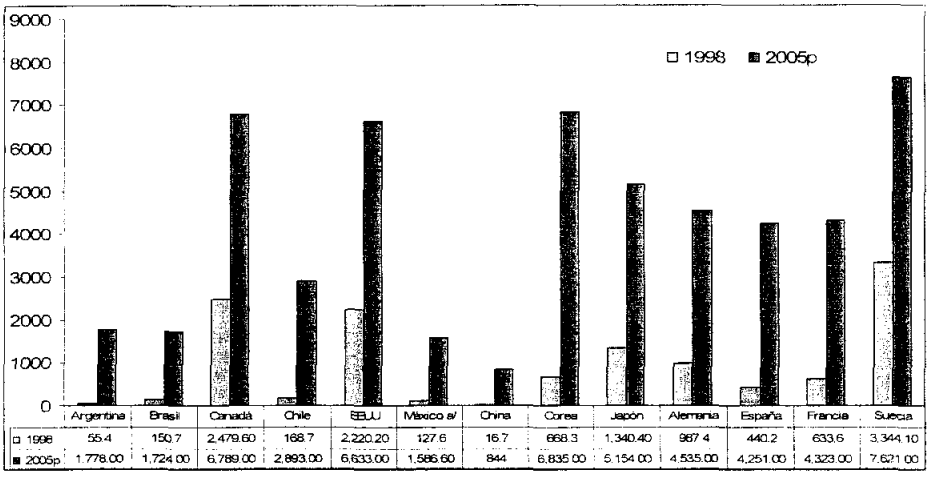
Un factor restrictivo en el desarrollo de una economía del aprendizaje, es el carácter monopólico de las TIC que promueve mercados no competitivos, por ejemplo, para el caso del acceso a Internet en México si se contrata el servicio Infinitum, con una velocidad 1Mbps, con 100 llamadas locales y 200 minutos de larga distancia nacional, tiene un costo aproximado de $549.00 pesos mensuales, es decir un costo anual de $6,588.00 -datos que fueron calculados en base a la página web de Telmex-infinitum del 31 de Enero del 2007. En cambio, en Inglaterra se obtiene el servicio de 8Mbps (ocho veces la velocidad que en México), llamadas locales y de larga nacional ilimitadas por €19.99 Euros mensuales, lo que significa €239.88 euros anuales, que de acuerdo a la página de Talk-Talk, y considerando el tipo de cambio en $21.6468 pesos por Libra de acuerdo a x-rates.com, (ambos del 31 de Enero del 2007), el costo en pesos del servicio británico es de $5,193.00 pesos. Lo cual no sólo significa un diferencial de $1,395.00 pesos en un año, sino también un servicio de mayor potencia-velocidad[67] (ver gráfica 5). El monopolio en las TIC restringe el acceso digital a poblaciones con ingresos bajos, desalienta la inversión de las empresas en nuevas tecnologías, por ejemplo mientras que en los países desarrollados en 1998 los usuarios promedio llegaban a 1,397; para 2005 se incremento a 5,823 usuarios (datos para Canadá, Estados Unidos, Corea, Alemania y

[67] Información proporcionada por Ing. David Tabeada "Acceso de Internet en México", viernes, 02 de febrero de 2007. Acceso 20 Septiembre de 2007. http://monitorpolitico.com

Francia). El acceso a Internet creció 45.2% anuales entre 1998 a 2005; mientras que los países latinoamericanos, se incremento de 126 a 1995 usuarios por cada 10 mil habitantes para los años señalados. Aún que la tasa de crecimiento anualizada es de 212% para países latinos (datos para Argentina, Brasil, Chile y México), la brecha digital es amplia (ver gráfica 5), brecha que se traduce en una lenta expansión de las condiciones necesarias para generar comunidades de investigación, redes y laboratorios que impulsen nuevos procesos de aprendizaje, innovación y generación de flujos de conocimiento.

Gráfica 5

Usuarios de Internet por países seleccionados.
Por cada 10 mil habitantes. Países seleccionados. 1998 y 2005.



	Argentina	Brasil	Canadá	Chile	EE.UU	México a/	China	Corea	Japón	Alemania	España	Francia	Suecia
1998	55.4	150.7	2,479.60	168.7	2,220.20	127.6	16.7	668.3	1,340.40	967.4	440.2	633.6	3,344.10
2005p	1,778.00	1,724.00	6,789.00	2,893.00	6,633.00	1,586.60	844	6,835.00	5,154.00	4,535.00	4,251.00	4,323.00	7,621.00

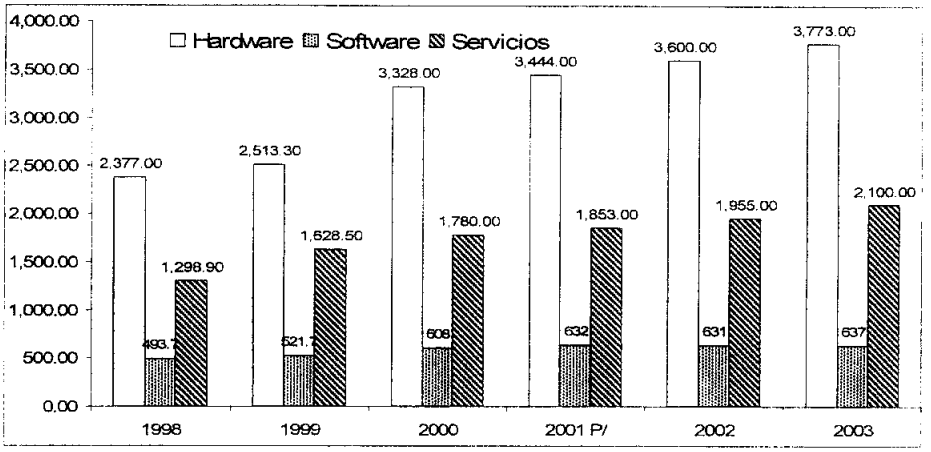
Fuente: Elaboración propia en base a INEGI. http://www.inegi.gob.mx Para México 2000: COFETEL. Dirección de Tarifas e integración Estadística, con base en información de SELECT. La cifra por cada 10 000 habitantes es estimación del INEGI, con base en datos del XII Censo General de Población y Vivienda, 2000. Para Mexico a partir de 2001: INEGI. Encuesta Nacional sobre Disponibilidad y Uso de las Tecnologías de la Información en los Hogares. Las cifras por cada 10 000 habitantes son estimaciones con datos de Conciliación Demográfica, INEGI-CONAPO, 2007.

La incorporación tardía de las TIC en Latinoamérica y especialmente en México, representa una serie de restricciones en la conformación de sistemas de innovación nacional, falta de políticas sectoriales que intensifiquen los espacios industriales y tiendan a cerrar los desequilibrios regionales y tecnológicos; por ejemplo en México, los montos de negocios en la industria de tecnologías de la información para 2003 ascienden a $6,510 millones de dólares, que representan 22% del mercado tecnológico; entre 1998-2003 una cuarta parte del mercado tecnológico estuvo concentrado en las TIC y tres cuartas partes en las telecomunicaciones. La gráfica 6 señala el crecimiento entre hardware y software en las TIC mexicanas, crecimiento que sigue el patrón mundial, destacando la dinámica del hardware, que concentró 57.2% en promedio (entre 1998-2003) contra 10.7% en software. La brecha entre hardware y software tiende a ampliarse, debido a que los ritmos de crecimiento promedio anual del hardware son

de 10.2% en el periodo señalado, contra 5.4% del software. Según datos Sallstrom y Damuth (2003), la brecha entre hardware y software podría ampliarse en los próximos años para los países que están llegando tarde a la incorporación de TIC en la base tecnológica del sistema productivo. Para el caso de México, 98% de las empresas representan el mercado objetivo de las empresas de software; el consumo de software en México es emergente y de baja sofisticación. Aunado a lo anterior, está lo reciente de esta industria, que si bien tiene un enorme potencial de crecimiento se encuentra en una etapa de despegue; donde la industria del software se caracteriza por ser joven y dominada por empresas de tamaño micro y pequeño.

Gráfica 6

Mercado mexicano de hardware, software y servicios relacionados con el desarrollo de software 1998 - 2003. Millones de dólares



Elaboración propia con base en INEGI. 2004

La incorporación tardía de México en el flujo internacional de las TIC, la división digital en el acceso a las TIC, los desequilibrios regionales la falta de competitividad del sector de las TIC; así como el reciente crecimiento del PIB informático en los últimos años en éste sector ha conducido a la intervención activa del Estado, generando una serie de políticas gubernamentales orientadas a apoyar a las instituciones de fomento productivo, asociaciones especializadas o bien, a crear leyes y decretos que regulen el mercado. El lento crecimiento de la participación de las empresas mexicanas está obstaculizada por diversos factores, desde el carácter monopólico del sector, lo reciente de la industria, falta de recursos humanos especializados, escasez de profesionistas, altos costos de mano de obra especializada, regulación poco flexible y necesidad de capacitar al personal de nuevo ingreso; además de la fortaleza del peso mexicano, elevados costos de acceso a la infraestructura tecnológica, etc. También pesa el limitado acceso al financiamiento bancario, la falta de capital semilla y de

riesgo, así como la debilidad de los programas de investigación y desarrollo. Aún con estas restricciones el PIB informático ha incrementado su peso relativo en el PIB total de 3.2% en 1994, a 4.5% en 2004 (ver cuadro 10). Crecimiento que se debe en parte a la intervención gubernamental que asimila que el problema de la creación y difusión de capacidades tecnológicas y está ampliamente relacionada con la disposición de políticas sectoriales en apoyo a las TIC (Cimoli: 2000; Katz: 2000)

Cuadro 10

Producto Interno Bruto Total e Informático
1994 - 2004. Miles de Pesos, 1993=100

Año	PIB Total (precios de mercado)	PIB Informático	Participación PIB Informático
1994	1,311,661,116	24,614,015	3.2
2001 /P	1,602,640,366	56,701,330	3.5
2002	1,613,206,366	59,647,799	3.7
2003	1,633,075,722	64,067,428	3.9
2004	1,679,150,127	75,246,937	4.5

P/ Cifras preliminares. Fuente: INEGI. Sistema de Cuentas Nacionales de México. Cuentas de Bienes y Servicios. Varios años. Tomo II.

La intervención pública a través de mecanismos novedosos hacia fines de la década de los noventa, son los fondos sectoriales implementados en América Latina que consistió en la introducción de diferentes instrumentos financieros en la realización de actividades de ciencia y tecnología con orientación hacia las pymes, que consistió en la combinación de fondos sectoriales e incentivos fiscales (Casalet: 2003; Yoguel: 2003). Destacándose dos objetivo principales: a) la creación y fortalecimiento de un mercado de servicios tecnológicos que proporciona servicios, como capacitación, asistencia técnica, consultoría especializada y formación de los recursos humanos; acorde a las necesidades de los procesos productivos: b) Fortalecer las capacidades de investigación & desarrollo de las universidades, centros de investigación independientes y de las empresas (Fisher y Klein: 2003).

5.3.- Políticas sectoriales en la industria del software

A continuación abordaremos uno de los principales programas sectoriales de apoyo a la industria del software en México (PROSOFT), el cual se caracteriza por su heterogeneidad regional, interacción con universidades, centros de investigación, asociaciones industriales, etc. Para varios autores como Capdevielle, Casalet y Cimoli (2000); Bisang y Malet (2000); Crespi y Katz (2000) entre otros; el modelo de industrialización por sustitución de importaciones (ISI) significó la participación del sector público a través de la creación de

infraestructura local orientada a las capacidades de ciencia y tecnología así como el desarrollo de recursos humanos especializados. En este sentido, se suponía que el conocimiento formaba parte de un bien público, externo a los flujos del consumo y excluido del conjunto de materias primas alineadas al proceso de producción. Se consideraba que la producción de conocimiento era suficiente para que se difundiera y circulara dentro de la economía; en otras palabras, se creía que bastaba con incentivar la oferta de conocimiento a través de políticas tecnológicas, como: a) el patrón lineal y *top down* en la difusión del conocimiento que se basaba en una serie de instrumentos políticos que asumían que las innovaciones y el conocimiento codificado se transfirieran siguiendo una trayectoria lineal y unidireccional desde las instituciones publicas, como son las universidades y centros de investigación hacia el proceso de producción. Gibbons, et.al. (1994) definen que el modo tradicional de conocimiento es aquel que va de la investigación básica a la aplicada, y de ahí al desarrollo experimental y a la innovación, es decir la generación de conocimiento se concibe bajo un concepto lineal (Yoguel: 2003); b) oferta institucional centralizada y selectiva, hace referencia al conjunto de políticas tecnológicas que trajo consigo la aventura del ISI, que significo financiamiento del sector público y regulado por el Estado en la generación de conocimiento, ello implicó que el gasto en investigación y desarrollo fuese financiado con fondos públicos y realizado en instituciones del Estado, concentrados en sectores de telecomunicaciones, transporte, energético, nuclear, aeronáutico, etc. en colaboración con el sistema de educación superior y centros de investigación que operaban con presupuesto público[68] (Katz:2000 y CEPAL:2002). Se incentivó la generación de conocimiento a través de subsidios y aranceles como mecanismos para favorecer la adaptación y transferencia al sistema productivo local del conocimiento desarrollado por empresas extranjeras. La difusión del conocimiento fue limitado a los espacios industriales donde se localizaban los conglomerados locales; c) Gestión vertical desde las organizaciones y hegemonía del sector público. Como el financiamiento provenía del Estado, las prioridades en investigación y desarrollo de la ciencia y la tecnología, estaba dictaminada por el propio Estado. Paralelamente, las estrategias propias de los investigadores basadas en el principio de libertad de cátedra como en los intereses propios que les regían resultaban en acciones determinadas en un contexto de intereses propios de los

[68] Katz, Jorge, 2000; Pasado y presente del comportamiento tecnológico de América Latina; Santiago; CEPAL; Serie desarrollo productivo, num.75; CEPAL, 2002; Fortalecimiento de los sistemas de innovación y el desarrollo tecnológico; en Globalización y desarrollo; Santiago; CEPAL.

científicos y las agendas del Estado, quedando a un lado el sector productivo y con débiles vínculos con las empresas.

Posteriormente al cambio estructural que significó la década de los ochenta, sobrevino un intenso debate en torno a la intervención del Estado en la generación de conocimiento. señalándose que el Estado, debe participar marginalmente en la economía, solo cuando haya "ruidos" en el mercado; se concluyó que una mayor desregulación y un menor intervencionismo público facilitaría la difusión de la información permitiendo hacer mas explícita la demanda de tecnología, estimulándose la difusión del conocimiento en la economía. Esta posición significó una visión horizontal en la que se planteaba que la garantía del acceso a la información permitiría solucionar los problemas relativos a la creación, adopción y difusión de la tecnología; paralelamente se limitaba la intervención del Estado, casi exclusivamente, a la creación y difusión de bines públicos (Katz; 2000, Cimoli; 2000; entre otros)[69]. Se pueden identificar las siguientes características de este enfoque basado en la demanda: a) adopción de políticas horizontales.: donde la intervención del Estado garantizaría el comportamiento eficiente de los mercados; la demanda de las empresas cumpliría un efecto de resonancia en la selección de tecnología y en la investigación, así como en la ciencia y tecnología; se consenso un mayor compromiso entre la realización de actividades innovadoras con el sector productivo y estuvieron supeditadas a la obtención de recursos externos provenientes de organismos internacionales. b) fomento a la demanda y patrón *bottom-up* de difusión del conocimiento que consistió en el diseño de instrumentos encaminados a fomentar la demanda y favorecer los flujos de información hacia el sector productivo. Por un lado, se implementaron subsidios a la demanda a través de financiar proyectos presentados por las empresas y por otro lado, con el fin de facilitar el acceso a los flujos de información al sector productivo, se dispuso de especialistas y consultores en actividades tecnológicas (Dini, Corona y Jaso; 2002). Es decir, se orientaron políticas publicas de ciencia y tecnología que expresaran las necesidades del sector productivo; así mismo se incentivó la vinculación del sector público con las universidades e instituciones públicas y privadas, supeditando el otorgamiento de recursos a la realización de proyectos tecnológicos conjuntos; igualmente se promovió la

[69] Katz, op.cit; Cimoli, Mario; 2000; *Developing innovation system*, en Cimoli, M. (ed.) Developing innovation System: México in a global context; Londres, Continnum; pp. 1-20; Cimoli, Mario; y Mariana Della Giusta; 2003. The nature the technological change and its main implications on nation system of innovation; en Jaime Aboites y Gabriela Dutrenit (coords); innovación, aprendizaje y creación de capacidades tecnológicas; México. UAM-Xochimilco/Miguel Ángel Porrua; pp. 47-102.

colaboración con las empresas y organismos que promueven actividades de ciencia y tecnología. c) introducción de mecanismos de mercado en la gestión de organizaciones; se estimuló la autonomía de gestión de las instituciones y se implementaron mecanismos de recompensa basados en resultados (Casalet: 2003; Pacheco: 2003; Yoguel: 2003); paralelamente se promovieron consultarías dedicadas a "vender" y "proporcionar" servicios técnicos, logísticos y tecnológicos en detrimento de investigaciones e investigadores menos orientados el mercado. d) bajo gasto en ciencia y tecnología; aún con nuevas estrategias de financiamiento y apoyos a la ciencia y tecnología, el gasto en investigación y desarrollo en la región latinoamericana es escasa, continua siendo financiada por el Estado, aunque las empresas son financiadas indirectamente a través de subsidios, aranceles y exención de impuestos el sector privado sólo participan marginalmente con un tercio del total invertido. Gibbons, et.al., señalan este tipo de producción de conocimiento posterior a la ISI, como un conocimiento generado en un contexto de continua negociación para su aplicación eficiente. Lo relevante es que no se generará el conocimiento a menos que los intereses de varios actores se hayan negociado e incluido; implícitamente ello significa consenso, negociación y transdisciplinariedad entre varios actores.

En este modelo la generación de conocimiento es más que el conjunto de especialistas o disciplinas que colaboran en equipos en problemas específicos, significa la integración de diversas habilidades, destrezas y capacidades en la creación de conocimiento que se sirven de flujos de información provenientes de diversas disciplinas; así como la participación de experiencias, habilidades y contextos de aprendizaje (conocimiento tácito, codificado, situado, etc.). Esta es la razón, por la cual, Gibbons et.al., señalan que se trata de un *conocimiento socialmente distribuido*, que no necesariamente es producido en un contexto normativo de reglas o procesos instituidos en universidades y colegios, en centros de investigación y consultarías, en agencias gubernamentales y laboratorios industriales, sino que se genera y produce a través de una serie de *interacciones sociales*, institucionales y gubernamentales, en comunidades enlazadas en red, o bajo sistemas de colaboración. Los grupos de investigación son menos institucionalizados, los actores que participan se reúnen en equipos de trabajo flexibles y dinámicos, temporales y especializados y en redes que se reconfiguran continuamente en comunidades y colectividades temporales, de corte interdisciplinario

integrándose científicos sociales y naturalistas, ingenieros y doctores, como necesidad no de la colectividad, sino del problema a solucionar.

La posición posterior al periodo de industrialización y sustitución de importaciones (ISI) tiene que ver con las generación de *conocimiento socialmente distribuido* a partir de la implementación de mecanismos basado en resultados, generación de conocimiento aplicado y que sea incluyente de varios actores y agentes institucionales, como es el caso de las políticas sectoriales de ciencia y tecnología mexicanas creados entre 2002 y 2003, nos referimos a la estrategia nacional de generar nichos de tecnología digital en México, de ésta se derivo una serie de políticas sectoriales que involucran sectores de telecomunicaciones, aeronáutica, electrónica, sistemas de informática y, específicamente el sector del software.

La política industrial del año 2001 y 2002 se orientó hacia programas públicos de desarrollo industrial que promovía el análisis del mercado de industrias y servicios digitales. En este sentido los grandes núcleos industriales del país como Nuevo León, Jalisco, Aguascalientes o Baja California se constituyeron como localidades o regiones atractivas para iniciar el análisis de la existencia o no de una Economía digital, ya que representan el espacio idóneo para la generación de sinergias digitales. En otras palabras, se observan ventanas de oportunidad tecnológicas para México y la posibilidad de insertarse en la cadena global de las tecnologías de la información y comunicaciones (TIC). Destacando Estados que cuentan con alto potencial para desarrollar capacidades digitales a través de mecanismos públicos con orientación a la economía digital; entre los más importantes sobresalen: Aguascalientes, Guanajuato, Jalisco, Nuevo León, Sonora, Baja California, entre otros.

Las estrategias sectoriales en la generación de economías digitales en México, el Estado participa para encauzar la expansión de un mercado basado en los flujos de información y la transformación de éste en conocimiento, de tal forma que la participación de diversos actores como el Estatal y Federal, sector privado, sector educativo y empresarial, asociaciones y centros de investigación públicos y privados acuerdan alianzas y proyectos de colaboración para desarrollar proyectos específicos en relación a la industria del Software. En distintos trabajos relacionados con la industria del software en México[70], destacan varios Estados por encima de la media nacional, sin embargo son los centros industriales por

[70] Ruiz, C. Duran 2004; Potencialidades de las entidades federativas para desarrollar núcleos de economía digital, Facultad de Economía, UNAM, México.

excelencia o por historia económica como Jalisco, Nuevo León, y Distrito Federal los que mejor infraestructura poseen para desarrollar un sector tecnológico que genere un sistema interactivo local que gestione el desarrollo de software y servicios relacionados. Las entidades cuentan con un uso intensivo en las denominadas tecnologías de la información y con una industria local de software a nivel nacional. El estudio de Ruiz, (2004) valoró el potencial de las entidades con potencialidad de infraestructura tecnológica en México a través de índices cuantitativos que relacionan desde numero de computadoras y becas de CONACYT hasta el entorno del numero de empresas con ISO 9000 y numero de alumnos en informática.

Ruiz construye un conjunto de indicadores que le permiten analizar distintos grupos de Estados que cuentan con políticas sectoriales específicas, planes y programas para desarrollar una industria del software local, de tal forma que describe: a) a*quellos que cuentan con un programa formal de desarrollo de Industria del Software* como Nuevo León, Jalisco, Guanajuato, Puebla, Morelos, Sinaloa (éstos Estados tienen un índice de economía digital superior a la media nacional)con Aguascalientes, Yucatán, Hidalgo y Campeche (poseen un índice de economía digital menor a la media nacional). b) Estados que *no cuentan con un programa formal de desarrollo de Software, pero han manifestado su intención de desarrollar un programa formal,* por ejemplo Distrito Federal, Baja California, Veracruz (Estados que tienen un índice de economía digital superior a la media nacional) y San Luís Potosí, Baja California Sur, y Durango (Estados que tienen una economía digital menor a la media nacional). c) Estados que no *han manifestado su intención en desarrollar una industria del Software*, como son México, Coahuila, Sonora, Tamaulipas, Chihuahua, Michoacán, Colima, Guerrero, Tlaxcala, Chiapas, Quintana Roo, tabasco, Nayarit y Zacatecas. Para continuar con el análisis sectorial de la generación de capacidades en la industria del software, sólo retomamos seis Estados que concentran 68.7% de la cuota de mercado en la industria del Software en 2003. De los cuales destaca el DF que concentra 47.3%, seguido por Nuevo León con 14.0% y el Estado de México con 9.1%; Jalisco sólo representa 4.1%, Aguascalientes 0.48% y Morelos el 0.5% de un mercado que en 2001 significó $2,428.4 millones de dólares. Ruiz (2004) señala que uno de los elementos centrales en el desarrollo de la industria del Software es la demanda diferenciada por cada entidad federativa. Uno de los factores claves para potenciar la demanda es la posición tecnológica local de cada entidad, la estructura local de los sistemas sociales entre agentes locales, sector productivo, universidades, empresas

líderes, así como el conjunto de relaciones de confianza y colaboración que se hayan conformado en las redes, comunidades y colectividades que existan en la localidad. En el cuadro 11 observamos (Ruiz: 2004) algunos índices de economía digital que posicionan a las entidades locales y permiten evaluar la potencialidad de las entidades en los nichos de economía digital.

Cuadro 11

Índices de capacidades digitales para el desarrollo de la industria del Software. Por estados seleccionados de México 2002

	Programa formal en la Industria del Software			Promedio	No programa	
Entidad / Índices	Jalisco	Nuevo León	Aguascalientes		BCN	DF
Cuota de mercado*	4.1%	14.0%	0.48%	2,428.4	2.3%	47.3%
Capital Humano	5.23	9.68	7.16	5.0	8.77	10
Aprendizaje e Innovación	7.78	7.74	3.44	4.20	6.45	10
Empresarialidad	8.93	8.37	2.56	4.45	6.58	10
Entorno Favorable	4.42	7.80	7.41	5.02	6.04	8.57
Infraestructura Digital	8.40	7.49	3.07	4.69	6.99	10
Valor del mercado	9.03	9.68	2.9	5.00	8.39	10
Capacidades locales	7.30	8.46	4.42	4.73	7.20	9.76

Fuente: Elaboración propia a partir de Ruiz, C. 2002. Programa para el desarrollo de la industria del software (PROSOFT) 2002 y 2003. Secretaria de Economía; reporte de potencialidades de las entidades federativas para desarrollar núcleos de economía digital. Clemente Ruiz Duran 2002; reporte electrónico.
* Cuota de mercado en la industria de $2,428.4 millones de dólares en 2001.

Como vemos en el cuadro 11 destaca el posicionamiento del Distrito Federal, seguido por los Estados de Jalisco y Nuevo León. El caso del Distrito Federal lo abordaremos hacia el final de este apartado por ser paradigmático su caso porque hasta Junio de 2004 no contaba con un plan formal para desarrollar una industria de Software, a pesar que concentraba poco mas del 40% de las empresas desarrolladoras. Los indicadores del cuadro 11 son una guía cualitativa del las condiciones o esfuerzos de los actores locales para generar infraestructura tecnológica, académica, humana, técnica, industrial, entre otros componentes que ya hemos señalado. Sin embargo, estamos de acuerdo que las relaciones de confianza y cooperación entre los agentes no suceden por "generación espontánea", sino que deben coexistir políticas selectivas de impulso a las condiciones sociales, técnicas, de recursos humanos, educativos, etc., que potencien, gestionen desde abajo las capacidades endógenas necesarias para generare un sistema epistémico sólido, con identidad local. Las entidades citadas en el cuadro 11 establecen diferentes estrategias con la intención de desarrollar una industria de Software en

sus localidades; propuestas que son presentadas por diversos actores locales, como universidades, empresas, gobierno, asociaciones, etc.

En 2004 el Estado a través de la Secretaria de Economía promovió una política sectorial especifica de apoyo a los sectores productivos del software, dándosele prioridad a aquellos Estados que contaran con programas formales en el impulso a la generación de sistemas productivos de software, como es el caso de Nuevo León y Jalisco, seguidos por Baja California y Aguascalientes. Los apoyos financieros contemplan diferentes aspectos como la infraestructura física, académica, capital humano, aprendizaje e innovación, nivel de empresarialidad, valor del mercado de TIC, entre otras variables que generen sinergias orientadas a favorecer el desarrollo de software. Empero, el financiamiento esta condicionado a la participación de diversos agentes además de la inversión Federal como las empresas entre otros actores institucionales además de Universidades, centros de investigación públicos o privados, entre otros. Favoreciéndose el fortalecimiento de sistemas de interacciones socio-técnicas entre asociaciones, gobierno, empresas, instituciones educativas y organismo locales. Como por ejemplo las alianzas como el *consejo ciudadano consultivo para el desarrollo de la industria de software* en Nuevo León, integrado en 2002 por el gobierno del Estado, el secretario de desarrollo económico y el director general del programa *Monterrey Ciudad Internacional del Conocimiento*; además del delegado de la Secretaría de Economía, NAFINSA e INEGI; así como instituciones académicas como UANL, ITESM, TEC-MILENIUM, UDEM, UR; Asociaciones civiles de Software como CANIETI, AETI, ANADIC; empresas de software, como Softek, Neoris, Towa, Quality, Internacional de sistemas, entre otras. Esta alianza ciudadana se basa en un "proyecto sistémico" que genera sinergias y niveles de confianza entre los actores locales. PROSOFT en colaboración con los otros agentes entre 2004-2005 acumuló una inversión cercana a los mil millones de pesos con el objetivo de desarrollar centros de desarrollo de Software. Además de Nuevo León, existen sistemas sociales sistémicos como el de Jalisco, Aguascalientes, Baja California, entre otros.

A continuación abordaremos el programa nacional de la industria del software (PROSOFT), instrumento que se forma bajo los preceptos del modelo de subsidio a la demanda que provee de importantes fondos públicos a los espacios industriales del país, que como bien señala Casalet (2000 y 2003) la intervención del Estado hacia los noventa se orienta a la creación de un entorno de eficiencia para el sector productivo, ya que la

"liberalización" del mercado implicó una mayor vulnerabilidad de las empresas, especialmente las pymes. El Estado "vigila" y propicia nuevas relaciones de colaboración que configuran nuevas competencias y capacidades locales que contribuyen al sistema epistémico, de aprendizaje y conocimiento.

5.4.- Programa sectorial específico: PROSOFT

Intentaremos puntualizar la estrategia macroeconómica que ha implementado México en la Industria del Software a través del Programa nacional de la industria del Software (PROSOFT) [71] y, la influencia reciente de ésta política sectorial en la industria. Este programa denominado PROSOFT es de reciente creación; primero se implementó como un programa de alcance regional en mayo de 2004, donde participaban mancomunadamente Estado-Federación. Para Junio de 2005, se extiende al resto de del país y en Diciembre de 2006 es significativa la intervención financiera del sistema universitario público y privado, cámaras industriales, centros de investigación y desarrollo de tecnología y, empresas del sector. Un aspecto importante de PROSOFT es que articula significativamente la participación de industrias locales de software, cámaras de la industria e instituciones académicas locales e internacionales dedicadas al desarrollo de software. PROSOFT apuesta a que México puede integrarse a flujos internacionales del mercado mundial de las TIC[72]. De acuerdo con PROSOFT (2004), México cuenta con una posición favorable para convertirse en un competidor de talla mundial, gracias a su ubicación geográfica, perfil demográfico y desarrollo tecnológico. Los nichos de software, que PROSOFT identifica a nivel mundial, son servicios profesionales, que representa un mercado mundial de $340.2 mmd en 2003; y productos de software con un valor de $56.6 mmd (PROSOFOT: 2004:276).

No obstante el potencial de los nichos tecnológicos que PROSOFT señala, la industria del software en México es incipiente, como ya dimos cuenta párrafos arriba, en el periodo comprendido entre 1998 y 2003 la industria del Software, -con respecto al total del sector tecnológico (TIC mas telecomunicaciones)- no supera 2.2% en promedio y, comparado con el mercado de las TIC (hardware, software y servicios), la cuota de mercado es de 10.7% en promedio entre 1998-2003; ésta industria del software requiere una seria de generación de

[71] Secretaría de Economía, disponible en: http://www.software.net.mx Acceso en Febrero de 2005.
[72] El mercado mundial de las tecnologías de la información y comunicación para 2002, representó $2.18 miles de millones de dólares, que se distribuyen en cuatro rubros generales: telecomunicaciones 56.4%; Servicios de TI 24.6%; hardware 15.5% y Software 3.5% (Gartner:2003).

condiciones estructurales, económicas, sociales, culturales, acumulación de conocimientos, que no se generan por decreto o por "cañonazos" de dinero. La industria está ligada a la existencia de variables cualitativas, no sólo a programas públicos de desarrollo industrial y existencia de un mercado local de industrias y servicios que consuman nuevas aplicaciones; sino a la existencia de condiciones de acumulación de conocimientos en aquellos agentes que participan en esta industria; llámense asociaciones, academia, empresas e ingenieros de software. En otras palabras se requiere de un contexto dinámico, pro-activo en términos tecnológicos modernos, que potencien la generación y adquisición de capacidades tecnológicas, circunstancias que por lo regular sólo favorece a los agentes y sistemas socialmente arraigados, como los de Jalisco, Nuevo León entre otros, creando fuertes desequilibrios regionales; por ejemplo, los tradicionales centros industriales de Jalisco y Nuevo León concentran en 2005 el 40.6% del total de PROSOFT, por tanto, podríamos estar en presencia de la ampliación de la brecha digital que da cuenta Ruiz (2004) a partir de favorecer sistemas de interacción tecnológico ya sólidos (ver cuadro 12). En los siguientes apartados abordaremos los distintos agentes pro-activos que son favorecidos por mayores incentivos para acceder a los mecanismos de soporte financiero.

Cuadro 12

Inversión total en proyectos aprobados en la Industria del Software en México. Estados Seleccionados. 2004 y 2005

	Proyectos 2004	Monto 2004	Proyectos 2005	Monto 2005	Crecimiento 2005-2004
Total	**68**	**249,523,273**	**181**	**749,520,322**	200%
Jalisco	13	46,534,240	14	204,195,833	339%
Nuevo León	17	52,010,964	32	100,317,660	93%
Aguascalientes	3	8,103,430	15	24,326,723	200%
BCN	7	44,194,116	9	33,083,391	-25%
ZMDF	--	--	--	--	--
Asociaciones*	6	70,701,523	25	179,577,878	154%
Resto	22	27,979,000	86	208,018,836	643%

*Cuando se señale Asociaciones, se entenderán por aquellos organismos de promoción a la industria del Software reconocidos por PROSOFT, como: Asociación Mexicana de la Industria de Tecnologías de Información, A.C. (AMITI); Asociación Mexicana para la Calidad en Ingeniería de Software (AMCIS); Asociación Mexicana Empresarial de Software Libre, A.C. (AMESOL); Cámara Nacional de la Industria Electrónica, Telecomunicaciones e Informática (CANIETI). Fuente: Elaborado en base al Informe preeliminar de PROSOFT 2005. Realizado por la UAM a encargo de la Secretaría de economía.
Nota: El monto de 2004, contiene la partida de NAFIN por $64,389,196.00

5.4.1.- Interacciones sociales entre agentes institucionales

El PROSOFT desde su creación en 2004, favoreció la participación de la academia, la constitución de asociaciones, la coparticipación del Estado y, el financiamiento estaba condicionado a que el sector productivo no sólo participara, sino que también formara parte de los inversionistas. De esta manera los actores locales como el Estado; el sector privado, el educativo y empresarial; así como las asociaciones industriales, establecen alianzas y proyectos específicos en relación a la industria del software. Para los primeros dos años de operación del PROSFOT (2004-2005) la consolidación del *sistema social interactivo* entre agentes institucionales, según damos cuenta en el cuadro 13, la participación de los organismos institucionales (CANIETI, AMITI, AMCIS, entre otros) tuvo un crecimiento importante, en 2005 superior a los 179 millones, contra los 70 millones de 2004, incremento que significó un crecimiento del 174%.

Como daremos cuenta en el siguiente cuadro 13, otros agentes institucionales aumentaron su colaboración en la industria del software, y sobre todo la cada vez mayor presencia del sector privado. Del total invertido por las asociaciones en 2005, 48.2% es aportado por el sector privado; en este mismo año los centros industriales por excelencia, como Nuevo León y Jalisco, concentran 44.7% del monto total invertido y representan una cuarta parte de los proyectos financiados, es decir en promedio cada proyecto de software significa 6.6 millones de pesos, en contraparte, para el resto de los 135 proyectos, el financiamiento promedio es de 3.3 millones de pesos y, una de las economías digitales mas competitivas (según Ruiz: 2004) que es la zona del valle de México, hasta septiembre de 2006, no figuraba dentro de los estados que recibían financiamiento del Estado vía PROSOFT (Ver cuadro 13). 2005, significó una nueva distribución del financiamiento de PROSOFT, la participación relativa de cada uno de los agentes varió significativamente, destacando lo siguiente:

- El peso relativo de PROSOFT como organismo promotor (inversión) representó el 56% en 2004; para 2005 disminuye al 26%. Por el contrario, el sector privado, incrementa su participación porcentual del 24% al 48% para los años correspondientes.
- El incremento del presupuesto de PROSOFTO se incrementó de 249.5 millones de pesos en 2004 a 749.5 millones en 2005, crecimiento del 200% (Ver cuadro 13). De este incremento, las asociaciones fueron mas beneficiadas, incrementando sus recursos de 70.7

millones en 2004 (no se incluyen los recursos de NAFIN de 2004) a la cantidad de 179.5 millones de pesos en 2005. Inversión en la que sobresale el sector privado, que aportó 86.6 millones vía asociaciones, porque el total del sector privado, representa el 89% mas que la inversión de PROSOFT.

- La participación del sector educativo en 2005 es importante, incrementando su participación en poco mas del 280%, al situarse la inversión general del sector educativo en 13.2 millones en 2005, contra los 3.4 millones de 2004.

Cuadro 13

Distribución de la Inversión por organismo promotor en la Industria del Software en México. Estados Seleccionados. 2004 y 2005

	PROSOFT 2004	PROSOFT 2005	Estados 2004	Estados 2005	Sector Académico 2004	Sector Académico 2005	Sector Privado 2004	Sector Privado 2005
Total	**139,700,000**	**191,612,134**	**42,546,659**	**107,339,099**	**3,463,000**	**13,271,763**	**60,360,484**	**363,202,498**
Jalisco	19,734,284	38,008,411	8,800,000	17,711,835	840,000	630,000	17,159,956	142,630,587
Nuevo León	18,382,993	27,198,000	13,448,991	20,000,000	986,000	nd	19,142,980	50,316,460
Aguascalientes	3,517,000	6,450,118	2,160,000	3,133,698	nd	nd	1,598,300	14,562,906
BCN	18,838,948	9,470,018	11,295,168	6,831,008	177,000	427,860	13,808,000	14,026,708
ZMDF	--	--	--	--	--	--	--	--
Asociaciones*	67,518,775	45,071,001	nd	nd	nd	4,385,294	3,182,748	86,647,753
Resto	11,708,000	65,414,585	6,842,500	59,662,557	1,460,000	7,828,609	5,468,500	55,018,084

Fuente: Elaborado en base al Informe preeliminar de PROSOFT 2005. Realizado por la UAM a encargo de la Secretaría de economía
Nota: El monto de 2004, NO contiene aquí la partida de NAFIN por $64,389,196. nd= No hay datos.

La participación relativa de PROSOFT en la inversión total de la industria del software sólo se incremento 37% en 2005 con respecto a 2004, de hecho es uno de los porcentajes más bajos con respecto a la participación de otros agentes institucionales, como los Estados y el sector educativo. La inversión total de los agentes institucionales, significa un financiamiento importante, ya que los Estados, sector educativo mas PROSOFT, acumulan una inversión en 2005 de 312.2 millones, cantidad equivalente al 86% de lo invertido por el sector privado; es decir, que en 2005 el sector privado superó al invertido en PROSOFT, sin embargo aún habría que calcular el apoyo vía impuestos a esta inversión del sector privado (ver cuadro 13).

Abordemos brevemente la participación de los organismos promotores que integran a las asociaciones. En el destino de los recursos de PROSOFT hacia las instituciones que forman parte de las asociaciones civiles reconocidas por la Secretaría de Economía, destaca la Asociación Mexicana de la Industria de Tecnologías de Información, A.C. (AMITI); Asociación Mexicana para la Calidad en Ingeniería de Software (AMCIS); Asociación Mexicana Empresarial de Software Libre, A.C. (AMESOL); Cámara Nacional de la Industria

Electrónica, Telecomunicaciones e Informática (CANIETI) (ver cuadro 16). Las asociaciones tienen una participación significativa en 2005, año en el que disponen de 179 millones, cantidad que representa 72% del monto total de PROSOFT en 2004. Lo destacable, es que de este porcentaje, 48.3% proviene de empresas privadas (86.6 millones de pesos). Las asociaciones representan un sistema de interacciones donde se concilian intereses y objetivos de las empresas que ahí participan.

Por ejemplo para 2005, la Asociación Mexicana de la Industria de Tecnologías de Información (AMITI) es una de las primeras organizaciones nacionales que concentra más de 200 empresas de Tecnología de la Información; AMITI se convierte en una asociación de confianza, genera certidumbre en las intenciones que propone, se observa un salto cuantitativo en la cantidad de proyectos que presenta en 2005, y la cantidad de inversión de acumula en 2005; la Cámara Nacional de la Industria Electrónica, Telecomunicaciones e Informática (CANIETI), es la segunda organización que agrupa a poco más de 300 empresas de tecnologías de la información en México, nacionales y extranjeras que en 2005 invirtió 60.3 millones de pesos en el desarrollo de la industria del Software (ver cuadro 14).

Cuadro 14
Inversión total en la Industria del Software
en México. Por Asociación 2004 y 2005

	Proyectos 2004	Monto 2004	Proyectos 2005	Monto 2005
Total	**6**	**70,701,523**	**25**	**179,577,878**
AMCIS	1	231,600	2	5,687,600
AMITI	3	6,080,727	10	**113,548,055**
NAFIN	2	64,389,196	--	--
CANIETI	--	--	13	60,342,223

AMSCIS: asociación Mexicana del Software.
AMITI: Asociación Mexicana de Tecnologías de la Información
CANIETI: Cámara Nacional de la Industria Electrónica y Tecnologías de las Información
Fuente: Elaborado en base al Informe preeliminar de PROSOFT 2005.
Realizado por la UAM a encargo de la Secretaría de economía

5.4.2.- Infraestructura y capital humano

La industria del Software se distingue del resto de la industria por representar una alta tasa de innovación en procesos, motivo por el cual no sorprende que más del 50% de los proyectos y del monto destinado entre 2004 y 2005 represente mas de 137 millones de pesos destinados a la categoría de gestión e innovación tecnológica. Siguiendo en importancia, la categoría de fortalecimiento de las capacidades locales de promoción y apoyo financiero (ver cuadro 15)

Cuadro 15

Inversión total en la Industria del Software por categorías. México 2004 y 2005. Precios corrientes

	Monto 2004	Monto 2005
Total	139,700,000	191,804,908
Desarrollo de gestión e innovación tecnológica	27,356,402	109,885,701
Habilitación e infraestructura	18,470,328	57,011,826
Capacidades regionales, financieras y promoción	29,484,093	24,907,380
Nacional Financiera	64,389,177	nd

Fuente: Construido en base a Informe preeliminar de PROSOFT 2005

Si bien es cierto que a la categoría de gestión e innovación tecnológica se destinaron poco más de 137 millones en forma acumulada por ambos años, es en 2005 el año en que se acentúa el financiamiento en este rubro que crece en más de tres veces, destacando la categoría de Innovación y desarrollo tecnológico con una concentración acumulada en ambos años superior a 59 millones, seguido por el monto destinado a la formación de capital humano con poco más de 40 millones (Ver cuadro 16).

Cuadro 16

Inversión total en Desarrollo de gestión e innovación tecnológica en la industria del Software en México 2004 y 2005. Precios corrientes.

	Monto 2004	Monto 2005
Total	27,356,402	109,885,701
Formación y Desarrollo de Capital humano	5,268,330	35,561,702
Calidad y Capacidad de Proceso	7,827,468	15,045,060
Innovación y desarrollo Tecnológico	14,260,604	44,886,840
masa crítica del sector	0.00	14,392,098.00

Fuente: Informe preeliminar de PROSOFT 2005. Realizado por la UAM.
Integra las categorías 1,2, 3 y 9 del Informe preeliminar PROSOFT 2005

5.4.3.- Acciones a nivel local

Es un reclamo constante por parte de las empresas que integran el sector del software el no contar con los recursos económicos suficientes para promover y comercializar el software manufacturado o bien, que las empresas de software no participan en exposiciones o ferias internacionales donde ofrecer el producto desarrollado; así como la falta de inversión para ofrecer seminarios de organización administrativa, cursos de certificaciones internacionales, entre otro tipo de actividades que fortalezcan las capacidades locales. Las capacidades regionales son las que en el mediano plazo nos permitirá evaluar el impacto de PROSOFT en la localidad. Aún así es la categoría menos favorecida por la inversión. (El dato de 54.3

millones, lo obtenemos de restarle al total del rubor Capacidades regionales, la cantidad destinada a NAFINSA, por desconocer si el destino de estos recursos se aplican al sector). Este análisis visto para los principales estados que hemos seleccionado el comportamiento es variado y aleccionador. Entre los estados ganadores, sobresale Jalisco y Nuevo León, ambos estados centralizaron 40.3% del monto acumulado total de la inversión que se dirigió a la fortalecer la industria del Software mexicana, porcentaje que asciende a 402 millones de pesos repartidos en 76 proyectos entre ambos Estados.

Cuadro 17

Inversión en diferentes capacidades regionales en la industria del Software en México 2004 y 2005. Precios corrientes.

	Monto 2004	**Monto 2005**
Total *	29,484,093	24,907,380
Desarrollo de Capacidades financieras, empresariales y de estrategia	2,396,500	6,893,107
Fortalecimiento de capacidades regionales y empresariales	22,262,964	6,025,997
Promoción y comercialización	4,824,629	11,988,276
Nacional Financiera	64,389,177	nd

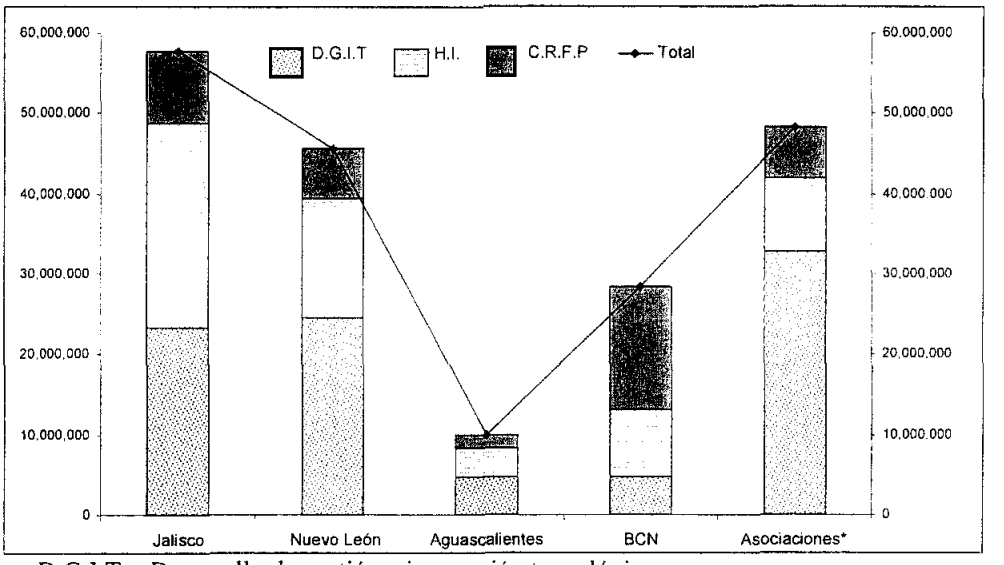
Fuente: Informe preeliminar de PROSOFT 2005. Realizado por la UAM.
Integra las categorías 4,6, 7 y 8 del Informe preeliminar PROSOFT 2005.
* NO incluye la inversión de NAFINSA: $ 64,389,177.00

Una de las preguntas claves en este tipo de inversiones dirigidas a la demanda, es que si los fondos sectoriales y los mecanismos alternos de subsidio e incentivos fiscales resultan o no en acciones sub-optimas de política tecnológica, con menores potencialidades de impacto o con escasos efectos multiplicadores en la generación de sinergias locales; es decir, que los agentes pro activos en la industria que se intenta promover, están concentrados en los espacios industriales clásicos, y por tanto ya cuentan con dispositivos sociales, redes o comunidades sistémicas que estimulan la generación de una ambiente de flujo de información y conocimiento, por tanto se puede incrementar la división digital en el país. Aunque la producción de conocimiento, en nuestro caso de software, es un resultado complejo que se genera en redes de empresas e instituciones y, por tanto, las dificultades como lo que hemos llamado la "*aflicción del software*", puede tener diversos orígenes, en esta investigación nos concentraremos no en la red sino en la base de esta que es el programador, sin desconocer que una investigación más completa tendría que abordar a agentes meso y macro, diferencias entre equipos de trabajo de diseño, y el cliente que si serán analizados. En este tenor, veremos

en el siguiente capítulo cual es el perfil de la mano de obra de programación en esta industria del software.

Gráfica 7

Inversión acumulada por principales categorías en estados seleccionados de la industria del software en México. 2004-2005 Pesos corrientes



D.G.I.T.= Desarrollo de gestión e innovación tecnológica
H.I. = Habilitación e infraestructura
C.R.F.P. = Capacidades regionales, financieras y promoción
Fuente: Informe preeliminar de PROSOFT 2005. Realizado por la UAM

Capitulo VI

Perfil socio-profesional de un trabajo en construcción.

El trabajador cognitivo, entre profesión, ocupación y empleo

6.1.- Introducción

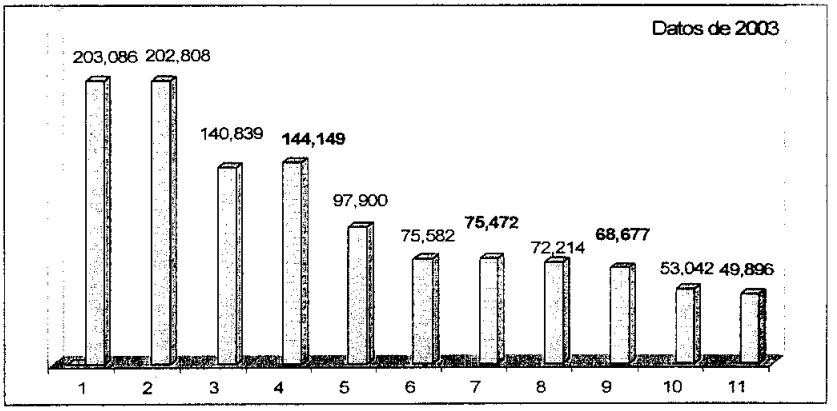
La información que se presenta a continuación tiene por objetivo conocer cuales son las condiciones de la fuerza de trabajo profesional del programador de Software. La profesión de programador de software no está definido en la amplia matricula de las profesiones de educación superior, es decir la oferta de profesiones en México no contemplan una carrera profesional que otorgue el Titulo de Ingeniería del Software al menos hasta 2003 (ANUIES, 2003), aunque es amplio el abanico de profesiones vinculadas con esta actividad, como veremos más adelante. También hacia el interior del mercado de trabajo del software es heterogénea la cantidad de descripciones de las ocupaciones relacionadas con la programación del software que va del diseñador, analista, programador, informático, etc.; también el conjunto de conocimientos, habilidades y destrezas requeridas para el empleo de programador son variados. En el primer apartado se presentan estadísticas generales de ingeniería en computación e informática en México, como un gran agregado de las diversas licenciaturas en ingenierías. Estos datos se construyeron a partir de la Encuesta Nacional de Ocupación y Empleo de la Secretaria del Trabajo y Previsión Social (STPS-ENOE) disponible en el observatorio laboral (http://www.observatoriolaboral.gob.mx/, acceso 18.09.07), así como estadísticas del Instituto Nacional de Estadística, Geografía e Informática (INEGI). Además la información se complementa con los resultados finales de la encuesta nacional denominada "Estudio para determinar la cantidad y calidad de los recursos humanos necesarios para el desarrollo de la industria del Software, 2004" realizado en México por la Universidad Autónoma Metropolitana (UAM), unidad Xochimilco, por encargo de PROSOFT -Secretaría de Economía. El segundo apartado, es en torno al contexto profesional en el cual está inmerso el programador de Software. En este apartado nos concentramos en la zona metropolitana del Distrito Federal. Los datos cuantitativos los obtuvimos de una encuesta que realizó la Universidad Autónoma Metropolitana (UAM) denominada "Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información en la Zona Metropolitana de Monterrey y Zona Metropolitana del Distrito Federal, 2004" por encargo de PROSOFT - Secretaría de Economía.

6.2.- Construcción social de una profesión reciente

La profesión de la computación e informática en México como tal no cuenta con más de tres décadas de estar presente entre los registros de la educación superior en México (ANUIES), el número de egresados no es muy significativo con respecto a otras carreras. La cantidad de egresados en el rubro general de tecnologías de la información y comunicación en México (datos de INEGI) se consolidó entre 1995 y 2001 con 10 mil y 15 mil egresados respectivamente. Estos datos pueden implicar una agregación inadecuada, porque este concepto de tecnologías de la información y comunicación se ha posicionado socialmente como aquel sector donde convergen la informática y la electrónica (Mertens: 1994; Castells: 1999), lo que podría significar que se agreguen profesiones de la electrónica que no necesariamente comprende la computación e informática. Al respecto, diferentes instituciones como INEGI, ANUIES, ENOE-STPS, incluyen distintas licenciaturas e ingenierías bajo el concepto de ingeniería de computación e informática (ver gráfica 8).

Gráfica 8

Matrícula de nivel licenciatura en tecnología de información y comunicación por carreras, 2001/2002



1) Lic. Derecho, 2) Lic. Administración, 3) Contador Público, 4) Tecnologías de la Información y Comunicación, 5) Ing. Industrial, 6) Medicina,7) Lic. Informática, 8) Lic. Psicología, 9) Ing. en sistemas computacionales, 10) Arquitecto, 11) Ing. Electrónica.
Fuente: Elaboración propia en base a INEGI, http://www.inegi.gob.mx Acceso Julio de 2007

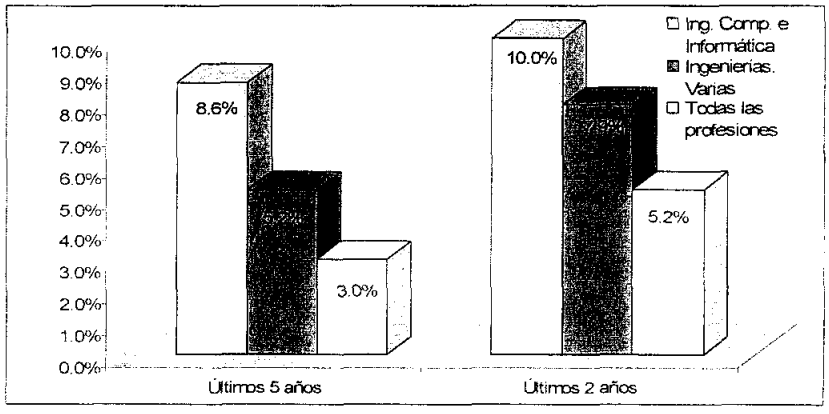
En una encuesta realizada desde 1998 a 2007 por el observatorio laboral de la STPS[73], entre los profesionistas que estudiaron una licenciatura o ingeniería de computación e informática, se observó que la profesión de ingeniero en computación e informática en los últimos dos años (1995-1997) tuvo ritmos del 10% de crecimiento anual, con respecto a otras ingenierías y el resto de las profesiones. Este dato es significativo porque este ritmo de

[73] http://www.observatoriolaboral.gob.mx/ acceso 18 09 07

crecimiento en el empleo entre ingenieros de computación e informática se mantiene en promedio en 8.6%, contra otros ritmos menores de otras ingenierías; indicadores que permiten advertir una presencia importante de estas profesiones en el mercado laboral (ver gráfica 9).

Gráfica 9

Porcentaje de crecimiento en distintas profesiones comparadas con los Ingenieros en computación e informática en México. ocupados al 1er. trimestre de 2007

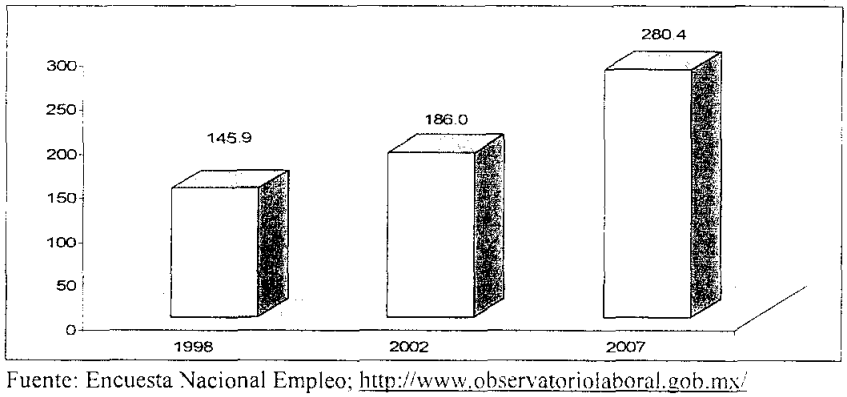


Fuente: Encuesta Nacional Empleo; http://www.observatoriolaboral.gob.mx/ acceso 18 09 07. (Cifras anualizadas al primer trimestre del 2007 de la ENOE)

Este posicionamiento del empleo profesional de los ingenieros en computación e informática refleja el crecimiento de empleo de este tipo de profesionistas, que según datos de la gráfica 9 el numero de empleados remunerado se incrementó de 146 mil en 1998 a poco mas de 280 mil en el primer trimestre de 2007, representando un crecimiento del 92.2% (ver Gráfica 10).

Gráfica 10

Profesionistas ocupados-remunerados en las ingenierías de computación e informática en México. Años seleccionados. (Datos en miles)



Fuente: Encuesta Nacional Empleo; http://www.observatoriolaboral.gob.mx/ acceso 18 09 07. (Cifras anualizadas al primer trimestre del 2007 de la ENOE)

Un segundo aspecto es que 27.4% señala una ocupación general como Ingeniero industrial o arquitecto, con "títulos" sociales y tradicionales, reconocidos en el medio, mas allá de aquellas ocupaciones distintivas del Ingeniero en computación e informática como el

Arquitecto de software, Ingeniero del Software o de Diseño o programador de Software, desarrollador de Sistemas, tester, probador de calidad, diseñador de interfaz, etc. Ninguna de estas ocupaciones del sector de computación e informática son evidentes entre los ocupados.

Cuadro 18

Porcentaje de las principales ocupaciones donde se desempeñan los que trabajan y estudiaron ICI en México. primer trimestre de 2007

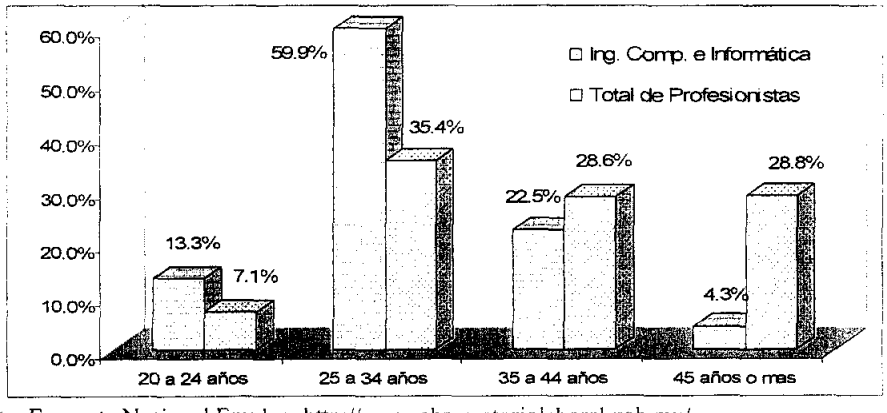
Arquitectos, ingenieros civiles, químicos, ingenieros industriales, otros.	27.4%
Técnicos en dibujo, ingeniería y operación de equipos de imagen y sonido.	6.4%
Directores, gerentes, y administradores de áreas o establecimientos, empresas, instituciones y negocios públicos y privados.	7.5%
Jefes de departamento, coordinadores y supervisores en servicios estadísticos, informáticos, ingeniería e investigación.	5.9%
Secretaria, taquígrafos, capturitas y similares.	7.2%
Otros	**45.6%**

Fuente: Encuesta Nacional Empleo; http://www.observatoriolaboral.gob.mx/
Acceso 18 09 07. (Cifras anualizadas al primer trimestre del 2007 de la ENOE)

Lo reciente de estas profesiones de computación e informática no ha logrado consolidarse socialmente o "infiltrarse" en el estatus socio-profesional, no se instituye como una profesión de reconocimiento social, no se embebe socialmente un término que de cuenta de la heterogeneidad profesional, como si lo hizo la ingeniería civil y la arquitectura. Según observamos en la gráfica 11, 73.2% de los Ingenieros en computación e informática son jóvenes, con edades que oscilan entre 20 y 34 (ver gráfica 11). Porcentaje que coincide con los datos señalados por PROSOFT (2004), donde 91% de los programadores de software tienen menos de 15 años de antigüedad como profesionistas, es decir se insertaron en el mercado profesional hacia fines de los década de los ochenta.

Gráfica 11

Porcentaje de profesionistas en Ingeniería en Computación e Informática versus el total de los profesionistas en México. Por grupos de edades.

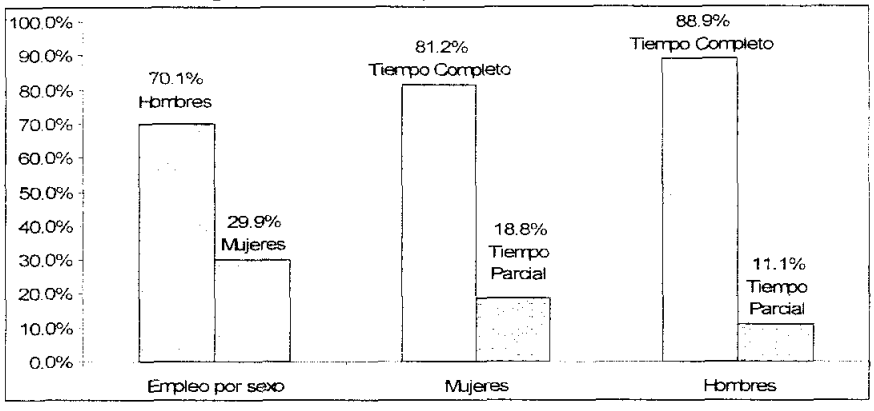


Fuente: Encuesta Nacional Empleo; http://www.observatoriolaboral.gob.mx/
Acceso 18 09 07. (Cifras anualizadas al primer trimestre del 2007 de la ENOE)

En cuanto al tipo de jornada de trabajo, la gráfica 12 nos muestra la tendencia a la masculinización de la profesión, donde de cada 10 ocupados en la ingeniería de computación e informática siete son hombres, aunque las diferencias en tipo de contratación a tiempo completo o parcial por genero no son significativas, 81.2% de las mujeres contratadas son a tiempo completo y el de hombres es 81.2%.

Gráfica 12

Jornada de trabajo y empleo por género en los profesionistas que estudiaron la carrera de Ingeniería en Computación e informática en México.



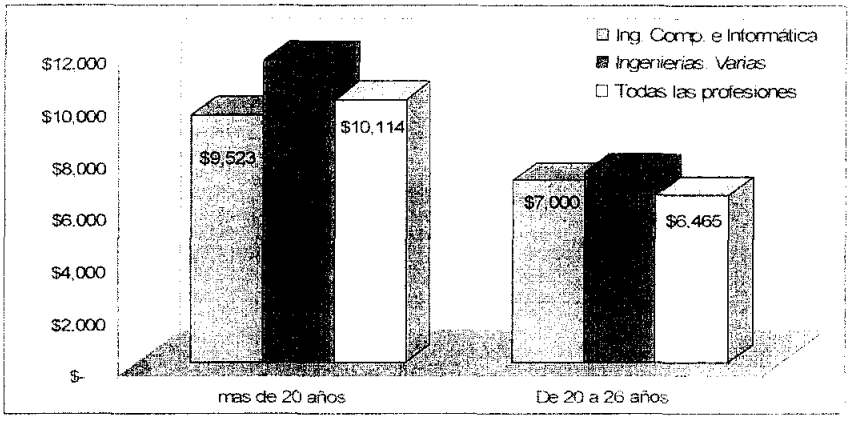
Fuente: Encuesta Nacional Empleo; http://www.observatoriolaboral.gob.mx/
Acceso 18 09 07. (Cifras anualizadas al primer trimestre del 2007 de la ENOE)

En lo relativo a las renumeraciones, diferenciaremos según la formación entre[74]: 1) las personas ocupadas que estudiaron esta carrera; 2) los profesionistas que estudiaron carreras similares; 3) de todas las personas ocupadas en el país que estudiaron una carrera profesional. Asimismo, se presentan las mismas comparaciones de ingresos para las personas con trabajo que tienen edades entre 20 a 26 años y que estudiaron una carrera profesional. Los ingresos se refieren al sueldo mensual percibido (neto). La gráfica 13 tiene dos lecturas, la primera es que si excluimos el rubro de jóvenes de mas de 20 años, tenemos que los profesionistas entre 20 a 26 años reciben prestaciones económicas cuasi-homologadas, hecho que seguramente refleja la falta de experiencia, habilidades y destrezas del personal; la segunda, tiene que ver con que cuando se observa la columna de mayores de 20 años, destaca que el sueldo mensual neto de los Ingenieros en computación e informática es menor que el resto de las profesiones. Estos datos parecen plantear una paradoja ¿Profesiones de alta exigencia técnica e intelectual en condiciones de remuneraciones precarias?

[74] Datos construidos a partir de la Encuesta Nacional de Ocupación y Empleo (ENOE); 2005, Secretaría del Trabajo y previsión social; 2004; Disponible en Internet, Acceso Febrero de 2005.

Gráfica 13

Sueldo mensual neto en distintas profesiones comparadas con Ingenieros en Computación e Informática en México. Primer trimestre de 2007



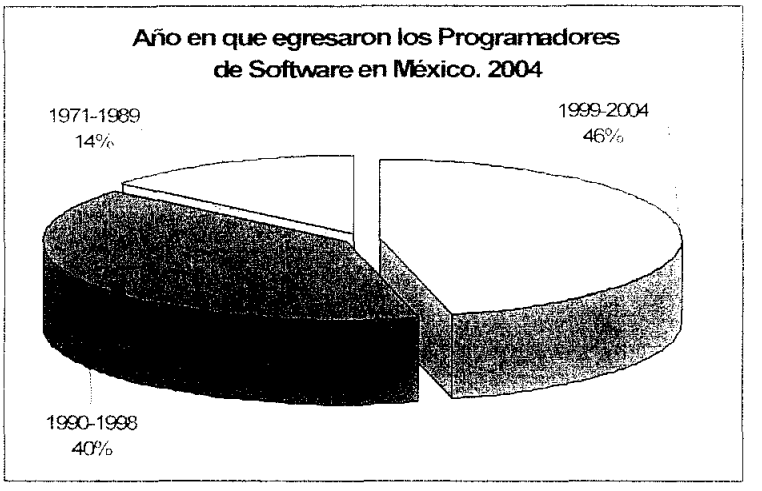
Fuente: Encuesta Nacional Empleo; http://www.observatoriolaboral.gob.mx/
Acceso 18 09 07. (Cifras anualizadas al primer trimestre del 2007 de la ENOE)
Nota: Los ingresos se refieren al sueldo mensual percibido (neto).

6.3.- Perfil de una ocupación reciente

Como señalábamos en el apartado anterior la profesión de los ingenieros en computación e informática es reciente en el mercado profesional, la demanda para este tipo de profesiones no es del todo precisa, en otras palabras, esta profesión aún no adquiere el "rango de reconocimiento social"con un conjunto de experiencias sociales, técnico-profesionales, acumulación de títulos educativos, centros de investigación, etc. que representan una expresión acumulada de años de *construcción social de la ocupación*,por ahora se percibe como una ocupación para y de jóvenes y paralelamente compleja para los adultos; percepción que es señalada como una brecha digital del tipo generacional (Crovi:2004). La profesión de la computación e informática en el argot técnico considera una convergencia entre electrónica-informática-telecomunicaciones, ésta "triple hélice" de profesiones ostenta cada una por su lado, una serie de limitaciones técnicas que les hace a su vez interdependientes; por ejemplo la informática no aceleró la transmisión de la información hasta que las telecomunicaciones a su vez necesitaron de una transformación del uso de cables de cobre a fibra óptica, pero la fibra óptica se aplicó en la medida que la electrónica ofreció microprocesadores eficientes capaces de procesar simultáneamente video, imagen y sonido, es decir mayor procesamiento de información. Como eje transversal, hace presencia la Ingeniera del Software que comprende

un poco de electrónica (hardware) y flujos de información y transformación de ésta en conocimiento codificado (software)[75].

Gráfica 14



Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México" Secretaría de Economía-UAM. 2004.

En la literatura de ingenieros es común percibir que la industria del software esta en proceso de consolidación, que hace falta superar la "etapa artesanal", concepto común entre gerentes y líderes de empresas de software. La "etapa artesanal" hace referencia no sólo al hecho de que la ingeniería del software es relativamente nueva en comparación a "otras" ingenierías en donde las ocupaciones están socialmente divididas en arquitectos, ingenieros, ingeniero en electrónica; ingeniero químico, etc. También hace referencia al contexto "joven" de los programadores puesto que 86% se graduaron después de 1990; 87% están centralizados en equipos de trabajo con menos de 10 empleados y 83% están concentrados en micros, pequeñas y medianas empresas.

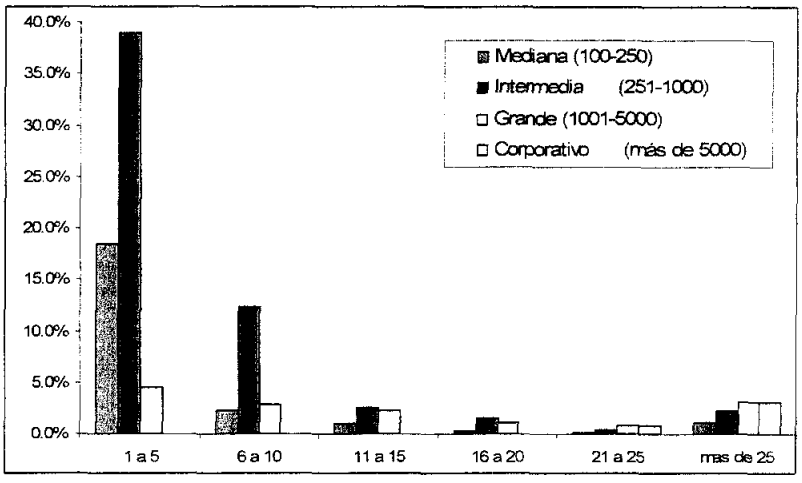
La industria del software es inédita en la región, posee pocos años de madurez para que las *relaciones de interacciones sociales* cuajen en redes, comunidades y colectividades que configuren *relaciones sociales competitivas*, por el contrario las fuerzas productivas son de reciente formación (ver gráfica 15). Esta concentración y lo reciente del trabajo, explica en parte porque en algunas entrevistas con gerentes de software señalaban que existe alta rotación laboral, insuficientes habilidades y conocimientos entre programadores "lo mas difícil es

[75] De aquí en adelante son estadísticas, gráficas y datos tomados y construidos a partir de PROSOFT, (2004) "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México", elaborado por la Universidad autónoma metropolitana a encargo de la Secretaria de economía.

ponerme a enseñarles...a decirle que deben de empezar a hacer....pero aprenden rápido" (gerente RF1), es decir que las empresas poseen incipientes aprendizajes, por varias razones, destacando el tamaño de los centro de desarrollo de software, que, según observamos en la gráfica 15, 39% están concentradas en empresas medianas (100 a 250 empleados) y 17.4% en intermedias (251 a 1,000 empleados).

Gráfica 15

Distribución de los profesionistas del Software por tamaño de empresa versus tamaño del área de sistemas en México. 2004



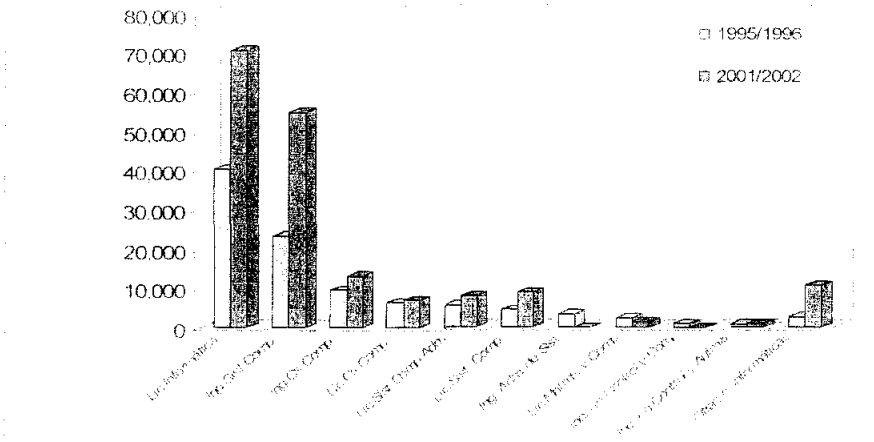
Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México" Secretaría de Economía-UAM. 2004.

Las áreas de desarrollo de software están centralizadas en unas cuantas personas (nótese en la gráfica 15 que las empresas poseen pequeñas áreas de sistemas concentradas en menos de 5 empleados, es decir 69.5% de los equipos de trabajo del área de sistemas se compone de 1 a 5 empleados y 17.5% de 6 a 10 profesionistas de software ocupado en empresas que poseen sus propios centro de desarrollo de sistemas informáticos, concentrándose 87% de áreas de sistemas informáticos con menos de 10 trabajadores.

Otro ingrediente cualitativo que nos permite visualizar lo incipiente de la ocupación, es el abanico de ofertas entre ingenierías en computación y las licenciaturas en informática que son ofertadas por la educación superior (ANUIES-INEGI: 2006), según vemos en la gráfica 16 tan solo la educación pública ofrece un abanico superior a 14 profesiones entre ingenierías en computación y licenciaturas en informática; a esta lista habría que agregar las de educación superior privada, diplomados y postgrados (públicos y privados).

Gráfica 16

Matrícula de nivel licenciatura en tecnología de información y comunicaciones por carreras, 1995/1996 y 2001/2002



FUENTE: Elaboración propia en base a INEGI, http://www.inegi.gob.mx Acceso Julio de 2007

La concentración en la matricula, según la gráfica 16 a nivel licenciatura ofrecida por la educación superior se incremento significativamente en el ciclo escolar 2001/2002 y las preferencias académicas se orientaron a Ingeniería en computación y licenciatura en informática. Según datos de la gráfica 16 para 2002, 40% de las inscripciones fueron en licenciatura en informática; 31% en ingeniería en sistemas computacionales y 7% en ingeniería en ciencias de la computación; éstas 3 profesiones concentran 78% de la demanda de las profesiones, tan solo las primeras dos carreras juntas, en 2002 se ubican en el cuarto lugar de las profesiones con mas población atendida y, ocupan el primer lugar en las ingenierías. Las preferencias entre ingeniería en computación y licenciatura en informática ha implicado un debate no explicito entre empleadores y trabajadores. A continuación lo abordamos como preferencias no escritas entre la preferencia por contratar técnicos o profesionistas (ver gráfica 16).

Este debate no escrito entre contratar profesionistas y técnicos lo percibimos en algunas entrevistas realizadas en la zona metropolitana en la ciudad de México, entre Mayo y Septiembre de 2006, observamos más de tres posturas entre líderes de proyecto y programadores, con respecto a que tipo de profesiones, calificaciones y cualificaciones deberían impartir las universidades y que perfil profesional debe contener a quien se contrata: por el momento, señalaremos tres posturas de los gerentes:

- La industria del software requiere técnicos especializados, no profesionales que sólo desean "tirar código", ignorando metodologías y métricas de trabajo.

- Si se requiere de profesionistas con carrera universitaria, pero no poseen las cualidades y habilidades que la industria requiere; es decir, no poseen el perfil profesional que la industria demanda.
- No es suficiente la educación superior, deben existir vínculos con la industria local, es decir los programadores una vez que se gradan no poseen práctica suficiente real de los conocimientos que demanda el mercado y el sistema de cualificaciones de los profesionistas (desvinculación empresa-universidad).

El debate plantado en el mercado profesional del software entre empleadores y empleados es rudimentario, sin embargo funciona más como un indicador que nos señala no sólo lo incipiente de las *interacciones sociales profesionales*, sino que también se está en un proceso reciente de la *consolidación social de la ocupación de programador de software*, que emerge entre debates señaladas por gerentes, líderes de proyecto y programadores; polémicas que no necesariamente son contradictorios; por el contrario son expresión de la complejidad, ambigüedad y claros-obscuros de un proceso de trabajo reciente. Trabajo que se conforma a su vez de un complejo renglón tecnológico, que hace presuponer que no se contrata mano de obra descualificada, sino que existe mano de obra especializada. Estas posturas oscilaron entre los entrevistados que señalan la diferencia entre contratar técnicos profesionales y profesionistas universitarios. Aquellos que consideran que se debe contratar personal con perfil de técnicos, argumentan que ello es deseable porque de ésta manera es posible que las empresas son las desarrollen las habilidades y conocimientos necesarios; en cambio cuando contratas un profesionista universitario éstos "aprenden trabajando y cuando mejoran sus capacidades personales...piden que les pagues mas o simplemente se van" (Gerente TADI DC2). O bien, a medida que adquieren mayores conocimientos formales, acumulan experiencias, "saben hacer" mejor los procedimientos y, cuando eso sucede el programador se percibe así mismo que puede anhelar mejores condiciones de trabajo, y emprende una posición de "hacen como que me pagan, hago como que trabajo", otros programadores asumen una posición de "primas donas" que lo único que quieren hacer es programar sin que nadie les moleste y sin involucrarse en otros temas como el diseño, pruebas de calidad, o en documentar el proceso (gerente DYYA RF2). Por el contrario aquellos gerentes y lideres que señalan que los programadores deben poseer preparación académica a nivel profesional, solo que debe haber

cauces de comunicación empresas-universidad, impulsarse entre ambos espacios acuerdos para estancias, prácticas profesionales, e incluso que los empresarios formen parte de los académicos.

Está polémica entre contratar técnicos o profesionista en el proceso de trabajo del software, en una primera aproximación parece no reflejarse en el mercado laboral, ya que cuando vemos los datos del cuadro 19, observamos que 78.3% de los Ingenieros de software poseen una carrera profesional terminada y, sólo 19.6% declaró ser técnico; 12.9% señaló ser técnico medio superior y, 6.7% superior universitario. Sin embargo, cuando observamos el detalle por grupos de edad, el debate se replantea, según podemos ver en el cuadro 19, los profesionistas del software no sólo están conformados en su mayoría por "ingenieros o licenciados" como apreciamos anteriormente, sino que entre los mas jóvenes de 18-23 años. 43.6% declaró ser técnico y 54.6% ser profesional, suponemos que este crecimiento en los jóvenes técnicos podrían ser porque hay graduados del sistema de tecnológicos profesionales de reciente presencia en el sistema educativo mexicano (ver cuadro 19).

Cuadro 19
Formación académica versus edad de los profesionistas
de Software en México.2004

	Total	18 - 23	24 - 30	31 - 35	36 - 40	+ de 40
Total	100.0%	8.9%	46.8%	21.7%	14.6%	8.0%
Técnico medio superior	**12.9%**	**33.7%**	8.7%	15.4%	8.1%	16.5%
Técnico superior universitario	6.7%	**9.9%**	6.4%	5.7%	6.1%	8.4%
Profesional	**69.3%**	44.5%	75.5%	65.8%	71.4%	66.9%
Profesional y Técnico	9.0%	10.0%	8.3%	9.5%	12.9%	4.1%
Ninguno	2.0%	2.0%	1.1%	3.6%	1.5%	4.1%

Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México" Secretaría de Economía-UAM. 2004

Otra lectura importante del cuadro 19, está relacionado con los rangos de edad, donde observamos que 68.5% son jóvenes entre 24 y 35 años, lo cual significa que de cada diez programadores jóvenes, siete son profesionistas; por el contrario, los mas jóvenes de entre 18 y 23 años, sólo cuatro de cada diez son profesionistas, dato que parecería ser contradictorio para una profesión que supone alta especialización. Este debate se complejiza cuando examinamos el conjunto de habilidades, conocimientos y destrezas que deben conocer los programadores de software, inmersos éstos en un contexto de reciente conformación, con trabajadores con pocos años de experiencia, destacándose que 76% posee grado universitario.

81% menos de 15 años de experiencia y 77% son menores de 35 años; es decir, es un mercado de trabajo joven, profesionalizado. Éste dato se observa, cuando los gerentes advierten que los programadores de software deben ser mas "técnicos", hacen referencia a la *división social del trabajo hacia el interior de la empresa*, es decir que los programadores se dediquen mas al aspecto práctico, cotidiano de la programación, de la codificación; proceso que no es sencillo, es complejo, álgido, ya que representa el punto en el cual se traduce el diseño del sistema informático en algoritmos que resuelve y soluciona los problemas planteados en el proceso de trabajo. Pero por otro lado, existe un problema central que es la complejidad en el proceso de abstracción, en el proceso de solución de problemas, que este trabajo implica para el programador. Complejidad que la ingeniería del software señala como "aflicción del software", que representa un conjunto de problemas, errores y defectos en el proceso de trabajo, problemas como la falta de documentación y comentarios de el conjunto de abstracciones y soluciones algorítmicas, incumplimiento en tiempos en la entrega, inobservancia en los requerimientos señalados por los clientes en el diseño original, etc.

Consideramos que las condiciones externas e internas en el proceso de trabajo, aunado a la interpretación subjetiva de los programadores llevan a que gerentes, y directivos y algunos académicos consideren que es pertinente impulsar el aspecto administrativo de la programación, es decir, que sea la gerencia quien realice el diseño, señale los requisitos de los módulos, y sólo encomendar a los programadores el aspecto de la codificación de los algoritmos. Lo que intentamos señalar es que, por un lado, se desea que el área administrativa predomine en el proceso de trabajo, ya que este es un trabajo altamente dependiente del trabajado cognitivo, no sólo incertidumbre en cuanto al procedimiento, sino también la documentación de lo que se hizo y porque se hizo tal procedimiento cognitivo.

> "...yo no he tenido que darle cuentas a nadie, entonces, de ahí, me cuesta trabajo el especificar lo que tengo que hacer (documentar el proceso), por que muchas veces al no tener que rendirle cuentas a alguien, lo vas haciendo conforme ves que te van llegando a veces las ideas, esto te hace como hacer lo que vas hacer... pero si me preguntas ¿que voy hacer mañana?, igual todavía no lo defino..." (Programador, JO3 MIXE).

La incertidumbre en el proceso de trabajo forma parte de la *aflicción del software*, y presiona estructuralmente al proceso de trabajo, consideramos que está presente un contexto de *resistencia al control* del trabajo al interior del proceso de trabajo. Lucha que veremos más

adelante e implica una serie de *consensos y arreglos sociales al interior del proceso de trabajo.* Por lo pronto abordemos un aspecto importante que tiene que ver con la división entre la administración y la parte técnica del proceso de trabajo del programador, donde también existe una confusión entre las tareas desempeñadas.

El cuadro 20 plantea una doble reflexión, la primera es que, así como hay una amplia gama de profesiones que integran en sus programas académicos la enseñanza del desarrollo de Software (ver gráfica 20) también es heterogéneo el conjunto de ocupaciones administrativas o técnicas; en las primeras 30.4% declaró ser Jefe o gerente, seguido de definiciones no muy claras como encargado (10.5%), Asistente/Auxiliar con 9.4%, etc. Diversidad de nombres que también se aplica en el caso de las ocupación técnicas donde los puestos mas representativos son ingeniero, programador, desarrollador y analista; en cambio el puesto de Líder de proyecto sólo es ponderado con 1.4% (ver cuadro 20).

Cuadro 20

Puestos que ejercen los profesionistas del software en México. 2004

INGENIERÍA	30.1%	ADMINISTRATIVO	63.1%
Análisis / Analista	5.5%	Consultoría / Consultor	0.6%
Desarrollo / Desarrollador	1.8%	Informática	0.3%
Especialista	0.6%	Administración / Administrador	5.3%
Operaciones / operador	2.1%	Coordinador / Coordinación	4.4%
Programación/ Programador	3.7%	Dirección / Director	1.1%
Ingeniería / Ingeniero	6.7%	Gerencia / Gerente	16.1%
Sistemas	0.9%	Subdirección / Subdirector	0.3%
Supervisión / Supervisor	2.9%	Subgerencia / Subgerente	0.8%
Técnico	1.4%	Asistente / Auxiliar	9.4%
Soporte técnico	3.1%	Encargado	10.5%
Líder de proyecto	1.4%	Jefe	14.3%

Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México" Secretaría de Economía-UAM. 2004. El porcentaje faltante del 7% es en Otros puestos de trabajo.

En el cuadro 20 damos cuenta de las ocupaciones dividiéndolas en dos columnas los administrativos (63.1%) y las de ingeniería (30.1%), esta distribución es significativa en el sentido que podemos tomarlo como un indicador de la *construcción social de la ocupación*, la construcción representa un doble debate, el primero es en torno al área que tiende a dominar, a presionar estructuralmente en el proceso de trabajo; a su vez este proceso de trabajo es una combinación *sui generis* entre elementos técnicos de ingeniería y proceso de administración.

El otro debate, es hasta que punto estamos en presencia de un intento primigenio de establecer formas fordistas/tayloristas de la administración científica; por un lado sería la

gerencia quien diseñara y concebirá el *como se hace* y *porque se hace*, y, por el otro estarían los que hacen la ejecución técnica del trabajo (Taylor, 1983).

Un argumento constante en el discurso de gerentes y programadores, es la desvinculación existente entre academia y sistema educativo. Plantean que existe un distanciamiento entre conocimientos formales que demanda la industria y la plantilla de materias que se imparten en las Universidades, brecha que amplia la curva de aprendizaje de los programadores una vez que son contratados en la empresa, lo cual se traduce en mala calidad en el desarrollo del proceso de trabajo.

> "Los clásicos programas de estudio de las universidades muchas veces no están actualizados y entonces cuando los jóvenes se enfrentan a la vida real se dan cuenta que están desfasados en metodologías, en tecnologías, en todo lo que deben de estar (*conocer, procesos, métricas de calidad, etc.**) dentro del ámbito productivo" (Líder, JA2, MIXE * señalamiento nuestro).

En las entrevistas los programadores también señalan esta brecha, en el sentido, que perciben que les hacen faltan determinado tipos de conocimientos. A preguntarles con respecto a que tipos de conocimientos consideraban que les hacia falta, opinaban que en administración de proyectos, habilidad en el trato con el cliente y falta de práctica "real" en el desarrollo de programas de Software (programador, JO3, MIXE).

6.4.- Construcción social del *saber hacer* en el trabajo cognitivo

Consideremos que el discurso que está atrás del debate en torno a la división del trabajo entre administrativos y técnicos, (ver cuadro 24, gráfica 18 y cuadro 25) que es el control del trabajo y determinar las tareas y actividades del programador, este discurso puede leerse en dos sentidos: a) Con nivel técnico de trabajadores-programadores, se estaría en condiciones de aprender de la empresa, sus procedimientos y sus reglas de programación acorde a las prácticas de la empresa misma, que se adopten y reconozcan la "forma" de desarrollar código del equipo de trabajo de la empresa; b) La adquisición de competencias se realizaría en la empresa, es decir se "instruirían" e identificaran con la empresa. Con la intención de crear en ellos un sentimiento de identidad para con esta.

Los trabajadores jóvenes no sólo no tienen experiencia, sino que además cambian de trabajo fácilmente, son más autónomo con respecto al "donde" ejercer su profesión, y proclives a cambiar de trabajo, aunado a ello esta presente una propensión a ser auto-empresarios. JA2 gerente y líder de proyecto en la empresa MIXE -con más de 250

programadores- es un experimentado ingeniero de software, quien comentaba que muchos de los ingenieros de software consideran que en la Universidad les educan en un espectro amplio de Ingeniería del Software, sin embargo es poco el tiempo que dedican a la práctica en el desarrollo de líneas de código, a la complejidad que implica elaborar algoritmos. Por tal motivo, JA2 es uno de los promotores líderes en impulsar un convenio entre Universidades y Empresas desarrolladoras de Software en la zona metropolita del Distrito Federal que se llama tecno-eje (http://www.tecnoeje.org/), éste aglutina más de 14 universidades de la región, tiene un programa denominado "generación de talentos" que es una estancia de estudiantes de Ingeniería de Software en las Empresas desarrolladoras de Software. Ahí aprenden a desarrollar software en "aprendizaje real".

Otras características que debemos tener en cuenta en este debate entre contratar técnicos ó profesionistas universitarios, quizá influya la alta rotación laboral y "pirateo" de profesionistas. El gerente del área de programación DC2, de la empresa CATI, comentaba que cerraron el área de desarrollo de Software debido a la alta rotación de los programadores y los altos costos que implicaba capacitarlos. Otro gerente RS2, de la empresa TADI, señalaba que en ese momento estaba "negociando" con un programador "talentoso" para incrementarle el salario, ya que en otra empresa le habían ofrecido mejores prestaciones económicas, y estaban negociado para igualar la cantidad que le ofrecían en la otra empresa.

> "...Lo chistoso es que cuando se tiene una gente capacitada, para la parte de Java y en un Aplication Server, -si los encuentra uno- es muy fácil contratarlo... pero al cabo de seis meses ya se sienten reyes, dioses y emigran (...) entonces es mejor que otro sufra esa forma a estarlo sufriendo directamente dentro de la empresa... por que toda la experiencia que se puede llegar a obtener de una persona que has estado tratando de levantarla la pierdes un momento (247:259). Hay una rotación muy alta (...) el costo de capacitar a una persona ya habiendo salido de la escuela, es alrededor de 12,000 o 14,000 dólares. Y si no te trabaja por lo menos año y medio pues no te conviene (Gerente, DC2, CATI).

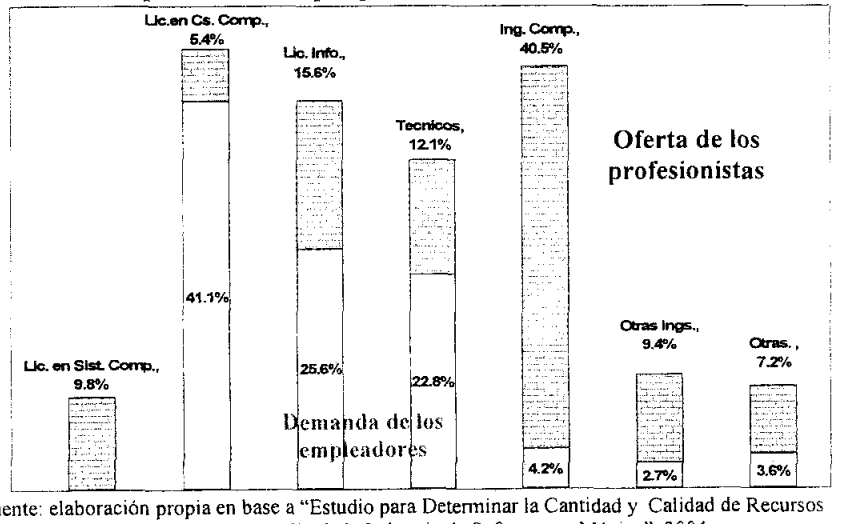
El gerente DC2, de la empresa CATI poseía una importante área de desarrollo por mas de 10 años (1990 a 2004), señala que los márgenes de ganancia por software desarrollado son cada vez menores, debido a la competencia de programadores individuales (freelance), rotación de programadores, altos costos de capacitación, y la presencia de empresas grandes compitiendo en el mercado de Software, lo cual les llevó a cerrar el área de programación, y subcontratar el desarrollo de los proyectos de software de la empresa (outsourcing), incluso

subcontratando a ex-empleados (Gerente, DC2, CATI). Consideramos que este debate, forma parte de un contexto más amplio relacionado con una reciente construcción social de la ocupación, tanto por las características ya señaladas, como por la posible configuración de un debate primigenio en torno al tipo de calificaciones y cualificaciones que demandan los empleadores ó gerentes, representada por la administración y empresarios y, por el lado la oferta los trabajadores-programadores, quienes ofrecen su saber hacer, aprehenden las "necesidades" de la nueva empresa, aquellos que comparten información entre sus iguales ya sea a través de Internet en comunidades simbólica de trabajo o bien en portales electrónicos personales como blogs, wikis, live.com, entre otros.

A continuación abordaremos el presente debate desde dos perspectivas, por el lado de los empleadores quienes demandan un determinado perfil y quienes ofertan un conjunto de habilidades profesionales. Si bien es cierto hemos señalado un amplio abanico en los puestos de trabajo éstos se "dividen" entre el área de ingeniería (30.1%) y área administrativa (3.1%). Se suma a esta división el perfil profesional. Al analizar el perfil de profesional que demandan los empleadores de software versus la oferta de la fuerza de trabajo en términos de profesiones el abanico de posibilidades se complejiza. Según datos de la gráfica 17, en lo correspondiente al perfil de la demanda por parte de los empleadores, observamos que se requieren 66.7% credenciales académicas centradas en licenciaturas, como las de computación (41.1%) e informática (25.6%); y 22.8% con nivel de técnicos.

Gráfica 17

Carreras estudiadas por los profesionistas de Software (Oferta) versus Demanda de empleadores de programadores de Software en México. 2004.



Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humano Necesarios para el Desarrollo de la Industria de Software en México". 2004, Secretaria de Economía-Universidad Autónoma Metropolitana.

La divergencia marcada entre una demanda por parte de los empleadores que solicita 66.7% con perfil profesional del tipo licenciado en informática y 22.8% como técnicos versus una oferta profesional de programadores con 49.9% concentrada en ingeniería y sólo 9.8% en licenciado en sistemas computacionales, así como el nulo perfil de técnico. Sin embargo tanto la demanda como la oferta profesional de los programadores de software esta condicionada por la existencia de una oferta profesional por parte de la educación superior, sea pública o privada (ver gráfica 17). Atrás de este debate parecería ser que los empleadores insisten en una *primigenia división del saber hacer* en este tipo de trabajos a través de licenciaturas que partan de que el desarrollo del software debe ser a través de *ciclo de vida* del software[76] que este dividido en cuatro grandes áreas: Conceptualización, formalización, procesamiento de datos e implementación. De esta forma parecería que las profesiones de tipo licenciado técnico poseen esta orientación del tipo administrativa. Mientras que la oferta de los profesionistas, considera que las ingenierías del software concentran aún todo el proceso de trabajo.

Esta reflexión deviene de la trayectoria de la ingeniería del Software que en un primer momento (1940-1970) abarcó todo el proceso en su conjunto del desarrollo del Software, sin embargo algunos autores expertos en la materia como Pressman (2002), Sommerville y otros (2005) señalan que se ha complejizado a tal punto el desarrollo de software que requiere el apoyo de nuevas ingenierías como es la ingeniería de requisitos, Ingeniería de la usabilidad, entre otras (Palacio, 2007). Mientras que la ingeniera del software detenta todo el proceso en su conjunto, señala lo complejo que es segmentar un proceso abstracto, basado en el conocimiento, donde a lo mucho propone métodos y proceso, herramientas y métricas de calidad, pero reconoce que no existen recetas o "un solo mejor camino" para desarrollar software. En este sentido los programadores de software, por el lado de la oferta procuran instruirse en profesiones que analizan todo el proceso en su conjunto, donde se contempla las diferentes posibilidades en el desarrollo del software, donde se enuncia las fases del conocimiento en *el ciclo de vida* del desarrollo de un sistema informático. Este debate no sólo esta presente en las estadísticas, sino que también lo apreciamos en las entrevistas realizadas en el Distrito Federal, los gerentes señalaban que "hacen falta buenos diseñadores, profesionistas que sepan como planear el ciclo de vida...el desarrollo de un programa" así que

[76] El ciclo de vida en el software hace referencia al recorrido en el desarrollo de un sistema, desde el momento en que el cliente solicita un programa informático, hasta el punto en que se implementa el programa en la empresa del cliente ya funcionando.

la empresa debe invertir en capacitación, en aprendizaje de los programadores, y cuando ello sucede, los programadores se van a otra empresa donde les pagan mejor (Líder, JA2, MIXE) En este punto coincidieron varios gerentes y lideres, como en aquel de que es mejor contratar técnicos.

> "en el momento en que ya pasan a la vida productiva, el esfuerzo para nosotros es capacitarlo (...) normalmente como tardaban en dar resultados obviamente no había una remuneración alta para el recurso y, entonces cuando iba aprendiendo y ya se fortalecía... se daban cuenta que ya estaban listos para irse con otra empresa y entonces migraba, y nosotros lo que estábamos haciendo, era preparar gente para que otros tuvieran el mayor provecho" (Líder, JA2, MIXE).

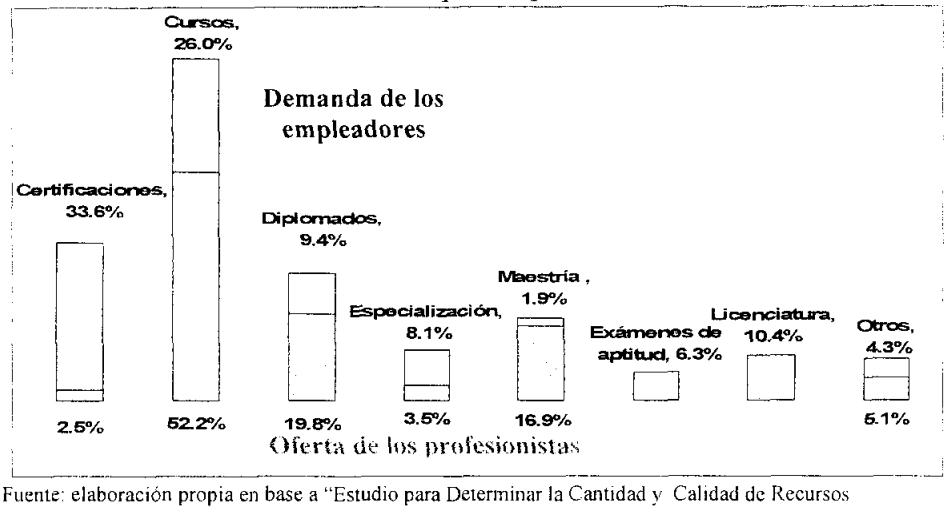
Por el contrario en entrevistas realizadas a programadores de Software, señalan que este es un mercado de trabajo que debe ser continuamente profesionalizado, que está especializado, expresiones que coinciden con 49.9% de los encuestados que declararon haber estudiado una Ingeniería y solo 12.5% estudió nivel técnico (ver gráfica 18).

> "....en la escuela yo creo que lo que te hacen es ejercitar la resolución de problemas porque no hay... no había mucha practica... entonces realmente donde te formas es en el ámbito laboral cuando ya te tienes que enfrentar a un problema y tienes que resolverlo tomas el sustento que tienes de resolver problemas, de buscar información y entonces es cuando empiezas a aprender realmente a solucionar problemas en el ámbito real (Programador, TADI MZ3)

El debate entre empleadores y oferentes continua, por ejemplo abordemos a continuación el tema relacionado con el tipo de conocimientos explícitos adicionales que deben poseer los profesionistas de Software más allá de los títulos académicos, se aprecia dos contrastes. El primero esta relacionado al hecho que 88.9% de los programadores de software se distingue por preferir estudios académicos-formales, como maestrías, diplomados y cursos. Por el contrario los empleadores demandan 37.3% de estudios formales, de los cuales sólo 1.9% corresponde a estudios de postgrado (ver gráfica 23). En segundo lugar observamos un contrasentido entre la preferencia de los programadores por cursos con 52.2%, y, por el contrario, en los empleadores sólo 26% comparte esta perspectiva y poco mas de un tercio (33.6%) de los empleadores consideran mas beneficioso la asistencia a certificaciones (CMM, MOPROSOFT, ISO, etc.) ya que consideran que estas aportan valor agregado a las empresas, caso contrario los programadores que sólo 2.5% considera certificarse (ver gráfica 18).

Gráfica 18

Estudios adicionales de los profesionistas de Software (Oferta) versus estudios demandados por empleadores en México. 2004



Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humano Necesarios para el Desarrollo de la Industria de Software en México". 2004, Secretaría de Economía-Universidad Autónoma Metropolitana.

En otras palabras, de cada cien programadores 52 consideran que se debe de estudiar cursos, 17 maestrías y 20 Diplomados. En contraste, los empleadores sugieren que los programadores deben tener conocimientos adicionales como certificaciones (33.6%) y cursos (26%). En cuanto a estudios no académicos, las tendencias son marcadas hacia los cursos y certificaciones. La tendencia hacia estudios formales consideramos que se debe al hecho que este tipo de credenciales formales permiten fortalecer y ampliar el conjunto de habilidades y destrezas personales en torno al contexto del proceso de trabajo y posicionarse frente al trabajo, más no para ascender de nivel o puesto de trabajo.

> "(...) si me mandaron a un curso, pero no fue así muy sofisticado, aprendí ciertas habilidades, pero básicamente es como para reforzar el entendimiento y la comunicación que debe de haber en los diferentes roles de un proyecto, no tanto para avanzar a un siguiente nivel." (Programador JO3, MIXE)

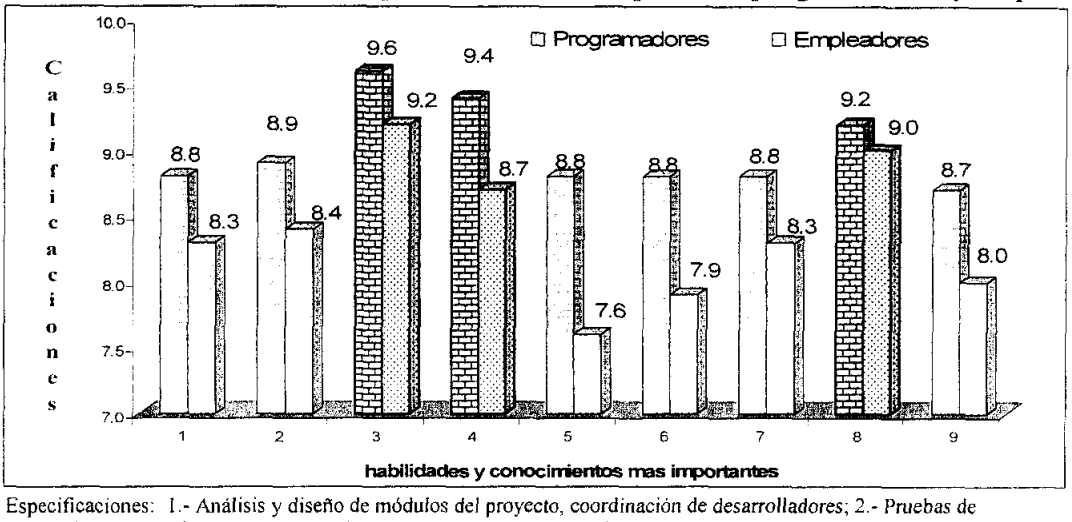
O bien, consideran que los cursos son muy puntuales, en novedades tecnológica, procesos, metodologías, etc.

> "Nos ha enviado a cursos. A mí me enviaron a uno para la tecnología que viene para 64 bits. Actualmente las máquinas trabajan a 32 bits entonces para hacer esa migración a 64 bits hay aplicaciones en las que lleva un proceso para hacer esa conversión, para que funcione los 64 bits" (Programador MIXE GG3).

Otro argumento en torno a la construcción social de la ocupación es el conjunto de habilidades y conocimientos más importantes en los próximos años, que como vemos en la gráfica 19, es una serie de habilidades de razonamiento, conocimientos complejos de procesamiento, así como manejo de sistemas operativos, bases de datos y aplicaciones comerciales, damos cuenta de una lista de técnicas, destrezas y habilidades abstractas que deberán tener los programadores: 1.- Análisis y diseño de módulos del proyecto, coordinación de desarrolladores; 2.- Pruebas de integración de módulo que componen el sistema completo; 3.- Habilidad para razonamiento y resolución de problemas; 4.- Habilidad de comunicación oral y escrita; 5.- Conocimiento de matemáticas; 6.- Conocimiento de procesos industriales y/o de operaciones; 7.- Administración de sistemas operativos; 8.- Manejadores de bases de datos; 9.- Paquetes de Aplicaciones de Software Comerciales.

Gráfica 19

Promedio de calificaciones de las habilidades y conocimientos mas importantes para los próximos tres años en el puesto de programador. Percepción de programadores y empleadores.



Especificaciones: 1.- Análisis y diseño de módulos del proyecto, coordinación de desarrolladores; 2.- Pruebas de integración de módulo que componen el sistema completo; 3.- Habilidad para razonamiento y resolución de problemas; 4.- Habilidad de comunicación oral y escrita; 5.- Conocimiento de matemáticas; 6.- Conocimiento de procesos industriales y/o de operaciones; 7.- Administración de sistemas operativos; 8.- Manejadores de bases de datos; 9.- Paquetes de Aplicaciones de Software Comerciales.
Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México" Secretaría de Economía-UAM. Noviembre 2004.

Según vemos en la gráfica 19, las calificaciones otorgadas por los programadores y empleadores a un conjunto de habilidades y conocimientos como más importantes en los próximos tres años son similares, es decir que para programadores como empleadores las destrezas mas significativas son las cualidades para el razonamiento y resolución de problemas, con calificaciones de 9.6 y 9.2 respectivamente; seguido de otro conjunto de cualidades como es la comunicación oral y escrita, con calificaciones de 9.4 y 8.7. Ahora bien,

los otros dos conjuntos de habilidades, son los conocimientos en el uso de sistemas operativos y de bases de datos, con calificaciones de 8.8 (programadores) y 8.3 (empleadores) para los conocimientos citados (ver gráfica 19).

Lo significativo de estas cuatro cualidades, es que las primeras dos hacen referencia a cualidades de índole subjetivo, son destrezas individuales que tienen que ver con experiencias, hábitos, pericias cognitivas, aptitudes y actitudes frente a un problema y la posición e intereses, representaciones e intencionalidades que significan éste tipo de cualidades. Mientras que el uso de sistemas operativos[77] y bases de datos[78] implica un constante actualización individual, una persistente renovación en el saber hacer personal, ya que dichos conocimientos se valoran por el predominio de X sistema operativo ó Z base de datos que sea significativa en ese momento de la contratación. Parte de la incertidumbre en el desarrollo de los programas de software, radica en el uso eficiente de la base de datos y la interoperabilidad de dicha base con el sistema operativo donde se instala.

Consideramos que las cualificaciones afirmadas por programadores y empleadores, coincide con lo expuesto por los entrevistados, donde señalan que entre los conocimientos mas importantes, no es tanto la licenciatura o maestría, sino como se resuelve el problema, la entrega a tiempo y con calidad acorde a las metas establecidas.

> "el cliente a mí no me va a preguntar si eres licenciado, si tienes una maestría... me va a pedir una aplicación funcional. Entonces yo en cuanto a eso sí me basaría, en que hace *(el programador),* en cómo trabaja bajo presión, como te lo mencioné, la capacidad que tiene para resolver problemas" (programador MIXE, GG3).

---

[77] El sistema operativo permite que muchos usuarios compartan una sola de muchas máquinas sin ser conscientes de ello y bajo medidas de protección de cada usuario en particular. El sistema operativo gestiona y controla incluso cientos de dispositivos de características muy diferentes en tiempo real. No existe una metodología a seguir en la construcción de un sistema operativo. La definición de una estructura por capas ayuda, pero no lo es todo. No existe, al menos hoy día, una teoría establecida que nos asista en la tarea de crear un sistema operativo a partir de una especificación de requisitos. El sistema operativo es el que transforma las ordenes del programa de software desarrollado al lenguaje maquina que interpreta el hardware. Tanenbaum, A. S., Operating Systems, Design and Implementation, 3nd edition, Prentice-Hall, 2006.

[78] Una base de datos o banco de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. En la actualidad, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos tienen formato electrónico, que ofrece un amplio rango de soluciones al problema de almacenar datos. http://es.wikipedia.org/wiki/Bases_de_datos Acceso 15 de octubre de 2007.

Para el experto en programación RS0, lo que cuenta al momento de contratar a un programador, no son las credenciales formales de educación, sino la forma sencilla de resolver problemas, y esto sólo se obtiene a través de las habilidades adquiridas con la experiencia:

> "Pues yo creo que es un conjunto de habilidades y experiencias pero... ¿cómo llamarlo? No es más que la forma de resolver problemas de la manera más sencilla" (Líder TADI, RS0)

El punto de convergencia de la gráfica 19 circunscritos por las primeras dos primeras habilidades con calificaciones mas altas, y el segundo bloque que representa conocimientos especifico en el uso de herramientas que interactúan directamente con la operabilidad del software desarrollado (bases de datos y sistema operativo). Las habilidades y destrezas subjetivas las describimos en el apartado 3.7, destacando en la pagina 95 que en el proceso de trabajo del software se circunscribe a un conjunto de subjetividades inherentes al proceso de trabajo: *subjetividad creativa, subjetividad signica y subjetividad que se objetiva*; así como un alto grado de integración del trabajo social en la pantalla del programador y conformación de un conjunto de *constelación de relaciones subjetivas* que se construyen formal e informalmente; representan intencionalidades y conflictos que pueden ser individuales o colectivos. Subjetividades que influyen y forman parte del conjunto de habilidades y conocimientos citados en la gráfica 19.

El sentido común (*timing cognitivo*) y la comunicación de éste a los otros programadores es importante para resolver acertadamente un problema planteado. Tan importante es la reflexión como la comunicación verbal y escrita (documentación, comentarios al código, etc.) en el desarrollo de un algoritmo porque, no existen recetas o un "único mejor camino" para resolver un problema, ya que las resoluciones tienen varias soluciones. Por ejemplo, si se tiene un sistema desarrollado en un lenguaje de programación X (recordemos que existen cientos de lenguajes de programación), en una base de datos (existen más de 10 bases de datos) y, se detecta un problema relacionado con el hecho que el programa de Software implementado se queda "colgado" no corre eficientemente, es decir, si funciona, pero no de una forma eficiente (suponiendo que no hay problemas de compatibilidad y de diseño grafico), implica una serie de alternativas:

> "el sentido común para poder resolver los problemas de una manera rápida y eficiente. Y la otra seria la facilidad o la habilidad para aprender, ya sea un nuevo lenguaje o aprender una nueva herramienta de desarrollo. Y otra cosa que

también se requeriría... seria la habilidad para comunicarse, para comunicar que es lo que esta haciendo, como lo esta haciendo, para comunicarse con el usuario o con el cliente y poder entender y saber que es lo que requiere, en este caso probablemente se comunique con el con el analista, con el diseñador, ósea dependiendo del rol que trae el programador en el proceso del desarrollo" (Líder MIXE, JG0)

Es importante la comunicación, ya sea oral o escrita, es fundamental en esta industria, por el hecho que las tareas están concatenadas, los módulos que integran el sistema del Software están interrelacionados, de tal forma que el atraso en una tarea o módulo puede implicar que no se entregue a tiempo el proyecto o bien, que una tarea que se podía elaborar en 3 días, requiere 5 o mas días

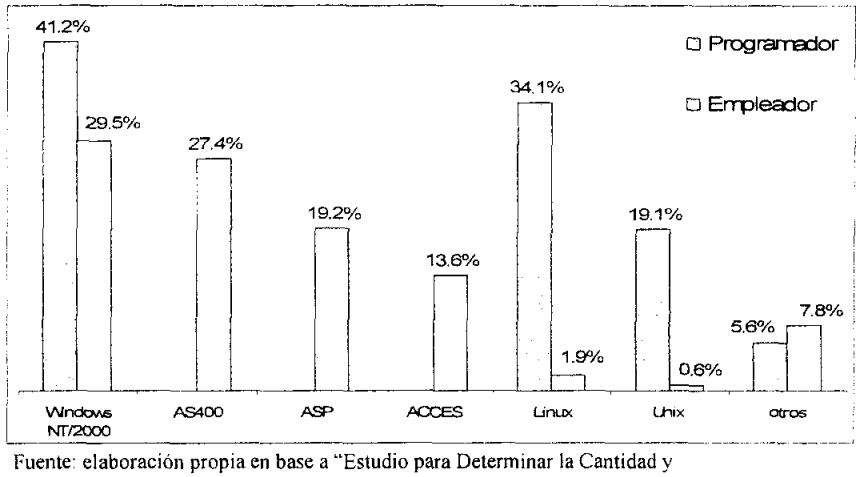
"en un principio te dicen son diez objetivos para diez meses y cuando tú ves esos diez objetivos, te das cuenta de que en el transcurso ya no fueron diez, fueron treinta que se fueron planteando, que te fueron pidiendo, «oye que ahora has esto... oye ahora has lo otro» (...) (Programador MIXE, GG3).

Ahora bien, el conocimiento en el uso de sistemas operativos y bases de datos es complejo de definir, ya que conocer como opera o configura un sistema operativo como UNIX; LINUX; MINIX entre otros; y/o una bases de datos como Sybase, Oracle, Informix, DB2, MySQL, Dbase, VMS, MS SQL Server, SQL Base, Ingres, Progress, ODBC, JDBC, Pervasive, Clipper, Excel, Acces, Mopix, Cobol, entre muchos otros requiere de una capacitación continua, de una intencionalidad e iniciativa personal, que no le cueste a la empresa, sino que forma parte de las cualidades de la oferta de cualidades de los programadores. Mas adelante volveremos sobre este punto, en el hecho que los empleadores no invierten en la capacitación de los programadores, son éstos quienes se auto-capacitan. La acumulación de habilidades, destrezas, conocimientos, prácticas, pericias técnicas en el uso de herramientas como lenguajes de programación, plataformas tecnológicas y bases de datos, conforma un heterogéneo conjunto de talento creativo que es la materia prima de este tipo de proceso de trabajo. Por ejemplo, la pericia técnica en el uso de bases de datos es un conjunto sustancial en el proceso de desarrollo de un sistema informático, ya que es donde se almacenan los datos de la empresa, es una especie de matriz de datos, donde se realizan los "movimientos" reales de los datos que son ingresados al sistema; son herramientas que coadyuvan en la eficiencia y optimización de los recursos informáticos del Software. Ahora bien, esta polémica en torno al conjunto de habilidades y conocimientos en torno al uso de

herramientas tecnológicas como bases de datos, lenguajes de programación, plataformas tecnológicas, etc. se complejiza porque existen dos modos de desarrollar software, el modo privado y el modo libre. El primero se explica por el uso y pago de licencias, venta y desarrollo de software desde cero. El software libre[79] está exento en el pago de licencias, la venta y desarrollo de software no inicia de cero, recomienza a partir del software disponible en las comunidades de software libre (Debian, Ubuntu, Gnome, etc.). Este doble camino en el modo de desarrollo de software, forma parte del debate en torno a las habilidades y conocimientos de los programadores de software en los próximos años.

Gráfica 20

Bases de datos más importantes en los próximos tres años
para el puesto de Programador de Sistemas en México. 2004



Fuente: elaboración propia en base a "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México" Secretaría de Economía-UAM. Noviembre 2004

Según vemos en la gráfica 20, más de un tercio de los programadores (34.1%) considera que la base de datos en sistemas libres como Linux[80] será importante en los próximos tres años, dato

[79] El software libre suele estar disponible gratuitamente (pero no hay que asociar software libre a software gratuito), o a precio del costo de la distribución de éste, sin embargo no es obligatorio que sea así y, aunque conserve su carácter de libre, puede ser vendido comercialmente. Análogamente, el software gratis o gratuito (denominado usualmente freeware) incluye en algunas ocasiones el código fuente; sin embargo, este tipo de software no es libre en el mismo sentido que el software libre, a menos que se garanticen los derechos de modificación y redistribución de dichas versiones modificadas del programa. No debe confundirse "software libre" con software de dominio público. Éste último es aquél por el que no es necesario solicitar ninguna licencia y cuyos derechos de explotación están disponibles en la red. Cualquiera puede hacer uso de él, con fines legales y consignando su autoría original. http://es.wikipedia.org/wiki/Software_libre Acceso 15 de octubre de 2007.

[80] Linux es la denominación de un sistema operativo tipo-Unix y el nombre de un núcleo. Es uno de los paradigmas más prominentes del software libre y del desarrollo del código abierto, cuyo código fuente está disponible públicamente, para que cualquier persona pueda libremente usarlo, estudiarlo, redistribuirlo y, con los conocimientos informáticos adecuados, modificarlo. Los primeros sistemas Linux se originaron en 1992, al combinar utilidades de sistema y librerías del proyecto GNU con el núcleo Linux, completando un sistema

que contrasta con los empleadores, quienes señalan toda una gama en bases de datos, desde las clásicas Windows NT/2000 (29.5%); AS400 (27.4%); SAP (19.2%); ACCES (13.6%), entre otros. Nuevamente, aquí se plantea una divergencia entre lo que estiman los programadores y los empleadores. Las diferencias son fundamentales, los programadores tiene claro que el mercado del Software esta dividido entre aquellos que pagan licencias de uso (Software comercial), por esto 41.2% estima, que la base de dato que mas se utilizará es Windows, seguida del 19.1% con Unix. Para el otro mercado, el del Software libre, también lo tienen claro el 34.1% de los programadores consideran que será Linux la base de datos en los próximos años que predominará en el mercado. Por el contrario los empleadores señalan otras bases de datos dispersas, pero a Linux prácticamente no lo mencionan, en cambio las bases de datos más demandadas, después de Windows, serían AS400, ASP y ACCES. La característica de estas bases de datos es que se debe pagar licencias de uso, diferencia que plantea un debate incipiente en el orden académico y profesional en torno a la incidencia del modo de desarrollo de Software libre y el modo de desarrollo de software de patente. Debate que no abordaremos en la presente investigación, sin embargo dejamos aquí planteado, que entre las expectativas de los programadores está el instruirse en el uso y desarrollo de plataformas de software libre, como Linux, lo cual consideran una ventaja competitiva en los próximos años.

6.5.- Habilidades, destrezas y cualificaciones en el trabajo cognitivo

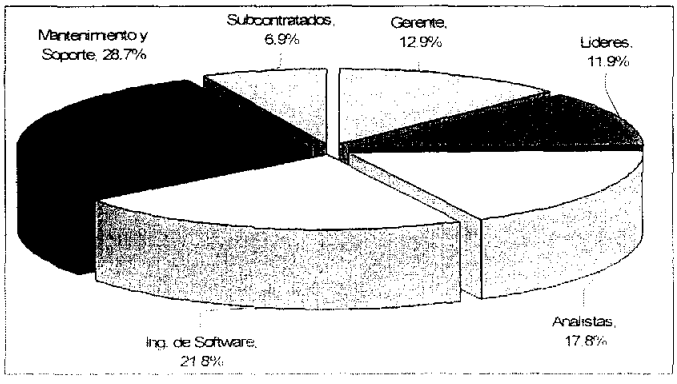
A continuación abordaremos los temas relativos a las ocupaciones y calificaciones, donde observamos que los resultados correspondientes a la desagregación por tipo de empleados técnicos muestran que 28.7% del total de los empleados de las empresas encuestadas se dedican a actividades de mantenimiento y soporte, seguido de ingenieros en software con 21.8% y analistas con 17.8%. En términos generales el empleo de gerentes y líderes del proyecto es consistente con promedios de 12% a 13%, respectivamente. Cabe destacar que el 28.7% en mantenimiento se puede explicar a la existencia de software creado por terceros que implica contratos asociados de mantenimiento o de actualización, lo que hace suponer que las

---

también conocido como GNU/Linux[2] . Desde finales de 2000 Linux ha obtenido un aumento en el apoyo de diversas empresas multinacionales del mundo de la informática, tales como IBM, Sun Microsystems, Hewlett-Packard y Novell. Actualmente Linux es comercializado en computadores de escritorio y portátiles por Dell y Lenovo, además hay un grupo numeroso de compañías establecidas en Taiwán que planean hacer lo propio. http://es.wikipedia.org/wiki/Linux Acceso 15 de octubre de 2007.

empresas de software están funcionando también como centros de asesoría y mantenimiento de software, mas que desarrollo de nuevas aplicaciones (ver gráfica 21).

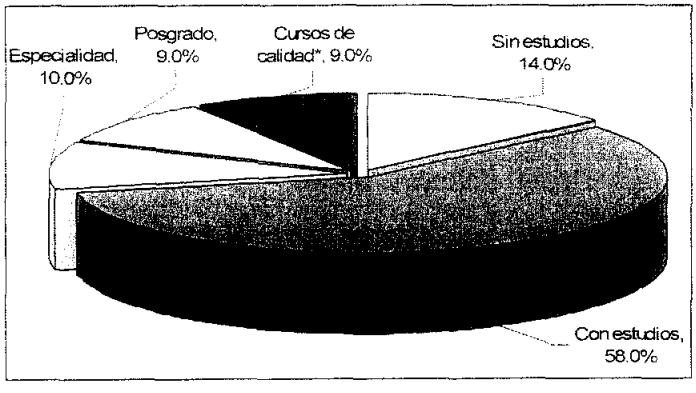
Gráfica 21
Distribución porcentual de las ocupaciones técnicas en las empresas desarrolladoras de software en el valle de México. 2004



Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana.

Los resultados correspondientes a calificaciones formales en los empleados técnicos muestran que 58% tienen estudios profesionales y sólo 14% no posee estudios universitarios, datos que confirman lo señalado en el apartado anterior, que se trata de un trabajo con alto índice de profesionalización (ver gráfica 22).

Gráfica 22
Calificaciones formales entre el personal técnico de las empresas desarrolladoras de software en el Valle de México. 2004



Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana a encargo de la Secretaría de Economía

Cabe destacar el bajo nivel de capacitación en cursos para mejorar la calidad (sólo 6.9%) ya sea en ISO 9000, CMM, PSP/TSP, entre otro; así mismo los bajos cursos en

especialidad y postgrado, lo que nos permite deducir que la capacitación mediante cursos formales, no es un factor importante en la contratación de personal o bien que las empresas no están capacitando a su personal técnico en los cursos señalados ya sea en ISO 9000, CMM, PSP/TSP (ver gráficas 22). Puede existir un conjunto de capacitaciones no formales que los programadores de software realizan por su cuenta, ya sea a través de inscribirse en paginas Web con acceso gratuito, o bien en comunidades de programadores donde intercambian documentos de cursos o comentarios con respecto a como solucionar determinado tipo de problemas.

Una de las herramientas de programación mas heterogéneas son los lenguajes de programación, en la actualidad se contabiliza mas de dos mil lenguajes, destacando la denominada "software wars"[81] entre los lenguajes para software comercial y software libre.
Los lenguajes de programación más utilizados muestran que la mayoría de las empresas utilizan principalmente los siguientes lenguajes: Visual Basic, SQL Server, HTML, SQL, Java, XML y Java Script. Cabe señalar que el lenguaje de programación Sybase tiene el mayor porcentaje de uso con 79% en las empresas encuestadas en la ZMDF; así mismo sobre sale una tendencia de la tecnología Web, y por otra parte muestran la preferencia de las empresas por una tecnología accesible y más económica en su mantenimiento y soporte a sistemas, como las bases de datos de software libre: MySQL, entro otros (ver cuadro 21).

Las plataformas tecnológicas más utilizadas por las empresas encuestadas se deben en gran medida a la trayectoria comercial de las empresas proveedoras, así como en el uso de estándares para sistemas abiertos y multiplataforma. En el caso particular en el uso Linux se debe a su característica de software libre. Los resultados correspondientes a las bases de datos más utilizadas por las empresas muestran que las principales bases de datos son MS SQL Server, Oracle, Acces, ODBC y MySQL. El caso de los manejadores de bases de datos es muy similar a la situación de las plataformas tecnológicas, donde las preferencias de las empresas se enmarcan en estrategias comerciales y nichos de mercado de las principales empresas proveedoras, distinguiéndose MySQL que aparte de ser software libre esta orientado a Web.

[81] La guerra del software es una lucha ideológica y económica en torno a que modo de desarrollo es el que mejor posee un control en la "aflicción del software". http://etsiit.ugr.es/alumnos/mu01/guerraSoftware.html acceso 15.04.07

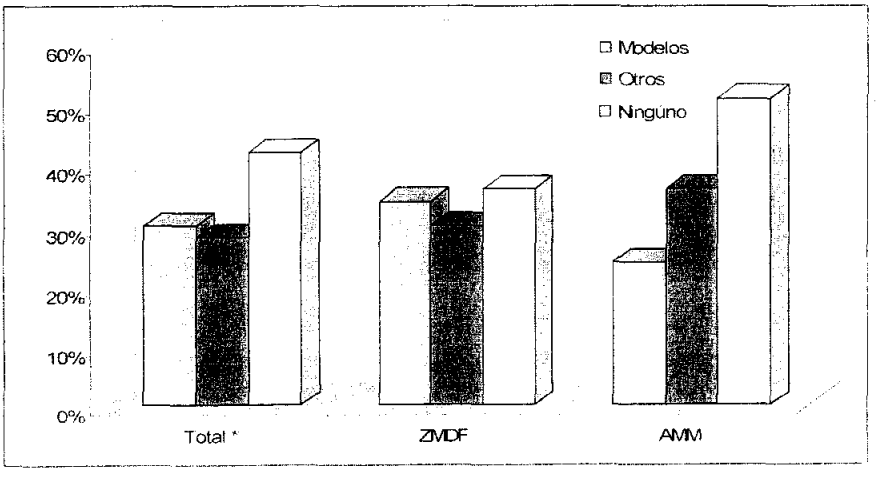
Cuadro 21
Principales tecnologías empleadas en las empresas de software en el valle de México. 2004

Lenguajes de programación	Plataformas tecnológicas	Bases de datos
Visual Basic; b/Com; C; C++; Cobol; Xbase Power Builder; SQL Server; Servd WWW; Delphi; HTML; XML; Java; Java Script; Active X; Latte; Visual Café; SQL; RPG; Clipper; Progress; Oracle; Basic; Fox Pro; Dbase; Sybase; Informix; DB2; ERP; Macro media; Firmware; Macro Excel; Smallt alk; Otros.	OS/2; Windows; Windows CE; Palm OS; Unix Unix base OS; Linux; Apple Mac OS X 10.1.5; Apple Mac OS X 9.2.2, Sistema Mini computer; Sistema Mainframe; Sistema Operat (RTOS); Otros.	Sybase; Oracle; Informix; DB2; MySQL; Dbase; VMS; MS SQL Server; SQL Base; Ingres; Progress; ODBC; JDBC; Pervasive; Clipper; Excel; Acces; Mopix; Cobol; Otros.

Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana a encargo de la Secretaría de Economía

El resultado correspondiente a la implementación formal de metodologías de desarrollo y calidad en el proceso de generar el software a la que declararon las empresas encuestadas, ya sea que estén certificadas o en proceso de obtener la certificación (implementación), 30% del total de empresas se apegan a los estándares internacionales ISO 9000 o CMM/CMMI, un 28% de las empresas encuestadas reportó que su metodología no corresponde a los estándares indicados (sino a otra no especificada) y 42% de las empresas encuestadas no utilizan ninguna metodología (ver gráfica 23)

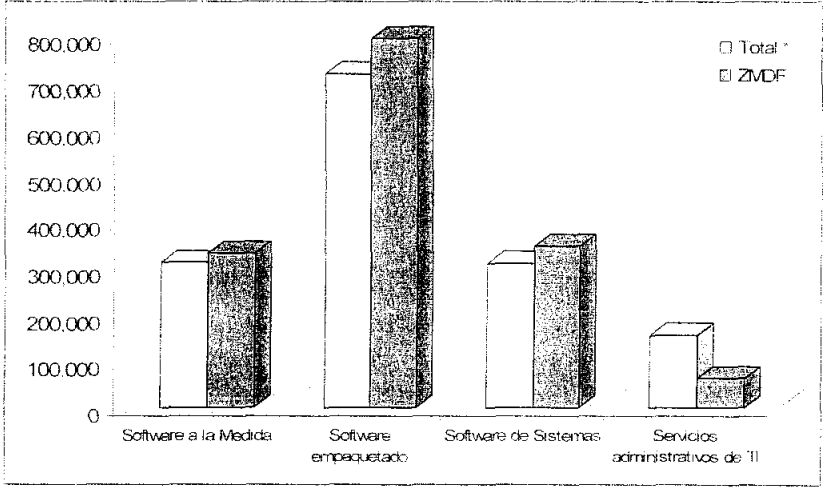
Gráfica 23
Porcentaje de empresas que implementan alguna metodología de desarrollo en las empresas de software en el valle de México. 2004



Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana a encargo de la Secretaría de Economía. * Suma del área metropolitana de Nuevo León y la ZMDF.

Cuando se les preguntó en que proceso se estarían certificando las empresas reportaron que cuentan con certificado o en proceso de evaluación, sólo 22 empresas declararon tener o estar en proceso de evaluación; correspondiendo 14 casos a la ZMDF y 8 para el área metropolitana de Monterrey. Para el caso de la ZMDF, significa que sólo 18% declaro estar certificadas, de las cuales 3 están certificadas en ISO-9000; 3 en CMM/CMMI (nivel 2 en CMMI); y 8 mas en otros tipos de procesos no especificados. La implementación insuficiente de programas de calidad del tipo CMM, puede estar relacionado no sólo con el ato costo, sino también con la percepción de que dichos proceso aplican para grandes empresas que poseen una estructura rígida en la división del trabajo. La falta de programas de calidad, también está relacionada con la productividad, donde el aporte de un empleado al producto anual de la empresa, expresado en términos monetarios. El cálculo de la productividad se obtiene multiplicando el número de empresas (por tamaño de empresa) por el promedio del rango de ventas correspondiente. Este importe se divide entre el número de empleados estimados para cada uno de los rangos de ventas, cabe señalar que el cuadro presentado solo representa la productividad con base en el promedio de empleados de acuerdo al tamaño de empresa.

Gráfica 24
Productividad de los empleados en las empresas desarrolladoras de software en el valle de México. 2004



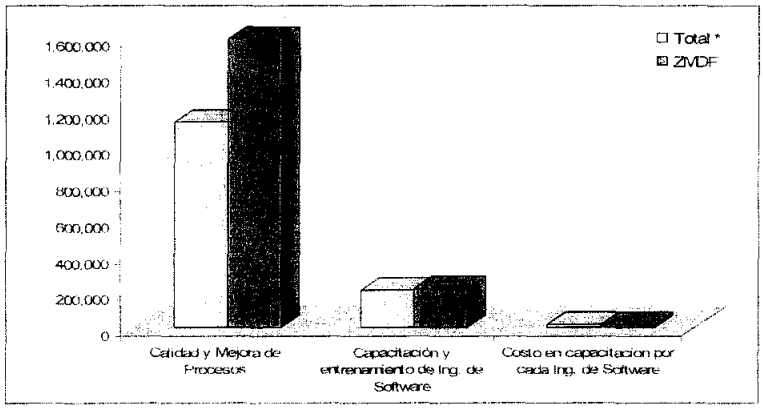
Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana a encargo de la Secretaría de Economía. * Suma del área metropolitana de Nuevo León y la ZMDF.

La inversión promedio en actividades de calidad y mejora de procesos de desarrollo de software es alrededor de 1.1 millones de pesos por empresa, destacándose que las empresas grandes gastan el doble del promedio. En lo relativo a la inversión en capacitación y

entrenamiento de ingenieros de software el gasto anual promedio es alrededor de $200,000 pesos y el costo promedio por ingeniero de software es de 16,185 pesos, destacándose el hecho de que las empresas grandes invierten menos en capacitación y entrenamiento cuando esta se mide por ingeniero de software. En términos generales, las cifras de las empresas ubicadas en la ZMDF son más favorables o superiores a las reportadas con respecto a Nuevo León –por citar un ejemplo-

Gráfica 25

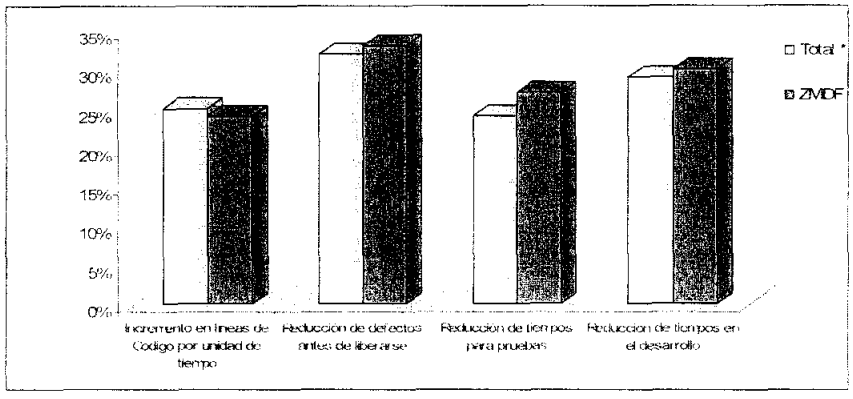
Inversión total en capacitación y mejora de los procesos y por costo invertido en ingeniero de software en las empresas de software en el valle de México. 2004



Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana a encargo de la Secretaría de Economía. * Suma del área metropolitana de Nuevo León y la ZMDF.

La inversión en capacitación y entrenamiento por ingeniero de Software, supone una mayor eficiencia en los procesos, consiste en hacer más y mejor con los mismos recursos, midiendo los efectos de los recursos en los resultados. En este sentido, las empresas calcularon resultados eficientes en los procesos. En términos generales los porcentajes señalados en la gráfica miden la eficiencia de los recursos humanos y del modelo de organización de la empresa, observándose resultados favorables, en promedio un 25% de eficiencia en los indicadores. Estos indicadores suponen la satisfacción del cliente y, puede medirse como el grado de pertinencia en que recibe el producto o servicio y se mide a través del Coeficiente de Satisfacción el cual se obtiene por medio de encuestas aplicadas al usuario de los servicios. Para el estudio en cuestión se cita un 30% en la reducción de los reportes de defectos post liberación del Software, es decir de los errores o inconsistencias en el software ya implementado en la empresa del cliente. Porcentaje, que esta ligado a la calificación de 9.3 de satisfacción del cliente.

Gráfica 26
Eficiencia productiva medida en un mejoramiento porcentual con respecto al año en las empresas desarrolladoras de software en el valle de México. 2004



Fuente: PROSOFT 2004. Estudio del nivel de madurez y capacidad de procesos de la industria de tecnologías de información. Universidad Autónoma Metropolitana a encargo de la Secretaría de Economía. * Suma del área metropolitana de Nuevo León y la ZMDF.

Como dimos cuenta en el presente apartado destaca a nivel nacional la indefinición de los nombres de las profesiones a nivel académico, así como el debate entre los programadores como oferentes y los empleadores que se centra en la divergencia en torno a las cualificaciones y conocimientos que deben poseer los programadores, así como el perfil técnico o profesional de la ocupación. En el apartado correspondiente al valle de México, dimos cuenta de que es un conjunto de microempresas, con equipos de trabajo pequeños que no aplican métodos de calidad, con productividad es baja. Estos datos señalados, nos permiten considerar dos aspectos importantes: el reconocimiento social de la profesión y la conformación de la ocupación en el mercado de trabajo del software. La primera está dada por los debates señalados en torno a las consideraciones de capacitación, habilidades y destrezas, competencias y conocimientos que deben poseer los programadores y los señalamientos divergentes que hacen los empleadores. La construcción social de la ocupación, está mas relacionada con lo joven de los programadores, lo nuevo de esta ocupación que aun hoy en día no cuajan las metodologías y procesos propuesto por la ingeniera del software, o bien, se esta ante un proceso de trabajo que aún continua embebido en una constelación de subjetividades del proceso de trabajo concentradas en la creatividad del programador, pero fuertemente contextualizada por un nuevo tipo de interacciones sociales ya sea a través de colectividades, comunidades virtuales, prácticas sociales, interacciones interactivas del tipo pantalla a pantalla.

Capitulo VII

Contextos externos y procesos internos que configuran flujos de Aprendizaje en el proceso de trabajo cognitivo en el valle de México

7.1.- Introducción

Con respecto a la persistencia del riesgo e incertidumbre en el desarrollo el software a pesar de la existencia de la Ingeniería del Software, se ha integrado una corriente de académicos y empresarios europeos (esto no quiere decir que no haya estadounidenses) que critican el método general de la Ingeniería del Software, en el sentido que esta se ha centrado casi exclusivamente en los atributos y particularidades del software, y ha descuidado al usuario final, se ha centrado más en temas relacionados con el sistema de operación interna, con el rendimiento y fiabilidad del sistema; es decir en aquellos factores que pueden ser medidos objetivamente como: numero de errores por cada numero de líneas de programación; tiempos de respuesta y probabilidades de error por cada determinado numero de código, etc.; así, se progresó en aspectos "duros", deterministas, verticales que intentan solucionar la calidad del proceso de trabajo: métodos y herramientas de análisis, diseño, codificación y pruebas; revisiones técnicas formales; estrategia de pruebas multi-escalada; control de documentación de software y procedimientos de ajuste a los estándares de desarrollo de software; mecanismos de medida y de información.

En la presente investigación se construyó una tipología de las empresas entrevistadas y como emprenden con estrategias diferenciadas los espacios de producción en el proceso de trabajo: conceptualización/formalización de los requisitos del programa a desarrollar y el otro espacio el del procesamiento de datos e implementación del software acabado. Estos espacios de desarrollo no significan una estricta división social del trabajo, por el contrario, están yuxtapuestas, las fronteras son difusas y se intercalan y reconfiguran conforme se desarrolla el proyecto de software. Aún que ya se haya entregado el software, la esfera de producción continúa en la de implementación en casa del usuario final.

7.2.- Escenarios externos que presionan el proceso de trabajo

La organización industrial de las empresas estudiadas en el Valle de México, es heterogénea, atomizada; destacando seis de las 17 empresas en estudio, las cuales presentaron proyectos para formalizar la inserción de la industria del software del Valle de México en el grupo de

empresas que participan en el programa gubernamental de la industria del software (PROSOFT 2006). De las entrevistas se derivo una serie de singularidades divergentes y heterogéneas que nos permitió construir una serie de características generales de las empresas entrevistadas, dando lugar a una tipología por grupo de empresas (ver cuadro 22).

Cuadro 22

Tipología por tamaño de empresa que desarrolla software a la medida en el valle de México. 2005

Empresas*	Tamaño	Características
Empresas tipo 1 Colectividad sistémica		
ELLA	Pequeña	Esta inmersa en un periodo de reestructuración, esta saliendo de saldos rojos de 1.5 millones de pesos. Hizo convenios con Universidades. Es subsidiaria de MIXE.
DYAA	Pequeña	Empresa que ofrecía un servicio a la medida, conforme el tiempo consolido el servicio en un Producto que ofrece a más de 800 clientes. Hoy sólo contrata a 4 Programadores, de 20 que comprendia el proceso de trabajo.
MIXE	Mediana	Empresa que impulso la creación de una INTEGRADORA, y un Consorcio de Empresas y Universidades. Tiene más de 100 programadores.
DASA	Mediana	Empresa que se especializa en una serie de Productos, fue adquirida por uno de sus clientes (empresa global, de origen mexicano) y amplio su oferta a Servicios.
SOCU	Grande	Empresa que diversifica su oferta de actividades informáticas, áreas de capacitación, promueve el uso de metodologías de calidad. Emplea a más de 200 programadores.
CISE	Grande	Diversifica su oferta, asesoría técnica, asociaciones con empresas globales. Emplea a más de 300 programadores.
Empresas tipo 2 Crisis y reestructuración interna		
TADI	Micro	Empresa que se ha reestructurado tres veces, tras la crisis de 1994, cerró las oficinas, vendió activos y está operando al mínimo, no le interesa crecer. Expectativas limitadas en el mediano plazo. De contratar poco más 30 programadores, hoy sólo contrata a 3 programadores.
SITA	Pequeña	Esta inmersa en un periodo de reestructuración, esta saliendo de los saldos rojos. Esta reorientándose hacia el mercado de Asesorías en TI y desarrollo de Software.
MEKI	Pequeña	Empresa que está en pleno proceso de recuperación tras la crisis de 1994, con perdidas de 1.5 millones de dólares. De un plan internacional, con más de 60 programadores, hoy no contrata más de 10 programadores. De ofrecer toda una gama de servicios, hoy sólo se especializa en un nicho determinado. No existen expectativas de reorganizarse para abarcar un mayor número de servicios.
CATI	Pequeña	Empresa que al momento de ser entrevistada tenia más de 10 Programadores, sin embargo estaba en pleno proceso de cerrar el área de programación, porque era muy caro mantener el área. Hoy subcontrata, desarrolla servicios con los clientes que ya tienen en mayor parte y, contrata vía outsorurcing a programadores o Freiancee para el desarrollo de software.
HUTI	Pequeña	Empresa que tenía más de 10 Programadores, cerró toda el área, hoy el dueño se dedica a promover servicios especializados de TI en el extranjero.
Empresas tipo 3 Comunidades de aprendizaje virtual (*open source*)		
LIES	Micro	Empresa en la cual el Líder y socio es experto en SWL, ex académico, contrata a dos programadores expertos, ofrecen servicios. Tienen clientes importantes.
WALS	Micro	El Líder y Socio de la empresa, ofrece una serie de asesorías e implementación de Sistemas informáticos en SWL. Contratan ocasionalmente a programadores vía freelance. Tienen poco más de 10 años de experiencia.
BOCS	Micro	El Líder y dueño de la empresa, posee poco más de 30 años de experiencia. Tiene una amplia gama de grandes clientes, contrata de 3 a 4 programadores.
UFOS	Micro	El Líder y Dueño de la empresa, tiene más de 25 años de experiencia en el desarrollo de SWC y SWL. Ofrece distintos servicios a sus clientes. Contrata a 4 programadores.
BICS	Micro	Programador experto, con más de 20 años de experiencia, es solicitado por varias empresas de SWC y SWL para que soluciones problemas.
DINS	Micro	Programador en SWL que promueve las METODOLOGIAS ORIENTADAS A ASPECTOS con el fin de organizar el proceso de desarrollo de un SW.

Elaboración en base a las entrevistas. En el Capitulo Metodológico se amplía la información de las empresas entrevistadas.

* Nomenclatura utilizada para referirnos a las empresas. En el apartado Metodológico se explica el significado de estos acrónimos.

Una vez que consideramos el tipo de relaciones laborales, organización interna y externa de la empresa, relaciones entre agentes sociales, técnicos, académicos, nótese que las empresas no se agruparon por tamaño, sino que predomino el criterio del tipo de interacciones sociales que se establecían ínter empresarial y, en su entorno social, técnico y productivo; lo cual permitió construir cierto grado de homogeneidad hacia el interior de las tipologías propuestas (ver cuadro 23).

Cuadro 23
Tipología por características homogéneas entre las empresas que desarrollan software a la medida en el Valle de México. 2005

Empresas tipo 1 ***Colectividad sistémica***	**Empresas tipo 2** ***Crisis y reestructuración interna***	**Empresas tipo 3** ***Comunidades de aprendizaje virtual (open source)***
Modo de desarrollo de software privado*	Modo desarrollo de software privado*	Modo de desarrollo de software libre*
Ofrecen servicios de asesoría y consultoría a los clientes.	Ofrecen desarrollo a la medida y medianamente o nula consultoría.	Ofrecen adaptación a la medida y medianamente o nula consultoría.
Desarrollo de proyectos sencillos a complejos,	Desarrollo de proyectos menos que complejos,	Desarrollo de proyectos menos que complejos
Relación estrecha del cliente con el gerente ó *líder de proyecto* y responsable del proyecto.	Relación del cliente con el *programador responsable* del desarrollo del proyecto.	Menor interacción del cliente con el *programador responsable* del proyecto y más con el gerente.
Forman parte de Empresas Integradoras**	No Forman parte de Empresas Integradoras**	No Forman parte de Empresas Integradoras**
Son partner´s de empresas globales (IBM, ORACLE, etc.)	No tienen asociación con empresas globales.	No tienen asociación con empresas globales.
Aplicación formal de principios de la Ingeniería del Software.	No aplican metodologías estándar de la Ingeniería del Software.	No aplican metodologías estándar de la Ingeniería del Software.
Buscan certificar sus procesos en alguna norma de calidad.	No tienen entre sus planes certificar sus procesos.	No tienen entre sus planes certificar sus procesos.
Menor flexibilidad numérica	Mayor flexibilidad numérica.	Mayor flexibilidad numérica.
Estancias académicas en las empresas desarrolladoras.	No hay convenios con instituciones académicas.	No hay convenios con instituciones académicas.
Formalidad en equipos de trabajo	Informalidad en los equipes de tabajo.	Virtualización de los equipos ee Trabajo.
Definición/formalidad en jerarquías laborales	Informalidad en las jerarquías laborales	Informalidad en las jerarquías laborales
Funcionalidad/especificidad en los puestos de trabajo.	Ambigüedad en la funcionalidad de los puestos de trabajo	Ambigüedad en la funcionalidad de los puestos de trabajo
Mercado local, nacional Y, de exportación.	Mercado local y nacional	Mercado local y nacional.
Pertenecen a una asociación de la industria del software.	No pertenecen a una asociación de la industria del software.	Si pertenecen a una asociación de la industria del software.

* Software Privado: es aquel basado en pago de patentes. Lo hemos citado como Modo de Desarrollo de Software Privado (SWP.
* Software Libre: es aquel que NO paga patentes. Lo hemos citado como Modo de Desarrollo de Software Libre (SWL).
** Empresas Integradoras: Son aquellas empresas que pertenecen a un CLUSTER que agrupa distintas empresas. Este consorcio se presenta a su vez como una "macro empresa" que participa en licitaciones de proyectos complejos. Elaboración en base a las entrevistas.

El cuadro 22 y 23 se sintetizan las características de las empresas entrevistadas, como ya explicamos en el capitulo 5 y 6, la industria del software es de carácter incipiente, con pocos años de generar relaciones sociales de producción, con proximidades geográficas, organizacionales e institucionales limitadas a espacios de interacciones incipientes, que como daremos cuenta en los siguientes párrafos, se sucedieron al menos tres tipos de configuraciones en la industria del software: i) aquellas empresas que identificamos del tipo 1, que se caracterizan por cierto grado de dinamismo, competitividad y construcción de una configuración socio-institucional definida por una serie de *interacciones sociales* y una colectividad que genera *flujos de aprendizaje* en torno a la industria del software; ii) empresas del tipo 2 que es un conjunto de unidades empresariales que están saliendo de una crisis económica, están inmersas en un proceso de reestructuración interna y en exploración de un nicho tecnológico en el cual especializarse; iii) Las empresas del tipo 3, se les identifica en torno al modo de desarrollo de software basado en algoritmos de código libre (*open source*), están vinculadas a comunidades virtuales de programación y, el desarrollo del software tiene una fuerte dependencia de la materia prima disponible en comunidades virtuales ancladas en Internet.

7.2.1.- Construcción social de una *colectividad sistémica* en las empresas tipo 1

Como ya explicamos en el capitulo 2 y 4, la *aflicción del software* en el proceso de trabajo posee una serie de aristas estructurales que presionan a los agentes a nivel local e influyen, mas no determinan la acción social, sino que hay un espacio en el cual cuajan los intereses, las necesidades, las demandas, las eventualidades y contradicciones, -lo cual no quiere decir que dejen de existir o que se hayan solucionado- por el contrario, consideramos que se construyen una serie de significados y coincidencias, de intereses, a través de *interacciones sociales*, que van condensándose, en un contexto de incertidumbre, resistencias, consensos, negociaciones, arreglos sociales, conflictos y boicot, etc.; sin embargo, coexiste socialmente una serie de acuerdos tácitos, de representaciones y sentidos objetivos y subjetivos entre los agentes institucionales que convergen en una coyuntura determinada; como son los agentes públicos y privados del sistema académico, el sistema empresarial y, estrategias de políticas gubernamentales. Por ejemplo los programas financieros, leyes para la inversión, reglamentos de apoyo y gestión institucional; además de la coyuntura de necesidades en el desarrollo de

infraestructura, servicios y productos que favorezcan el entramado industrial, etc.; así como otro tipo de configuraciones que presionan y favorecen a los agentes a interaccionar e intervenir activamente en forma de alianzas, asociaciones y colaboraciones no necesariamente planificadas o previamente diseñadas, sino construidas en el proceso mismo de las contingencias; arregladas socialmente como formas de intervención de los distintos agentes que convergen en generar una serie de acciones inmediatas, que resultan no de un *individualismo metodológico* sino de una serie de *interacciones sociales*, sean a través de una configuración tipo red, comunidad ó de una colectividad.

Las diferentas *prácticas sociales* que se suceden a través de la interacción del tipo coyuntural dan lugar a distintos *flujos de aprendizaje,* que son generados en un contexto de colectividad que facilita la circulación del conocimiento tácito y *prácticas sociales* embebidas en el *saber hacer* cotidiano y constituidas como *poder* (conocimiento tácito, explicito y procesos de aprendizaje); dichos *flujos de aprendizaje* no son espontáneos o diseñados previamente en una red o en una suerte de maquinación gerencial; se configuran en contextos coyunturales mediante procesos contingentes, intemporales, laxos, con resonancias de conflictos y resistencias, de arreglos sociales y acuerdos implícitos, entre claroscuros no preestablecidos, no definidos, que se construyen a través de las distintas *interacciones sociales* que se suceden en diferenciados grados y magnitudes entre los agentes socio-institucionales.

Para Wenger (2001) el aprendizaje social es aquel que, por un lado, se desarrolla dentro de estructuras sociales, y por el otro, es generador de estas mismas estructuras o configuraciones sociales (aquí le llamaremos *prácticas sociales de aprendizaje*). Sin embargo, este autor también sostiene que la generación de contextos de aprendizajes sociales no están sujetos a un diseño previo del tipo racional, ni depende de él. El aprendizaje social no se puede diseñar, solo se puede facilitar o frustrar (Wenger, 2001: 273); lo que sí se reconoce es la importancia de una arquitectura, de una infraestructura para el desarrollo de este aprendizaje, es decir, el establecimiento de ciertas condiciones que resultan necesarias para facilitar el aprendizaje social y con ello, se puede decir que, también facilitar la *interacción social* que da lugar a distintas *prácticas sociales de aprendizaje.*

La configuración de *prácticas sociales de aprendizaje* es consecución de una serie de *interacciones sociales* que pueden ser creativas, de resistencia, de negociación, de conflicto, de consenso entre distintos agentes institucionales que contribuyen a una articulación entre

sectores, como son el económico y productivo, tecnológico y educativo, seguido de políticas públicas; además exige no sólo que se considere el entramado social y organizativo inter-empresarial, se requiere de la convergencia de representaciones y sentidos de los actores individuales; todo ello en un proceso de *interacción social* de las relaciones sociales que resulten en distintas *prácticas sociales*. Algunos investigadores han estudiado las diferentes prácticas sociales en la generación de conocimiento y aprendizaje; como Leydesdorff (1997); Etzkowitz, y Webster (1998); Casas (1998 y 2001); Luna (2001 y 2003); López (2001);

Consideramos que las distintas *prácticas sociales de aprendizaje* ya sea entre instituciones académicas, sector productivo empresarial o bien entre organizaciones sin fines de lucro, como centros de investigación y universidades, entre otros agentes, se han convertido en un importante núcleo de interés para las políticas gubernamentales, económicas e industriales de los países interesados en generar espacios de conocimiento, que para el caso de México está el PROSOFT, la ley de Ciencia y Tecnología del CONACYT, los apoyos financieros de la Secretaría de Economía, entre otros.

En México de pueden identificar distintas *prácticas sociales de aprendizaje* que se conformaron en las regiones de economía digital, señaladas por Ruiz (2004) para el caso de la Zona Metropolitana de Nuevo León, constituyeron diversos procesos de *interacción social* entre agentes a nivel local, meso y macro; tal es el caso del programa *Monterrey ciudad internacional del conocimiento* que se compone a nivel local por poco más de cien empresas seguidoras y poco más de ocho empresas lideres en el desarrollo de software, un amplio sistema de universidades publicas y privadas, asociaciones especializadas y dirigencias empresariales y gubernamentales. A nivel meso, destacan las políticas federales de promoción financiera y exención de impuestos, centros de investigación, participación de asociaciones nacionales, entre otras; a nivel internacional, se distinguen las alianzas con empresas globales, como Microsoft, SAP, Oracle, etc.; se suman los convenios con los consorcios internacionales de promoción e investigación; entre otras acciones institucionales orientadas al comercio exterior. Sin embargo, estas prácticas sociales de aprendizaje no son generalizables a todo el territorio nacional, ya que no todas las regiones cuentan con un desarrollo económico estable ni con el grado y magnitud de interacciones sociales de las relaciones sociales de los sistemas regionales de innovación articulados de Nuevo León (Ruiz, 2004; Cimoli, 2000). En este contexto, además de considerar las medidas institucionales que se promueven y, aquéllas de

carácter local desprendidas de políticas nacionales e internacionales, es importante comprender en parte el éxito, fracaso y/o alcance de las *prácticas sociales de aprendizaje* entre actores a nivel micro; es decir, en el flujo del proceso de trabajo de creación del software en el que el actor principal es el programador, aunque interacciona con su grupo de trabajo y en el caso del software a la medida con el cliente. En este nivel micro también se puede hablar de flujos y prácticas sociales de aprendizaje.

Sin embargo, para explicar el sistema colectivo de *prácticas sociales de aprendizaje* en las empresas tipo 1, es necesario tener en cuenta los distintos enfoques y metodologías de análisis para el estudio de los sectores de la industria; por ejemplo las propuestas de políticas públicas, programas de ciencia y tecnología; y la intervención del sector académico; que por citar algunos autores que ya han dado cuenta de estas temáticas, destacan Hedstrom y Swedberg (1994); Senker (1998); Pérez (1996); Peters, Groenewegen y Fielbelkorn (1998); Cimoli (2000); Villavicencio (2002); Casalet (2003); Jaso, (2003); Vera-Cruz (2003); entre otras investigaciones que han propuesto el análisis de redes de las interacciones entre academia y empresa; o bien el estudio desde las perspectivas sociales, como Casas (2001) y Luna (2003); también el análisis estructural de las redes como Malerba y Orsenigo (1996) y las propuestas de análisis sistémico de las redes, como Cimoli (2000). El enfoque de redes es uno de los más estudiados, contrastando con el más reciente análisis, el de *comunidades en práctica* de Wenger (2001); así como el de Amin y Cohendet (2004).

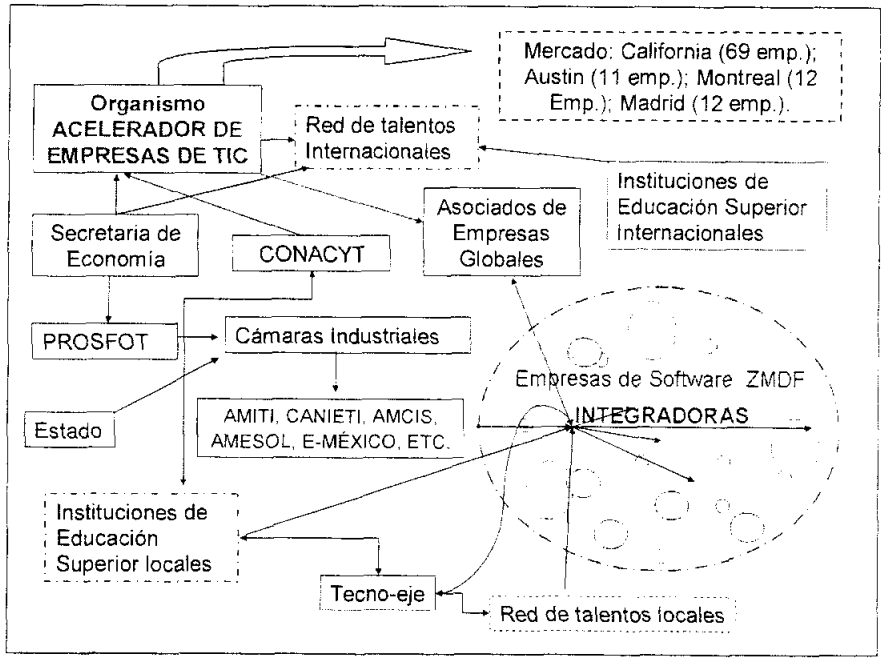
Éstos análisis se suman a otros paradigmas, como el de *triple hélice* de Etzkowitz y Webster (1998), que examinan las distintas *prácticas sociales de aprendizaje* entre el sistema educativo, sector productivo e innovación como un modelo helíptico (hélices girando); modelo que no dista mucho del *Modo 2* de producción de conocimiento de Gibbons (1994), donde las acciones de los agentes, llámense empresa, gobierno, academia u otros contribuyen a crear *flujos de conocimiento* para la generación de condiciones estructurales, sistémicas y sociales. Este modelo es similar al que proponen Nonaka y Takeuchi (1999) quienes explican las formas sociales de interacciones que se construyen y en las que participan distintos agentes, como son académicos, empresarios, agentes gubernamentales y los de asociaciones; así como un contexto amplío de políticas públicas, sistemas organizacionales, sociales y culturales; interacciones que se suceden en diversos contextos nacionales, sectoriales, empresariales, académicos y en el propio sistema productivo.

La identificación y caracterización de las distintas *prácticas sociales de aprendizaje*, son cruciales para explicar los *flujos de aprendizaje* entre distintos actores institucionales, sociales y empresariales; flujos que contienen coincidencias de intereses en el desarrollo de proyectos de investigación (Casas, 2001; Luna, 2003; Gutiérrez, 2003); como es la colaboración, la confianza, las representaciones y las capacidades de traducción de las relaciones sociales interpersonales (Luna y Velasco, 2003; 2005); así como las formas de organización institucional (Gutiérrez, 2003).

Estos contextos son los que caracterizan la asociación, alianzas o integraciones entre los agentes sociales, gubernamentales, empresariales y académicos que participan en la construcción del entramado que representa el conjunto de colectividades productoras de software en el Valle de México, destacándose distintas *prácticas sociales de aprendizaje*, por citar algunas: a) sistemas colectivos de integración (SICOLIN); b) comunidades tecno-académicas y c) articulación de conocimientos; prácticas que a su vez construyen heterogéneos *flujos de aprendizaje social*, como se muestra en el esquema 8 para las empresas tipo 1.

Esquema 8

Sistema de prácticas sociales de aprendizaje (SICOLIN) entre las empresas tipo 1 que configuran un flujo de aprendizaje en el Valle de México.



Elaboración con base en las entrevistas.

En el presente apartado se propone identificar las *prácticas sociales de aprendizaje* que se generan no de forma prescrita o prediseñada, pero que resultan fundamentales para la arquitectura del *flujo de aprendizaje* que configura el trabajo colectivo entre distintos agentes, cuyo desempeño responde a distintas lógicas institucionales y trayectorias personales.

Consideramos que las distintas *prácticas sociales de aprendizaje* son generadas por una serie de configuraciones sociales que resultan necesarias para facilitar la generación del *flujo aprendizaje* y con ello, se puede decir que, también facilitar la interacción social entre agentes.

A.- Sistema colectivo de integración (SICOLIN)

Las empresas enlistadas en el tipo 1, se caracterizan por promover la conformación de un *sistema colectivo de integración* (SICOLIN) en el cual se identifican cuando menos tres tipos de prácticas sociales: i) *aprendizaje social situado* y no sólo de participación/colaboración dentro de una red heterogénea; ii) *aprendizaje en colectividad;* hace referencia a las interacciones sociales entre distintos actores institucionales y de los procesos cognitivos que le sustentan para transformar la información en conocimiento y, iii) *construcción de significados y representaciones*; hace referencia a las percepciones, sentidos e imágenes del programador de software frente al trabajo.

El carácter flexible, ágil y dinámico del *sistema colectivo de integración* (de aquí en adelante SICOLIN) de las empresas de software en el Valle de México se propone partir de las interacciones sociales entre los agentes de las empresas enlistadas en el tipo 1, que construyen una serie de alianzas intermitentes, caracterizadas fundamentalmente por establecer coaliciones temporales, basadas más en una coyuntura de necesidades e intereses; la colaboración es por proyecto de trabajo, por necesidades especificas, como pueden ser recursos humanos especializados, recursos técnicos de una plataforma tecnológica, recursos tácitos, etc.; en otras palabras, el colectivo de empresas y recursos participan cuando los proyectos que se desarrollan lo requieran. El colectivo de empresas y recursos es un sistema reconfigurable, que se sustenta en procesos ágiles, flexibles, dinámicos, acorde a las contingencias del proyecto de software. En otras palabras, supóngase que la empresa MIXE como integrante del colectivo SICOLIN posee una plantilla de 10 programadores senior, 5 programadores junior, 2 lideres de proyecto y, un gerente de proyectos; la empresa MIXE está

desarrollando un proyecto de software que requiere de 2 nuevos programadores especializados en un lenguaje de programación orientada a programación Web, más un líder de proyecto experto en una plataforma tecnológica Sun; la empresa MIXE tiene dos opciones: recurre al SICOLIN para solicitar recursos humanos en el colectivo (práctica social de integración en recursos humanos) o bien, MIXE cede el proyecto al colectivo SICOLIN; donde la empresa MIXE es la líder del proyecto. De tal forma que el proceso de trabajo al interior del SICOLIN es colectivo pero participan sólo las empresas que poseen las capacidades requeridas por el proyecto, es decir no colaboran todas las empresas que integran la alianza, solamente aquellas que poseen las habilidades demandadas; es de integración porque se complementan los recursos técnicos, de aprendizaje, de conocimiento tácito o explícito, que aseguren el desarrollo del proyecto de software.

En las empresas que integran el SICOLIN, se detecto la configuración de tres *sistemas colectivos de integración* que podríamos señalar como *flujos de aprendizaje* diferenciados, con *prácticas sociales de aprendizaje* similares, con determinadas particularidades, pero con diferenciaciones que no abordaremos del todo en la presente, pero que intentaremos describir las *prácticas sociales de aprendizaje*, como son: i) aprendizaje socialmente situado; ii) aprendizaje en colectividad y iii) construcción de significados. Los tres SICOLIN que identificamos en las empresas tipo 1, -que por razones de anonimato denominamos con abreviaciones- son: sicolin *Qa*; sicolin *Em* y sicolin *Fs*. El SICOLIN *Qa* se conformó entre Noviembre y Diciembre de 2003, compuesto por siete empresas (de las cuales se estudiaron 3); *Em* entró en operaciones entre Junio y Julio de 2003, estuvo inicialmente compuesto por nueve empresas, de las cuales estudiamos dos empresas; el tercero *FS*, entró en crisis, por problemas de una alta nomina y pocos proyectos que sufragaran los sueldos y salarios (se estudió una empresa); *Fs*, fue fusionado por *Qa*.

Nos concentraremos en los primeros dos, ambos empezaron a interaccionar más fluidamente hacia fines de 2004, que es el año en el cual se inscriben las empresas del Valle de México en el programa público de PROSOFT (2004, 2005 y 2006), que como ya dimos cuenta en el capitulo 5, el valle de México contaba con una infraestructura digital de diez puntos (Ruiz, 2004), carecía de un programa formal de promoción de la industria del software, así como de un entramado social entre agentes que construyeran las sinergias necesarias para participar en las distintas áreas de promoción, producción, apoyos financieros y de gestión

administrativa de la industria del software. Se logró cierto grado de integración y formalización de la industria del software hacia fines del 2003 y se fortalecieron en 2004 a través de SICOLIN *Qa* y *Em*.

Las empresas que integran los SICOLIN coinciden en un contexto similar: están insertas en un proceso de consolidación, expansión y apertura hacia el extranjero; tienen convenios con el sistema académico local y nacional, como en el extranjero; una interacción fluida con asociaciones nacionales e internacionales, con centros de investigación; alianzas estratégicas con empresas trasnacionales como Microsoft, SAP, Oracle, SUN, HP, Peoplesoft, entre otras; etc. Entre las principales características del SICOLIN *Qa* y *Em* estudiados destacan: i) En caso de grandes proyectos de desarrollo que requieren un número amplio de programadores, las empresas medianas e incluso grandes al no disponer de recursos humanos y técnicos y de conocimiento tácito se integran en un colectivo determinado por los agentes empresariales y se presentan ante el cliente como una sola institución (*Qa* ó *Em*); ii) El colectivo de empresas a través del SICOLIN representan una serie de re-configuraciones flexibles, dinámicas, ágiles; no sólo en cuanto al tradicional "tamaño" de las empresas, sino que esta embebido en una especie de flexibilidad organizacional, flexibilidad numérica y flexibilidad productiva; al ampliarse las posibilidades de participación en mas de un sector tradicional o algún proceso en especial, haciéndose mas extenso el abanico de opciones, no sólo hacia la manufactura, sino también hacia alimentos, finanzas, gobierno, telecomunicaciones, salud etc.; iii) En caso de un proyecto de desarrollo complejo, que no necesariamente implique mas recursos técnicos, pero se requiera de conocimiento tácito, es decir de personal con determinado perfil de habilidades y destrezas, se solicita la intervención de los recursos humanos de otras empresas que integran el SICOLIN; o bien el proyecto que se trate puede ser retomado por la colectividad *Qa* o *Em*. Se identifican los recursos humanos necesarios y se les invita a que participen, no como individuo de la empresa A o B, sino como integrante del SICOLIN *Qa* ó *Em*, práctica social que denominamos *aprendizaje en colectividad*. Esta organización flexible, va mas allá de la flexibilidad numérica, y quizá estamos en presencia de un nuevo tipo de flexibilidad organizacional que podemos denominar *modo de producción ágil*. En el entendido, de que lo que esta en juego en este tipo de procesos de trabajo cognitivo es la construcción de confianzas para continuar con un *flujo de aprendizaje tácito*, donde fluyen habilidades y destrezas personales.

Las *prácticas sociales de aprendizaje* al interior del SICOLIN son resultado de una serie de interacciones sociales internas y externas que se suceden entre agentes institucionales, como son las asociaciones, universidades, política industrial, etc.; agentes industriales como empresas y asociaciones; así como acciones individuales de actores como son los programadores, líderes de proyecto, gerentes, etc.; interacciones sociales que generan prácticas de aprendizaje, como son los *aprendizajes situados, aprendizajes en colectividad y construcción de significados* entre agentes, actores e individuos que participan; éste tipo de aprendizajes entre empresas es diferenciado, acorde a las trayectorias de las empresas, maduración del SICOLIN como sistema; fortaleza e identidad de intereses y representaciones de las asociaciones, centros de investigación, académicos e incluso actividades gubernamentales que presionan y constriñen la acción individual y empresarial.

Por ejemplo la empresa MIXE estableció una serie de convenios con distintas universidades, alianzas con empresas competidoras, unificación de criterios en tormo al proceso de trabajo (calidad, procesos, metodologías, normas, etc.). Al interior de esta empresa -que forma parte del SICOLIN *Qa*-, uno de los gerentes y lideres de proyecto JA, es uno de los agentes claves en la consolidación del SICOLIN *Qa,* el gerente JA de MIXE, fue uno de los creadores de una serie de propuestas y alianzas con otras empresas para conformar el SICOLIN *Qa*, le tomo más de cinco años de gestión, crear lazos de confianza, de negociaciones y consensos que culminaron en la creación de una estructura organizativa, así como el antecedente fallido de un SICOLIN *Fs* que entró en crisis financiera.

> "hasta el año pasado se llevó a cabo la cuestión de que ya no podía solamente estar la academia, sino que también tenía que ser la industria; entonces es cuando entran todas las empresas de *Qa*. *Qa* es la integradora más fuerte... son siete empresas, todas se dedican a lo que es el desarrollo del software, esa integración es avalada por la secretaria de economía. Nosotros recibimos la cedula como una de las principales integradoras de empresas en el desarrollo del software" (Líder MIXE JA2).

El enfoque del SICOLIN, parte de la referencia metodológica de construir los procesos competitivos a través de generar proceso creativos entre agentes para comprender las distintas prácticas sociales y las capacidades de innovación (lo cual no quiere decir que hay ausencias de conflictos, choques, crisis, negociaciones, resistencias, etc.). Las prácticas sociales y las capacidades de innovación, dependen a menudo de la interacción entre agentes que generan *prácticas sociales de aprendizaje* y, éstas son favorecidas por la *proximidad geográfica* entre

actores (en términos de distancia-tiempo-territorio); pero más allá del territorio, está la *dinámica relacional* que supone compartir un *saber hacer* técnico para un aprendizaje social, como contar con recursos humanos competentes, especializados, etc.; y la *dinámica institucional*, entendida como un conjunto de sinergias participativas en asociaciones, programas gubernamentales, sentidos y percepciones de la empresa frente a la industria, etc.

El conjunto de *prácticas sociales de aprendizaje* entre empresas, supone repertorios de *rutinas* organizativas y *repositorios* de conocimientos tácitos y explícitos. Ambos, son recursos iniciales para hacer frente a problemas coyunturales planteados en contextos de información limitada y contingencias decisivas que presionan estructuralmente el *saber hacer* técnico; por ejemplo, las contingencias en una cabina aérea, la toma de decisiones de un inversionista financiero, la decisión de un programador al construir un algoritmo, etc. Para Nelson y Winter (1982), y Teece (1988), *las rutinas* hacia el interior de las empresas son las que desarrollan un sistema de competencias en el saber hacer de la empresa, entendidas como aquellas habilidades y destrezas que configuran los "sistemas competitivos" que distinguen a las empresas como especialistas. Son *las rutinas* transformadas en prácticas sociales a través de la acumulación e interacción en determinados ámbitos de las empresas que se constituyen como un conjunto de habilidades, destrezas, significaciones y sentidos, que denominaremos como *practicas sociales de aprendizaje* de una empresa o un sistema de empresas.

Consideramos importante señalar que hasta este punto, los *flujos de aprendizaje* y las *prácticas sociales*, en un contexto macro y meso no son suficientes para explicar como hacer frente al conjunto de contingencias e incertidumbres en el proceso de trabajo del software; consideramos que se debe abordar la perspectiva "microsicial" de las relaciones laborales; explicar las subjetividades, como los sentidos, representaciones y significados del programador en las interacciones sociales, que negociaciones y conflictos, resistencias y juegos de poder concurren entre los distintos agentes productivos; consideramos que las dinámicas relacionales y organizativas no explican por si solas la complejidad de proceso de trabajo y, por el contrario pueden obscurecer lo "realmente existente" hacia el interior del proceso de trabajo. Sin embargo, es importante conocer el contexto macro y meso en el cual se inserta la industria del software.

B.- Colectividad tecno-académicas

Para el caso de los centros digitales de Nuevo León o Jalisco (Valle del silicón mexicano) la *proximidad geográfica* entre los agentes institucionales como empresas, universidades, centros de investigación, etc., son significativas; sin embargo coexisten otras características importantes como los acuerdos, convenios e identidad de intereses culturales que se comparten en el territorio, espacio o región, dando lugar a procesos colectivos que se generan a partir de la *dinámica relacional e institucional* que se configura entre los agentes. Es decir la *dinámica organizacional* forma parte de la coyuntura en la generación de *prácticas sociales de aprendizaje*. Por ejemplo, los SICOLIN *Qa* y *Em*, coinciden en las estrategias de generar proximidades del tipo organizacional (relacional e institucional) con el sector académico (público y privado), con centros de investigación locales e internacionales, así como acciones individuales, como impartir clases en las universidades, estancias académicas de alumnos en las empresas con valor curricular, intervención de los gerentes en los planes curriculares en aquellas instituciones académicas donde haya convenios de participación, etc. Así las empresas que integran los SICOLIN con excepción de DYYA, promueven un programa que busca generar una red de talentos para la industria del software, esta red se organiza en función de dos grandes ejes: i) Con las universidades locales se han convenido estancias en las instalaciones de las empresas, a cambio se les otorga una beca los estudiantes, en algunos casos la beca es financiada por la Secretaria del Trabajo (STPS), la Secretaria de Educación Pública (SEP), ó por la empresa. Este tipo de convenio universidad-empresa, aglutina a más de 15 universidades públicas y privadas, más de 20 empresas, y asociaciones de la Industria del Software; ii) convenios con universidades del extranjero a través de organismos internacionales. Aquí operan organismos como CONACYT, TECH-BA, Secretaría de Relaciones Exteriores (SRE), FUMEC, etc. Todo el programa en su conjunto, se denomina Tecno-Eje, que en 2005 obtuvo un financiamiento de PROSOFT superior a los cinco millones de pesos para construir un edificio que albergará estancias académicas, centro de capacitación e investigación aplicada al desarrollo de software a la medida.

Una de las empresas con mas influencia en el SICOLIN *Qa* es MIXE que después de un periodo de más de 3 años de intentar establecer convenios firmes con las universidades locales, hacia fines de 2005 inició un proceso de consolidación para genera una práctica social de *aprendizaje situado*, en el entendido de que los alumnos avanzados de un conjunto de

Universidades locales, como UNITEC, Fidel Velásquez, TEC de Monterrey campus Ciudad de México, Universidad Autónoma Metropolitana, UPICSA del IPN, entre otras; residirán por un periodo de tiempo en las empresas (residencias) interactuando con las necesidades "reales" de la programación moderna y "practicando" como se hace el software:

> "...La Fidel Velásquez, Universidad del Valle de México, UPICSA del Politécnico, próximamente con la UAM. Entonces, todo esto es por que hay un corredor, que aquí nosotros le llamamos tecnoeje... Este Tecnoeje agrupa a todo este tipo de Universidades... y empresas" (Líder MIXE JA2).

Otra empresa con fuerte presencia en el SICOLIN *Qa* es la empresa CISE, que ofrece becas a los programadores recién graduados para que realicen estancias académicas en sus instalaciones, cabe señalar que esta empresa cuenta con todo un sistema curricular, diseñado por el gerente de capacitación JH0, quien posee estudios de Maestría en Ciencias de la Ingeniería del Software. El gerente JH0 señala que el sistema educativo no está capacitando a los ingenieros en la "frontera del conocimiento" que requiere esta industria; señala que las prácticas institucionalizadas en el programa global denominado en español como *buenas prácticas*, que alude a una serie de acciones y métodos que el programador debe cumplir, no se enseña su contenido en las universidades. Y esto es así porque muchos de lo profesores no cuentan con la experiencia de campo, luego entonces no son capaces de preparar a los alumnos en los requerimientos que solicita la industria.

> "...aquí tenemos un programa de becas, es gente que se gradúa y que nosotros la preparamos, la entrenamos con nuestra metodología y nuestras prácticas, y luego la incorporamos a proyectos; estamos invirtiendo en generar conocimiento y hacer crecer a nuestra gente" (gerente CISE JH0).

El directivo JH0, reconoce que el conocimiento entre la parte empírica y académica no se vinculan, no se desplaza la información a las aulas, aunado al hecho que la velocidad de cambio en esta industria es constante.

> "...eso es un circuló vicioso porque, uno va muy rápido... la gente lo va aprendiendo sobre la práctica, sobre sus necesidades y la gente que lo aprende nuevamente no es una gente que de clases (...) Entonces es un círculo vicioso que nunca va a terminar... no se regresa el conocimiento" (gerente CISE JH0).

La tercer empresa de este grupo SOCU implementa una estrategia mixta, posee una dirección de capacitación, separada de la empresa, es un lugar confortable en la delegación

Coyoacan, con un área lujosa, con dos salones principales, equipados para impartir asesorías. SOCU utiliza un concepto amplio de *universidad de servicios,* en el cual ofrece capacitación en todos los niveles. El responsable de la dirección de capacitación es el gerente JZ5, quien expresa que en el proceso de trabajo del software no se están implementando estándares de calidad, cada programador tiene su propia forma de conducir a buen término un proyecto, lo que explica que existan incompatibilidades al momento de integrar los módulos desarrollados, lo cual se traduce en errores en el desarrollo.

> "....Toda empresa...requiere conocimientos y...lo más importante tú necesitas ya empezar a conocer y manejar ya estándares... tú sabes manejar un proyecto de una manera -si quieres llamarlo autónoma- pero sabes hacerlo. Sin embargo de que sirve... que cada persona, maneje un proyecto de manera distinta si a la hora de la hora cuando empiecen a integrarlo va a hacer un santo relajo" (Director académico SOCU JZ5).

Estas estrategias de colaboración entre agentes institucionales se suceden no sólo por las proximidades geográficas, sino también identidad de intereses, representaciones y sentidos de los individuos que interaccionan socialmente, interacción que es una forma de *aprendizaje en colectividad*, mediado por estrategias del tipo reflexivas, rutinarias y cognitivas entre agentes sociales, como son asociaciones, universidades, política industrial, etc.; industriales como empresas; individuos como los programadores, líderes de proyecto, gerentes, etc. Es decir, no basta con coincidir en cuanto a intereses o afinidades culturales, industriales o de aprendizaje, también es importante reconocer la acción social individual, así como el aprendizaje social entre las instituciones.

Las practicas *sociales de aprendizaje* que se generan en y a través de las distintas prácticas sociales de aprendizaje (flujos de conocimientos tácitos y explícitos) que se producen a menudo a través de lo que Wenger (2001) llama *comunidades de práctica*; ó bien Hakanson (2005) denomina *comunidades epistémicas*; en una terminología parecida es el término *arquitectura de las comunidades en práctica* de Amin y Cohendet (2004). Nosotros, retomaremos el concepto de *comunidades de práctica y significaciones* y, haciendo énfasis en el carácter colectivo e informal del aprendizaje de los individuos que desarrollan las mismas prácticas diarias de trabajo en el contexto de determinadas organizaciones, pero éstas prácticas están embebidas de negociaciones, conflictos e intereses, resistencias e interjuegos que se resumen en el concepto de significaciones. Mientras que la noción de *comunidades*

epistémicas subraya más el carácter inter-organizacional de las *comunidades basadas esencialmente en el conocimiento* (Larsson et al 2006). Consideramos que la propuesta que denominamos *comunidades de práctica y significaciones*, permitirá construir una serie de proposiciones explicativas que hagan posible la percepción de las configuraciones sociales del aprendizaje a través de relaciones informales y la transferencia de conocimiento tácito y codificado que concurre en el proceso de trabajo del software a la medida. Consideramos que las empresas líderes que componen el colectivo SICOLIN poseen un conjunto de competencias suficientemente interactivas como para crear relaciones formales e informales, al tiempo que se apoyan en una configuración social de aprendizaje interna, basadas mas en la articulación relacional de intereses entre los individuos (significaciones); que como daremos cuenta en seguida, es la articulación local lo que permite a los actores, contar en la región con un *flujo de aprendizaje,* que expande sus fronteras más allá de los espacios regionales.

C.- Vínculos del conocimiento

Consideramos que en el SICOLIN coexisten interacciones sociales internas y externas a la empresa que configuran procesos de aprendizaje informales y formales; por ejemplo el líder de proyecto JA2 de la empresa MIXE gestiona hacia el interior del SICOLIN *Qa* una red de talentos locales mas allá de los espacios geográficos, es decir la cohesión de un SICOLIN va más allá de las formas de proximidad territorial y cobran importancia otros tipos de dinámicas relacionales e institucionales, sean formales e informales, locales y extra-locales, ambas deben complementarse para generar espacios de aprendizaje.

> "MIXE es una empresa que se dedica al desarrollo del software y se esta enfocando a la parte de lo que es la generación de talentos, a hacer fortaleza, que es una de las cosas que hemos visto que en el mercado tiene deficiencia, por que tenían esa brecha tan grande entre la academia y la industria. Entonces estamos...reduciendo esa brecha por medio de la vinculación con universidades. Ahora, el proceso principal ha sido la alianza con Universidades Tecnológicas para impulsar a los jóvenes para que continúen estudiando pero ya con el conocimiento, ya ingresando hacia la parte productiva (Líder MIXE JA2).

El líder JA2 es un actor clave, no sólo en la conformación de una red de talentos locales, que se estructura alrededor de un proyecto denominad tecno-eje; también es uno de los principales promotores de construir "desde abajo" la propuesta de los *sistemas colectivos de integración*, propuesta que si bien no es nueva en la industria de la manufactura, para el

caso de la industria del software quizá si lo sea, ya que PROSOFOT, por ejemplo no admitía este tipo de organizaciones empresariales (SICOLIN), reconociéndole jurídicamente y apoyándole con recursos financieros hasta la convocatoria de PROSOFT-2006.

> "Entonces hasta el año pasado se llevó a cabo la cuestión de que ya no podía solamente estar la academia sino que también tenía que ser la industria, entonces cuando entran todas las empresas de *Qa*. *Qa* es la integradora más fuerte...la idea es generar el talento y con eso generas capacidades para toda y cada una de las empresas. Nosotros al ser empresas pues prácticamente generamos programadores, vemos que ya son capaces y los turnamos ya, a empresas para que trabajen" (Líder MIXE JA2).

La configuración de la red de talentos y el tecno-eje representa una articulación de conocimientos no sólo a nivel local, sino también orientada al extranjero, ambas estrategias locales están vinculadas a centros de investigación, sistemas académicos y procesos productivos internacionales a través de apoyos gubernamentales y privados como los que provee TECH-BA y CONACYT; entre otros programas privados y públicos de la Secretaría de Relaciones Exteriores y BANCOMEXT en centros de producción global localizados en California y Texas en Estados Unidos, así como en Madrid, España.

7.2.2.- Crisis y reestructuración interna entre empresas tipo 2

En las empresas tipo 1, dimos cuenta de la configuración de *flujos de aprendizaje* a través de distintas prácticas sociales, como son el *aprendizaje social situado*, *aprendizaje en colectividad y la construcción de significados*, éstas prácticas sociales son constreñidas por estructuras territoriales (proximidad geográfica) y por interacciones creativas que denominamos dinámicas relacionales e institucionales (proximidad organizacional); así como otro tipo de presiones, de agentes locales, nacionales e internacionales. Sin embargo, como daremos cuenta en el presente sub-apartado, las empresas que denominamos del tipo 2, son microempresas con menos de 10 programadores, están iniciando nuevamente después de una crisis económica, que en el argot de las tecnologías de la información y comunicación se denomino "crisis de las punto.com".

Las empresas tipo 2 no están insertas en un proceso externo de configuración del tipo SICOLIN, lo cual no quiere decir que no estén interesadas o que no evalúen la importancia de un proceso de interacciones sociales orientadas al aprendizaje que les permita generar articulaciones con otras empresas de software en la región; es decir la dinámica institucional

no está presente en estas microempresas; hacia el interior de la organización empresarial no han cristalizado intereses y representaciones que les conduzcan a conformar otro tipo de estructuras organizacionales del tipo SICOLIN, lo cual no quiere decir desinterés, apatía o desconocimiento. Las explicaciones son múltiples, sin embargo en las entrevistas percibimos una homogeneización de algunas respuestas; desde aquellos gerentes que señalan que no les interesa participar en asociaciones especificas, o bien no les interesan apoyos financieros gubernamentales (PROSOFT, SHCP, CONACYT, AMICIS, etc.); pero destaca más una estructura homogénea de crisis, desorganización e individualismo; no poseen convenios con universidades, no pertenecen ninguna asociación civil, no tienen convenios de colaboración con empresas trasnacionales. El contexto en el cual se insertan este tipo de empresas implica escaso apoyo financiero, desarrollan proyectos pequeños, están "ancladas" a un nicho de mercado pequeño acorde a sus capacidades; están saliendo de una crisis, están e proceso de reestructuración, tienen deudas, no tienen asesorías financiera y técnica; etc.; otros argumentos destacan la volatilidad del mercado, los altos costos de mantener una nomina de programadores, mayor competencia local, caída en el margen de utilidades, competencia desleal, altos costos de la mano de obra calificada, etc

> "nuestra empresa decidió en enero de este año (2005) no contar con un área de desarrollo...El detalle importante es que nos estamos apoyando con empresas especializadas en el desarrollo de software que se han ido consolidando en el transcurrir de los años (*subcontratación*), y se han fusionado en grandes empresas para dar este servicio de una manera más concreta, incluso más barata..." (Líder CATI DC2).

Otra característica de las empresas del tipo 2 es que están inmersas en un proceso de reforma interna, ocasionada por distintos factores: Crisis del sector especulativo de "empresas punto.com" (1994-1996), mayor presencia de empresas globales en el mercado nacional, regulación del mercado a través de normas y estándares, apoyos gubernamentales, políticas financieras directas e indirectas sólo para empresas que implementen procesos de calidad certificadas, caída en las utilidades, mayor exigencia en cumplimiento de normas y estándares de calidad en el proceso de trabajo, presencia de nuevas tecnologías que exige asistencia a cursos, personal calificado, pago de regalías, pago de patentes, etc. Este contexto implicó un retraso técnico, competitivo y, sobre todo, restricciones para salir de la crisis económica y generar la creación de flujos de aprendizaje. Por ejemplo las empresas MEKI y TADI están

saliendo de la crisis. MEKI llegó a tener oficinas en Nueva York (1994-1995), ofrecía servicios de software global, contaba con una planta laboral mayor a 50 programadores localizados en la ciudad de México. Después de la crisis en 1995 quedó endeudada con más de 1.5 millones de dólares. Hoy se especializa en software para palm-top. Recientemente empezó a ofrecer desarrollo de software a la medida, pero solo a empresas locales. TADI, poseía un centro de desarrollo de software con más de 20 programadores, ofrecía software a la medida. Con la crisis de 1994, tenía deudas superiores al millón y medio de pesos con diferentes bancos; al respecto el gerente y dueño de la empresa RS0 señala que para solventar la crisis, vendieron pasivos particulares como carros, bienes inmuebles y tuvieron que mudarse de oficina para subsanar los pagos; cerró operaciones en 1994; operó entre 1995-2004 con diversos registros empresariales, se desempeño más como una empresa de servicios para con la cartera de cliente que tenían ofreciéndoles mantenimiento y actualización a los programas de software que había proveído a dichos clientes. Para 2005, cuando se realizó la entrevista, el gerente y dueño, comentó que la empresa tendía a estabilizarse gracias a un software contable que habían desarrollado a principios de los noventa y que era utilizado por más de 300 empresas locales, las cuales le proveían de recursos paras seguir operando, como proveedora de servicios, más que como desarrolladora de software.

Por otro lado, las empresas HUTI y CATI entre Enero y Junio de 2005 habían cerrado el área de desarrollo de software. HUTI cerró operaciones porque le resultó imposible mantener una nómina de 15 y hasta 20 programadores y no había los suficientes proyectos. Esta empresa, al igual que TADI, hacia fines de los ochenta desarrolló un programa de software administrativo para médicos, abogados, etc. con relativo éxito, contando con más de 100 clientes, sin embargo no se comercializó lo suficiente y la competencia de empresas globales que ofrecían programas similares, económicos y con mas funciones terminó por expulsar del mercado el software de HUTI. El gerente y dueño de HUTI, a Junio de 2005, continúa en la industria del software, pero como proveedor de proyectos internacionales de las empresas locales, es decir, se dedica a promover la industria del software del valle de México en California y Texas y recientemente en Madrid, España; forma parte de las empresas insertas en TECH-BA-CONCAYT-Secretaría de Economía.

La empresa CATI cerró el área de desarrollo de software y subcontrata empresas para que desarrollen los proyectos que le solicita la cartera de clientes que posee. El gerente DC2

de CATI comenta que subcontrata a los ex-empleados de ésta empresa para el desarrollo de software; señala que el mercado de software "dejó de ser negocio" en la medida que ingresaron empresas globales con estándares de calidad, con software estándar masivo y la competencia desleal de los freelance. La empresa SITA, que es una asociación entre dos gerentes que proceden a su vez de empresas que cerraron operaciones; están inmersos en un proceso de localización de nichos de mercado donde especializarse; en Junio de 2005 la empresa tenía una plantilla de 17 programadores, desarrollando proyectos para empresas locales e investigando programas de software propios, tipo software estándar para el mercado local.

Las empresas tipo 2, si bien no poseen convenios de colaboración, no están insertas en programas de integración, ni se vinculan con grandes empresas, no son proveedoras de software complejo, si tienen el objetivo de fortalecer las capacidades internas de organización, de incorporar metodologías estándar, entendiendo por ello a un conjunto de normas y procedimientos que generen al interior de la empresa un proceso de trabajo mas competitivo, pero no se trata de implementar procedimientos caros y complejos como las normas de calidad CMM e ISO, sino el de crear condiciones mínimas de organización interna.

> " ...los errores principales de lo que es programación antigua (...) era un error dejar que el programador decidiera como se iban a llamar las variables (...) esa libertad era buena para el programador (...) pero muy mala para el que seguía, el que seguía no entendía absolutamente nada de lo que narra el código que estaba viendo y sobre todo (...) es que como no había estándares en la programación y manejo de la documentación pues era cualquier cantidad de problemas para el usuario final, como el siguiente programador entendiera lo que estaba haciendo esa aplicación..." (Líder CATI DC2).

Las empresa tipo 2 están en un proceso de reestructuración interna, presionados por el mercado, por las grandes empresas, por la cada vez mayor presencia de estándares y procedimientos normativos para con un proyecto de software. Esta coerción externa es paralela a las representaciones de los actores laborales en las empresas, que redefinen sus acciones cotidianas orientadas a hacer frente a dichas presiones.

7.2.3.- *Comunidades virtuales de aprendizaje (open source)* en las empresas tipo 3

Las microempresas que conforman el tipo 3, se caracterizan por desarrollar software a la medida, pero basadas en plataformas tecnológicas y lenguajes de programación diferenciados

de las empresa tipo 1 y 2, la divergencia reside en el uso de un *sistema de signos y símbolos disponibles en la red*, que no necesariamente implica que sean gratis, sino que es un conjunto de algoritmos *open source* que conforman el *modo de desarrollo de software libre*, que es una configuración de *signos y símbolos disponibles,* representan una serie de rutinas de algoritmos que, a su vez forman parte de tareas de un programa de software. De tal forma que algunas tareas logran dar forma a nuevos programas (reutilización de código); o bien un programa existente consigue transformarse mediante la manipulación de los algoritmos en programas de software con inéditas rutinas, nuevos procesos o un diseño grafico optimizado.

El acceso al *sistema de signos y símbolos* del tipo *open source* se sucede a través de superar una serie de *barreras cognitivas* embebidas en una serie de *prácticas nomotéticas de aprendizaje;* por ejemplo, las representaciones y significados del hacker; la personalidad y sentidos de la cultura geek; la identidad para con la comunidad (Debian, Ubuntu, Gnome, Linux, etc.); el sentido del humor y habilidad cognitiva, etc. constituyen una serie de hábitos y modos de ser y proceder dentro de una *comunidad de aprendizaje virtual*, que se genera éstas *barreras cognitivas* como un proceso de identificación o restricción al programador medio que incursiona en los procesos de integrarse a *comunidad de aprendizaje virtual*. Sin embargo, son las distintas configuraciones que se instituyen como *interacciones virtualizadas* entre los distintos agentes institucionales que, constriñen y ciñen el proceso de trabajo de las empresas que desarrollan software a la medida, utilizando software libre (*open source*).

Los distintos agentes empresariales que utilizan software libre como Google, Wikipedia, NASA, Sony, Cisco System, IBM, AOL-Anywhere, Inte, Oracle, entre otras empresas; para el caso de empresas nacionales y locales en el Valle de México, se identifican empresas como: Inteligentes.com; GCIS (Global Consultants on integrated Solutions S.A. de C.V; Sayan Technologies; Grupo SIS; VoIP Technologies; Pragxis; Acsinet S.A. de C.V.; Factor Evolución (Linux para todos); Infotec; Open Intelligence S.A. de C.V; Nul Unu S.A. de C.V.; Red Hat de México S.A. de C.V.; Lingo System S.A. de C.V.; Software and Computing Services S.A. de C.V.; Synetcom Consultoría de Sistemas y Redes S.A. de C.V.; Tecnología Especializada Asociada de México S.A. de C.V; entre otras empresas que entrevistamos y que enlistamos con siglas (ver cuadro 22). Además coexiste un conjunto de asociaciones civiles locales de software libre, como AMESOL, Software.net; PROSOFT, Secretaría de Economía; así como múltiples programadores freelance; configuran todos ellos un conjunto de agentes

institucionales que conforman una serie de *comunidades de aprendizaje virtual* disponibles en Internet como son: www.debian.org; www.ubuntu.com; www.gnome.org; www.linux.org; entre muchas otras distribuidoras y desarrolladoras de software libre que operan a nivel global y local.

Consideramos que uno de los principales aportes del *modo de desarrollo en software libre* son las *comunidades de aprendizaje virtual* que configuran distintos *flujos de aprendizaje*, con sus propias singularidades, se constituyen como "tribus comunitarias de aprendizaje". En el argot informático del *open source* se les distingue a las heterogéneas tribus y sus integrantes como "debianeros", que son aquellos programadores que utilizan los programas desarrollados por www.debian.org; "linuxero". www.wikipedia.org los define como "una persona que usa habitualmente este sistema operativo...suele cooperar en actividades de Linux, tales como desarrollo de software...o simplemente adora este sistema operativo por encima de...Windows, Unix propietario..." (Wikipedia.org acceso 10 de Julio de 2008). Junto con otras comunidades como los gnomeros, ubunteros, etc; estas *comunidades de aprendizaje virtual* contribuyen a nivel global al desarrollo del *open source* y operan con diferentas *prácticas nomotéticas de aprendizaje*: *aprendizaje centrado en la fluidez de la información* hace referencia al acceso libre a los contenidos del software y existencia de foros de discusión y colaboración interactiva; *construcción de significados* más que la transmisión de contenidos. Sin embargo, dichas *prácticas nomotéticas de aprendizaje*, las podemos unificar en tres principales: *aprendizaje fluido*, *aprendizaje comunitario y construcción de significados*.

Las empresas enlistadas en el tipo 3 desarrollan programas de software a la medida utilizando software *open source,* lo cual no quiere decir que estén disponibles las rutinas de código en Internet, no significa que sólo se "adapte" el código y se genere un software; el proceso no es lineal, los programas de software libre deben superar una serie de *barreras cognitivas*, estar socializado con el sistema de *prácticas nomotéticas de aprendizaje* en las distintas comunidades de aprendizaje virtual, para acceder al conjunto de signos y símbolos que representa el *open sorce;* no sólo es un amplio repositorio de conocimientos que las empresas potencian, multiplican, adaptan y *re*-configuran bajo ciertos procedimientos de las propias *comunidades de aprendizaje*; consideramos que coexisten una serie de prácticas

nomotéticas para reconfigurar los signos disponibles en nuevos textos sígnicos que resuelvan los problemas planteados por el cliente o usuario final.

Re-visar los repositorios de código, *re*-diseñar las rutinas acorde a las necesidades planteadas por el usuario final, *re*-utilizar los dispositivos gráficos, re-aprehender procesos, en otras palabras, el open source implica conocer las practicas nomotéticas que se generar, estar familiarizado con los procedimientos cognitivos del software *open source,* que es promovido por las distintas distribuidoras virtuales, como Debian, Ubuntu, Linux, etc. Sin embargo, los procedimientos de acceso a estas comunidades se erigen como una serie de *barreras cognitivas* que están infiltradas entre los *flujos de aprendizaje* basados en *open source*, flujos que poseen sus propios símbolos y códigos culturales-virtuales; es decir, aquel programador "debianero", "linuxero", "gnomero", etc., está inserto en una serie de *interacciones visualizadas* que le permiten comprender la estructura y metodologías utilizadas en el conjunto de algoritmos *open source* de que se trate. Sin embargo, dichas *barreras cognitivas* se debilitan conforme el programador aprende las rutinas y procesos en los manuales, foros y guías de ayuda de la comunidad virtual que se trate.

> "...en los foros, en sitios de proyectos nuevos, en listas de anuncios de proyectos nuevos como freenet (http://freenetprojet.org)... bajar la aplicación, instalarla, ver como funciona, leer el código, incluso ver como la puedo adaptar a mis necesidades, en mi laboratorio propio en la casa..." (programador DINS)

En estas empresas tipo 3 que manipulan símbolos disponibles en Internet, no esta ausente de conflictos y resistencias que se conforman en la pantalla del programador; coexiste una serie de arreglos virtuales, negociaciones de significados y consensos entre el programador y el líder de proyecto; se eclipsa el boicot y el enfrentamiento en la pantalla del programador. Por ello, consideramos importante abordar un enfoque de micro-prácticas individualizadas, entendidas como acción social en la pantalla del programador, que consideramos constituyen parte explicativa de la *aflicción del software*; es decir, más allá de la configuración de los flujos de aprendizaje y las prácticas sociales embebidas en las interacciones sociales, ya sea en colectividades del tipo SICOLIN (empresas tipo 1) o en comunidades virtuales *open source* (empresas tipo 3) e incluso en las microempresas en reestructuración (empresas tipo 2), el análisis de redes no puede estrictamente substituir al de los procesos de trabajo.

7.3.- Procesos internos oscurecidos por las estructuras: Conceptualización de los requerimientos

En el presente apartado, abordaremos la organización de las empresas del valle de México, pero desde el segundo eje de análisis que en el anterior apartado no explicamos, es decir, aquí continuaremos una perspectiva micro, desde la acción social del actor individual, sea programador, líder de proyecto, gerente, tester de calidad o documentador de proceso. En el presente apartado extendemos los conceptos que se enunciaron en el capitulo 3, donde se explica que el proceso de trabajo del software a la medida comprende por cuatro grandes áreas: conceptualización, formalización, procesamiento de datos e implementación. En este apartado explicaremos desde la perspectiva de los actores la *conceptualización* del software, entendido éste como el diseño de concepto-grafías de los distintos requerimientos que solicita el cliente/usuario; hace referencia a describir el conjunto de requisitos del programa; en otro subapartado, se explicará la *formalización* de los requerimientos del usuario, y el proceso de cómo son aprehendidas y modificadas por el analista y/ó programador experto; ambos procesos, el de conceptualización y formalización de los requerimientos del programa está simbolizado por el proceso cognitivo; el *procesamiento de datos*, esta circunscrita a proseguir una serie de consideraciones lógicas entre los requerimientos trazados en el diseño o formalización del problema. El diseño se subdivide en módulos que poseen cohesión lógica y éstos a su vez se fragmentan en aplicaciones y éstas se dividen en cadenas o bucles; posteriormente las cadenas se transforman en instrucciones al programador, quien transforma dicha instrucción en un conjunto de algoritmos (símbolos) que construye a partir del lenguaje de programación (signos) que haya utilizado; consideramos que no es necesario abordar en este espacio el proceso de la *implementación* del software desarrollado porque ya se explico ampliamente en el apartado 3.8

Ahora bien, estas fases laborales no son estáticas, varían de acuerdo con el tamaño de la empresa, tipo de software que se desarrolla, cantidad de programadores disponible. Sin embargo, es común que quien desempeña una función en un proyecto podría desempeñar otra actividad en el siguiente proyecto, dependiendo de las habilidades y destrezas propias, como de las exigencias técnicas y metodológicas del proyecto; quien es analista en un proyecto en otro puede ser programador o líder. Lo transitorio de los puestos de trabajo parece estar relacionado con una serie de factores, desde los subjetivos como las habilidades, destrezas,

experiencia, etc. que se posee el programador para determinado proyecto, pero también porque las divisiones entre conceptualización/formalización y, por otro lado, procesamiento de datos/Implementación, no son rígidas, las fronteras entre ambas esferas no están bien delimitádas, determinadas o estructuradas; por el contrario el transito entre una y otra es habitual; los limites entre programador, gerente, analista y desarrollador están invisibilizadas, son otras características subjetivas las que las separan, como la experiencia, las habilidades y destrezas personales, arreglos y consensos, negociaciones formales e informales, etc.

7.3.1.- Empresas tipo 1

Una de la características de las empresas entrevistadas que denominamos del tipo 1 es que si bien existe toda una propuesta de mejora en los procesos de desarrollo, mediante la implementación de una serie de métodos y procesos postulados por la ingeniería del Software, en la práctica cada proyecto que se presenta posee una serie de particularidades que le hacen único; por ejemplo, tamaño y complejidad del proyecto, experiencia y conocimiento de los que participan en desarrollo del proyecto, experiencia de los clientes y usuarios del proyecto, cualidades y habilidades en el método y proceso que se utilizará, el objetivo y funcionalidad de aplicación, etc. (ve cuadro 23).

Cuadro 23

Particularidades de las empresas tipo 1

Empresas y tamaño	**Características de empresas tipo 1**
MIXE Grande	○ Desarrollos de Software Privado.* ○ Ofrecen servicios de asesoría, y consultoría a los clientes. ○ Desarrollo de proyectos sencillos a complejos.
SOCU Grande	○ Relación estrecha del Cliente con el Gerente ó líder de proyecto. responsable del proyecto. ○ Forman parte de Empresas Integradoras.**
CISE Grande	○ Son partner´s de empresas globales (IBM, ORACLE, etc.). ○ Aplicación formal de principios metodológicos, de la Ingeniería del Software. ○ Buscan certificar sus procesos en alguna norma de calidad.
DASA Pequeña	○ Menor flexibilidad numérica. ○ Estancias académicas en las Empresas desarrolladoras.
DYAA Micro	○ Formalidad en Equipos de Trabajo. ○ Definición/formalidad en jerarquías laborales. ○ Funcionalidad/especificidad en los puestos de trabajo.
ELLA Pequeña	○ Mercado local, nacional y de exportación. ○ Pertenecen a una Asociación de la Industria del Software.

Elaboración propia en base a las entrevistas.

Las discusiones en torno a estas particularidades varían, en el discurso de la calidad impera que se deben utilizar las normas ISO, CMM, etc., en la práctica no se implementan los

mismos procedimientos en cada proyecto, ya que cada proyecto posee sus particularidades, no hay un proyecto que contenga un "solo mejor camino de hacer las cosas" como en la organización científica del trabajo, al estilo del taylorismo, sino que existen varias formas de abordar un problema en el proceso de trabajo:

> "para poder realizar un proyecto, no solamente te puedes auxiliar de un solo proceso, son muchos procesos. Porque va a depender de todas las fases por las que atraviese tu proyecto. Entonces no puede existir una sola receta de cocina (...) cada proyecto tiene su propio dominio, sus propias complicaciones, su propio objetivo, (...) no es como construir un carro, o sea un carro sabes que a fuerzas tienes que empezar a ensamblar el chasis, y después el motor (...) como que hay cosas lógicas. Acá si lo hay pero, tienen la flexibilidad de escoger el ciclo de vida -acuérdate que hay varios ciclos de vida-... y dependiendo de ese ciclo de vida que tu escojas, es la manera en como vas a ejecutar los procesos, no puedes encontrar una sola manera. (...) Si no tenemos una buena práctica...de ingeniería de requerimientos, (...) es muy fácil que yo no entienda lo que el usuario o el cliente me este pidiendo, diseñe incorrectamente, construya incorrectamente, y le esté entregando algo que ni siquiera me pidió (Gerente CISE JH0).

Por otro lado, el líder de proyecto JA2 comenta que en la empresa simple y llanamente hacen entrevistas, reuniones y revisiones . Que la metodología que se estaba implementado en la empresa MIXE señalaba que debían nombrar un "comité de proyecto", que se integraba por un *líder* por parte de la empresa, y un *líder* por parte del usuario (cliente), ello no ocurría y, por otro lado, hasta el momento bastaba con una serie de entrevistas y revisiones entre el cliente y la empresa desarrolladora. Debemos de precisar que la realización de entrevistas y el procedimiento formal señalado por la ingeniería del software distaban de ser lo que la propia ingeniería señala. Explica el líder JA2 que el contacto con el cliente suele ser complejo y se divide en tres momentos (puede haber más): a) Entrevistas para definir la lista de requerimientos; b) Acuerdo del convenio de requisitos (puede ser formal mediante un contrato de requisitos o informal, solo de palabra); c) Aceptación de los módulos desarrollados que integran el sistema informático. Ninguno de estos momentos es estático, depende en gran medida del grado de experiencia e interés del cliente y usuarios finales.

En las empresas que denominamos del tipo 1, el que interactúa directamente con el cliente (cuando existe esta figura) es el líder del proyecto, el administrador de la configuración, el arquitecto o el gerente; sin embargo, el programador también tiene contacto con el cliente. Por ejemplo, el JO3 de MIXE comentaba que está desarrollando un sistema

informático en las instalaciones del cliente, pero que ello no significa una interacción continúa con el cliente, pero informalmente si intercambia dudas con respecto a las tareas del programa, o bien hace comentarios con respecto a los avances del proyecto:

> "El cliente se podría decir que participa en todo el proceso, pero no directamente con el programador, sino que participa con el arquitecto y con el administrador del proyecto. Con el arquitecto por que el cliente es el que tiene las necesidades, y el arquitecto es el que tiene que ver como el sistema va a solucionar esas necesidades. El administrador es el que tiene que ver con el cliente, en las juntas, las reuniones, los entregables, ver si hay alguna modificación por que como los proyectos no son tan cortos a veces se extienden, pueden producir modificaciones en cualquier momento para ajustarse a la realidad, entonces todo eso lo tiene que ver el administrador del proyecto con el cliente. Aunque nosotros estamos en las instalaciones del cliente, no tenemos ninguna relación de trabajo formal con ellos sino que es a través de terceras personas (Programador MIXE JO3).

Ahora bien, esta relación con el cliente, como señalábamos, depende de la complejidad del proyecto, "no hay recetas únicas" para determinar la cantidad de tiempo que requerirá algún modulo o tarea. Por ejemplo, el programador GG3 de la empresa MIXE participó en un proyecto en el cual tan sólo para la recopilación de información se requirió de cuatro equipos de trabajo, del cual GG3 era coordinador. Motivo por el cual tuvo una estrecha relación con el cliente:

> "tuve mucho contacto con el cliente (...) tenía juntas cada ocho días de revisión, de mis objetivos, cambios de esos objetivos les pedía entregables, se los revisaba, "sabes qué cámbiale, OK esta bien, ahora vamos a pasar a la siguiente tarea". Entonces sí tuve mucha relación con el cliente (programador MIXE GG3).

Podemos decir que la preocupación central en este punto es la comprensión e interpretación lo más puntual posible de las necesidades del cliente y usuarios finales:

> "Pero en proyectos grandes sí se presta a que surjan cambios, a que surjan interrogantes del cliente... tu mismo dices ¿aquí qué voy hacer? ¿Cómo le voy hacer? Haber...vas con el cliente y tu mismo también le propones al cliente oye sabes qué, ahora hay que hacer esto, "no es que esto no", también igual se dialoga para el punto, el objetivo que se vaya a cambiar o algo (programador MIXE GG3).

La comunicación con el cliente es importante, porque en caso de que se dejen pasar por alto "detalles" es cuando se inician los errores de diseño y funcionalidad. Por ello es

importante que se aprueben los requerimientos y se evalúen los módulos que se entregan, es decir los módulos que se van liberando. Ahora bien, también es cierto que la descripción de los requerimientos depende de la experiencia de los clientes al momento de describir las necesidades, de que el cliente conozca el contexto que desea informatizarse, de otra forma se acumulan una serie de contingencias. Al respecto líder de proyecto RF1 señalaba:

> "el cliente sabe lo que quiere, pero no sabe lo que necesita. Entonces tenemos que escuchar al cliente con atención, para entender que quiere, ya que lo comprendimos en serio a cabalidad, a profundidad (...) sólo entonces podemos ir nosotros trabajar en la definición de cómo vamos a construir una nueva función (...) para satisfacer las necesidades del cliente (Líder DYYA RF1).

Con el fin de minimizar la incertidumbre en cuanto a la licitación de los requerimientos, la ingeniería del software ha desarrollado herramientas[82] y metodologías como CMMI; UML; PMI, entre otras, e incluso se propone una ingeniería de requisitos (IR). Consideramos que existen otros aspectos complementarios que deben de tenerse en cuenta al momento del levantamiento de requisitos: soporte técnico del cliente (hardware), inter-operabilidad con otros sistemas informáticos que posee el cliente, presupuesto, limitación de tiempo para la implementación, cultura informática de los usuarios finales, tamaño, estructura y crecimiento del cliente en el mediano y largo plazo, etc. Complementos que no deben ser ignorados.

7.3.2.- Empresas tipo 2

En estas empresas, la ingeniería del software está presente entre los directivos de las empresas, empero reconocen que implementar las prácticas señaladas por la IS es costoso, optando por implementar estrategias de desarrollo acorde con el *background* acumulado por la empresa (empresas como depositaría de conocimientos) y por los programadores (Recursos humanos, generadores de interacciones sociales en el flujo de aprendizajes). Es verdad que no dejan de reconocer las empresas del tipo 2 como esencial está fase de contacto con el cliente,

---

[82] "Software Requirements: Objects, Functions and States", Davis, A. M., Prentice Hall, Englewood Cliffs, NJ, 1993 (pg. 27); Alexander, Ian. Requirements Engineering Tool Vendors and Freeware Suppliers. http://easyweb.easynet.co.uk/~iany/other/vendors.htm ; IEEE Software Requirement Engineering, Second Edition, Editado por Richard H. T. y Merlin Dorfman, IEEE Computing Society, New York, NY. 1997; Wiegers Karl E. Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. Microsoft Press, 2003. entre otros.

porque es ahí donde recogen las impresiones de que tipo de desarrollo están solicitando el cliente:

> "...nuestra metodología es básicamente el análisis, el contacto con el cliente, con esto empezamos a visualizar como cumplir con sus requerimientos pero eso también en base a todo lo que tenemos atrás, ósea ¿ya tenemos desarrollados algunos sistemas? y, en base a esos sistemas empezamos a visualizar la solución que se le puede dar a cierto cliente" (Programador TADI MZ3).

Nótese el comentario del programador MZ3, quien es un artesano[83], que enfatiza que el análisis estará determinado por lo que ya se ha hecho, es decir en lo que ya se poseen habilidades, destrezas, códigos que se puedan reutilizar, componentes que permitan desarrollar y garantizar en cierto grado el proceso de desarrollo. En este mismo sentido, el gerente RS0 de la empresa confirma lo dicho por el programador-artesano al decir que si un cliente les dijera "oye quiero un programa así y así" en el que no tuviesen experiencias previas, "es muy probable que tengamos problemas en un ochenta o noventa por ciento. Porque el desarrollo de la empresa se basa en experiencias similares" (gerente TADI RS0). Para el gerente RS0 de TADI -quien posee estudios de maestría en computación por el CINVESTAV- cuando se le preguntó que si emplea un método de análisis de requerimientos con el cliente, primero señala que si existe, sin embargo más adelante se contradice y afirma que los programadores que laboran con él son sus amigos de la universidad, poseen mucha experiencia y saben de antemano que es lo que van a hacer:

> "llevamos una cierta metodología que nos ha funcionado, es el análisis estructurado y diseño estructurado, desarrollo de prototipos (...) Sobre eso se hace una revisión, eso me lo pasa la gente que me ayuda a programar y.... digo, "ah sabes que pues esto va por aquí o no va por aquí...". Realmente no hay una metodología vamos...más a detalle... así como te lo explico, ahora la gente que me ayuda es gente que estuvo conmigo en la carrera, o sea, gente ya con mucha experiencia sabe de antemano ya como que...que resultados te va a entregar. (gerente TADI RS0)

[83] Le denomino artesano, porque MZ3 es un programador que diseña, desarrolla e implementa el Sistema informático. Existe un alto grado de confianza entre gerente y programador. Incluso el gerente le paga su salario más una comisión de la venta del sistema. Además el gerente le permite trabajar donde mejor le parezca y que tenga "trabajos por fuera".

Cuadro 24

Particularidades de las empresas tipo 2

Empresas y tamaño	Características de empresas tipo 2
SITA Micro MEKI Micro TADI Micro CATI Micro HUTI Miro	○ Desarrollos de Software comercial ○ Sólo ofrecen desarrollo a la medida. ○ Desarrollo de proyectos menos que complejos ○ Relación del Cliente con el Programador responsable del proyecto. ○ No Forman parte de Empresas Integradoras** ○ No tienen asociación alguna con empresas globales. ○ No aplican metodologías estándar de la Ingeniería del Software. ○ No tienen entre sus planes certificar sus procesos. ○ ¿Flexibilidad cognitiva? ○ No hay convenios con instituciones académicas. ○ Informalidad en los Equipos de Trabajo. ○ Informalidad en las jerarquías laborales. ○ Ambigüedad en la funcionalidad de los puestos de trabajo. ○ Mercado local y nacional. ○ No pertenecen a una Asociación de la Industria del Software.

Elaboración propia en base a las entrevistas.

En estas empresas del tipo 2 la *interacción social* con el cliente y con el líder de proyecto (cuando la hay) es cara a cara, conforme se progresa en el desarrollo del sistema informático, por lo regular los roles del líder de proyecto no son fijos, se le otorga ese nombramiento acorde a experiencia y habilidades con respecto al sistema informático que se pretende desarrollar. En este sentido, las habilidades y conocimientos no están jerarquizada entre líder y programador. Para el programador CM3 cuando se le pregunto donde radicaba el "análisis fino" de los requerimientos, ¿En el líder o en el programador?, ¿Quien es el que desarrolla los algoritmos que dan origen a dichos requerimientos?, señalo:

> "Yo creo que el trabajo fino, insisto, es del líder el que diga quiero "Que funcione así" (...) el tratar de darle al cliente lo que está pidiendo por que muchas ocasiones si no esta bien definido eso es lo que lleva a muchos problemas el proyecto, por que hay que hacer otro diseño, por que hay reprogramarlo (...) Si ya está todo definido ya a la hora de las líneas de código para que ... aunque a lo mejor puede ser una parte importante pero no creo que sea lo esencial para que el proyecto salga con éxito (programador MEKI CM3).

7.3.3.- Empresas tipo 3

En este tipo de empresas la característica sobresaliente es que desarrollan software libre, es decir, disponen de un amplio repositorio de sistemas informáticos disponibles en Internet, del tipo Open Source. Dichos sistemas informáticos están agrupados en dos grandes "familias": licencias GPL y BSD, ambas ofrecen la oportunidad de obtener un conjunto de

algoritmos en Internet y adaptarlo a las necesidades del cliente. Existen empresas virtuales que "empaquetan" los sistemas informáticos y los ofrecen en venta, no como licencias, sino como donaciones altruistas, por ejemplo Debian, Ubuntu, entre muchas otras distribuidoras. El gerente, programador y dueño JG2, de la empresa WALS comentaba que los clientes que buscan más este tipo de empresas de software libre son aquellos que requieren sistemas de monitoreo, administración de servidores, capacitación de personal en el uso de sistemas libres (Líder WALS JG2). Sin embargo, en lo relativo a la implementación se le cuestiona cual es el proceso que se sigue cuando un cliente solicita el desarrollo de programas informáticos: al respecto nos dice:

> "El proceso aquí seria evaluar la necesidad del cliente, que tiene funcionando, si hay algún sistema que actualmente ya hace algo o de plano no tiene nada (...) ver el modelo de trabajo que sigue (...) evaluar si hay un software OPEN SOURCE que satisfaga su necesidad, evaluarlo haber si existe, hasta que grado satisface la necesidad del cliente, si se puede instalar tal cual esta o si hay que hacerle modificaciones, hay que hacerle mejoras, hay que, en este caso rediseñar la base de datos, rediseñar la interfase, agregarle nuevos módulos, cambiar la funcionalidad un poco y en base a eso ya se le puede preparar una propuesta al cliente, decirle sabes que: hay esto, se te puede adaptar, te va a costar tanto, lo vamos a tener en tanto tiempo, funciona así y así; presentarle probablemente un demo del software si es que ya existe para que lo vea y presentarle la propuesta; tanto tecnológica como de servicio, obviamente como económica, para que vea cuanto va a invertir, que va a recibir y que beneficios va a obtener con ese sistema" (Líder WALS JG2).

Cuadro 25

Particularidades de las empresas tipo 3

Empresas y tamaño	**Características de empresas tipo 3**
LIES Micro WALS Micro BOCS Micro UFOS Micro BICS Micro DINS Micro	○ Desarrollos de Software Libre* ○ Sólo ofrecen adaptación a la medida. ○ Desarrollo de proyectos menos que complejos ○ Menor interacción del Cliente con el programador responsable del proyecto. ○ No Forman parte de Empresas Integradoras** ○ No tienen asociación alguna con empresas globales. ○ No aplican metodologías estándar de la Ingeniería del Software. ○ No tienen entre sus planes certificar sus procesos. ○ Mayor flexibilidad numérica. ○ No hay convenios con instituciones académicas. ○ Informalidad en los Equipos de Trabajo. ○ Informalidad en las Jerarquías laborales ○ Ambigüedad en la funcionalidad de los puestos de trabajo ○ Mercado local y nacional. ○ No pertenecen a una Asociación de la Industria del Software.

Elaboración propia en base a las entrevistas.

Una ventaja sobresaliente de este tipo de empresas es que adecuan, modifican, transforman, módulos o algoritmos disponibles en la red para implementar un sistema informático acorde a las necesidades del cliente. Lo destacable es que este software ya esta probado, ya cuenta con implementación en otras ciudades, empresas, etc. Por tal motivo el mantenimiento y actualización de nuevas versiones -en caso de que no se haya modificado del todo por parte de la empresa que implementa- éste se puede acceder de forma gratuita a través de Internet. Así mismo, este tipo de Software representa un "*prótesis cognitiva*" en el sentido que programadores de otras partes del mundo le darán servicio, actualización y mejoras e incluso el mismo cliente puede mejorar éste sistema informático, ya que cuenta con el acceso al código fuente.

> "el software libre precisamente al no ser un producto de una empresa que va a desaparecer, sino que es un producto que esta ahí y que es de la comunidad y que si yo no estoy, llega alguien mas y lo sigue desarrollando y, llega alguien mas empieza a colaborar pues el producto sigue evolucionando (...) (Líder WALS JG2).

7.4.- Procesos internos oscurecidos por las estructuras: Transformación de la información

Cuando el diseño de los requerimientos[84] es presentado por el arquitecto de software ó ingeniero de software, o bien por un programador que esté al frente del área de diseño, es cuando se inicia en parte la *interacción signica* en el proceso de trabajo del programador, es el punto en el cual la lista de requerimientos o tareas es definidas en módulos que integran el sistema, dicha información acumulada en *flujos de información* debe ser "traducida" a símbolos, iconos, algoritmos que resulten en una tarea o acción asertiva que demanda el módulo. Este proceso lo podemos definir como una *interpretación signica* la cual no es espontánea, racional o prediseñada; por el contrario esta contextualizada en una serie de

[84] En la etapa del diseño, Pressman y otros expertos en Ingeniería del Software, proponen una serie de modelos, sin embargo, la mayoría coincide en que el diseño debe: Considerar diferentes enfoques alternativos acorde a la Lista de Requerimientos. Es decir, no caer en la práctica de realizar el método de la Empresa desarrolladora; No inventar nada que ya este inventado. Hace referencia a la reutilización de código, subrutinas, incluso módulos, librerías, etc. que ya se hayan desarrollado previamente por la empresa, incluso si ya están a la venta; Que la modularidad del Sistema presente uniformidad, integración, fiabilidad; Admitir cambios, que no se trate de un diseño cerrado; Querer solucionar con código, no es diseñar; Valoración continúa del Diseño, en el proceso de desarrollo del Software; Revisar continuamente el Diseño, para evitar errores de concepto; Reducir en lo posible lo abstracto del diseño.

acciones colectivas, individuales, virtuales, emocionales, de conocimiento formal e informal, grados de experticia, habilidades cognitivas, destrezas, "timing" mental,[85] etc.

El objetivo del presente apartado entonces es descifrar las contingencias inherentes al proceso de la *interpretación signica* de los símbolos dispuestos en el diseño de requerimientos o en las necesidades del cliente y la transformación de éstos símbolos en signos a través de otros signos que son los lenguajes de programación; luego entonces, los *textos signicos* serían aquellos conjuntos de algoritmos que constituyen la *interpretación signica* resultante del análisis de requerimientos entre el cliente y la empresa desarrolladora.

La transformación de la información dispuesta en el diseño o lista de requerimientos a *textos signicos* que son simbolizados en líneas de código, constituye una serie de *interpretaciónes signicas* que están embebidas en un contexto de incertidumbres y contingencias inherentes al proceso de trabajo. Por ejemplo, en el desarrollo del software a la medida, según expertos como Halstead (1977), Jones (1981)[86], Toval, A., Nicolás, J. y Moros. B. (2001), Pressman, (2001)[87], entre otros, señalan que existen una serie de técnicas y metodologías para el desarrollo de software eficiente, los cuales no son implementados e incluso son desconocidos por muchos Ingenieros de Software. Sin embargo los autores consideran que la implementación de metodologías y métricas de calidad en el proceso de trabajo es cumplir con los procedimientos que estén contenidos en el diseño y debe ajustarse a los requerimientos solicitados por el cliente. Los requerimientos hacen las veces de una guía que puede ser interpretada por quienes crean los *textos signicos*. Es decir, las metodologías y métricas que se implementen deberán proporcionar información a los programadores con respecto de que contenidos se crearán, enfocándose en la estructura de los datos, funcionalidad, fiabilidad y cumplimiento de los requisitos solicitados por el cliente.

---

[85] Entenderemos por esos momentos de "lucidez o inspiración" de los programadores para resolver un problema. Lo contrario sería lo que en el argot de programación se conoce como "me cicle" que hace referencia a que el programador NO resuelve un problema.

[86] Citados en Software Engineering Institute. Requirements Engineering Project. Requirements Engineering and Analysis Workshop Proceedings (CMU/SEI -91 - TR-30). Pittsburgh, PA: Software Engineering Institute. Carnegie Mellon University. 1991. http://www.sei.cmu.edu/director/aboutSEI.html

[87] Toval, A., Nicolás, J. y Moros, B. Siren: Un proceso de Ingeniería de requisitos basado en reutilización. Jornadas de ingeniería de Requisitos Aplicada, Sevilla, pp 129-143. 2001. (Internet, citar) Pressman, R. S. Ingeniería del Software: Un enfoque práctico. Quinta Edición, McGraw Hill, Madrid. 601 p. 2002.

7.4.1.- Empresas tipo 1

Independientemente del método que se utilice, la *interpretación sígnica* exige que quienes interaccionan en el proceso de trabajo resuelvan el problema planteado, independientemente del procedimiento que utilizaron. El proceso de *interpretación sígnica* se sujeta a *tiempos y pensamientos*, que no sólo reivindican las habilidades profesionales y fiabilidad cognitiva del programador, también son importantes las *destrezas en el aprendizaje* (proceso de desaprender rutinas y reaprender nuevas rutinas) como son: habilidades personales de comunicación, tener un carácter disciplinado y atributos importantes para el intercambio de ideas y búsqueda de información y solución de problemas; ya que un proyecto de software requiere de un intenso intercambio de información, que en muchas ocasiones se desarrolla bajo condiciones de incertidumbre.

En párrafos anteriores, el líder de proyecto y gerente de capacitación de CISE JH0 comentaba que el desarrollo de un proyecto no se puede auxiliar de un solo proceso, no existen recetas, cada sistema tiene sus propios objetivos, dominios (P15 CISE JH0), es decir que dependen de la complejidad del proyecto, las rutinas o tareas a desarrollar. El gerente JH0 señala que una forma de controlar la incertidumbre en el desarrollo de un sistema, es el de verificar que se cumplan los sistemas de calidad de proceso (SQA), como los sistemas de calidad del producto (PQA) e implantar un metodología probada.

> "(...) mi puesto es gerente de procesos de software y cuido realmente que la mejora continua de la metodología se aplique correctamente en los proyectos (...) ahorita estoy llevando la iniciativa SQA de la empresa y PQA. El SQA es aseguramiento de calidad. Digamos es el aseguramiento del proceso (...) y PQA es aseguramiento de calidad del producto, uno cuida la calidad del proceso con el que se desarrolla y el otro cuida la calidad del producto, que no tenga defectos" (gerente CISE JH0).

La empresa CISE implementa una metodología propia, basada en *buenas prácticas*, método que le permitió a la empresa obtener el certificado en CMM nivel 3 (Modelo de Capacidad de Madurez por sus siglas en Ingles):

> "no es la metodología que propone CMM (...) Nuestra metodología está apegada a ese modelo y, realmente nuestra metodología es una arquitectura que tiene cuatro metodologías. Una metodología para administración de proyectos, una metodología para ingeniería de software, una metodología para la calidad o aseguramiento de la calidad del software y una metodología de ingeniería de procesos (gerente CISE JH0).

Al respecto el gerente JH0 de CISE, comenta, que ésta no es una contradicción con lo que decía en el párrafo anterior, en que no hay un solo proceso, un solo camino, una sola receta, lo que debe haber es una metodología que guíe el proceso de desarrollo del sistema, que sea sistemática y disciplinada en su aplicación (máximas de la Ingeniería del Software):

> "estas promesas de metodologías se forman de disciplinas, una disciplina es algo que se mapea como un área de procesos de CMM, por ejemplo, una disciplina que tenemos nosotros, es la administración de requerimientos y ahí hay toda una serie de procesos, buenas prácticas[88], actividades, todo lo que se tenga que ver. Otra es análisis y diseño y, allí hay toda una serie de cuestiones. A lo que yo me refería es que no dictamos un flujo en específico, de cómo se tienen que aplicar esas disciplinas, porque estaríamos cerrando a que a fuerzas trabaje así la empresa, y no puede ser así porque cada proyecto se ataca de diferente manera, inclusive depende del ciclo de vida, si escojo un ciclo de vida en cascada, a la mejor ese es secuencial en la utilización de las disciplinas; pero si yo escojo un ciclo de vida iterativo - incremental, es otra manera de como se buscan las disciplinas, no puedo...tengo que ser flexible, no me puedo cerrar a que las disciplinas se apliquen de una sola manera" (Gerente CISE JH0).

El gerente y líder de proyecto JA2, de la empresa MIXE, señala que, no sólo es importante repetir la metodología que mejor funciona en el proceso de desarrollo del sistema informático, también acuden a la asociación tecnológica, ya sea como integradora que están afiliados (*Qa*) o bien, acuden con un socio tecnológico -que en este caso es Microsoft- para que les imparta un curso " a la medida" en caso de que el proyecto a desarrollar sea complejo.

> "En ese momento en el que nosotros (...) empezamos a determinar, a ver cuáles son las metodologías que mejor nos han llevado a cabo para tener éxito dentro de lo que es el desarrollo del software. Es ahí lo que te decía, cuando MIXE y SOLG (socios de Quataria) se unifican y entran, en lo que es el programa o vínculo con el socio tecnológico, que es Microsoft" (Líder MIXE JA2).

Ahora bien, las metodologías de calidad en el producto, en el proceso, ya sean de Microsoft, Oracle, PMO, CMM, etc. contienen una fuerte dosis de propuestas en los procedimientos del como debe "dividirse" el proceso de desarrollo del sistema informático. Al respecto, por ejemplo CISE, que es una empresa con más de 24 años en la localidad, con una fuerza de trabajo superior a los 300 programadores, con metodologías ya probadas, con una

[88] Buenas practicas es un concepto que hace referencia a que en el proceso de trabajo se debe de implementar una especie de "un solo camino en el saber-hacer" haciendo referencia a que si bien es complejo el proceso de creación de sw, este debe seguir métodos y procesos y en la mediad que se sigan éstos, se obtendrá un sw seguro y de calidad. estas son las buenas prácticas.

estructura en equipos de trabajo, flexibles y dinámicos, certificada n nivel 3 de CMM (Gerente CISE JH0) señala:

> "la estructura organizacional es, en primer instancia una organización matricial, y es así precisamente para poder administrar todos los tipos de proyecto que tenemos aquí. Digamos que tenemos, actualmente (...) 3 ramas principales (...) La primera es desarrollo de software a la medida, en donde ahí ... un equipo de trabajo que esta asignado por un grupo en especifico con una planeación, productos a entregar etc. ... y ahí es donde generalmente se utiliza la metodología con un ciclo de vida muy bien definido etc. Luego tenemos proyectos de mantenimiento... ahí tenemos asignada gente a un cliente que atiende requerimientos menores y mayores y, ahí también tenemos... nuestra metodología realmente es un repositorio de una buenas practicas digamos, que sí están divididas por diferentes disciplinas o áreas del proceso que marca CMMI, pero que no marcan una manera en especifico de cómo ejecutarlas, ... porque a lo mejor al afinar el tipo de proyecto no tengo que hacer requerimientos...o a lo mejor si es de mantenimiento, no aplico como tal, la arquitectura, por que la de arquitectura ya esta definida, entonces ya nada mas instancio que tengo que revisarla, y tengo que agregar nuevos componentes. Entonces por eso, no hay... una receta de cocina específica, porque va a depender del tipo de proyecto... Luego otro servicio que tenemos la parte de consultoría, puede ser consultoría para generar sistemas, consultoría en tecnología, consultoría en arquitectura, consultoría en procesos inclusive, ahorita tenemos proyectos importantes que son este.... donde apoyamos algunos clientes tan bien en definir sobre su metodología..." (Gerente CISE JH0).

El conjunto de empresas del tipo 1 coinciden en que se deben de implementar las metodologías que dan cuerpo a la ingeniería del software. El líder de proyecto JA2 y actor clave en conformar uno de los "ejes" académicos Industria-Universidad en el valle de México de la empresa MIXE, insiste en utilizar las ultimas tecnologías blandas de organización UML en el proceso de desarrollo de un sistema informático.

> "Aquí es muy sencillo, te digo que se utilizan las ultimas tecnologías son las que te recomiendan las mejores practicas para que tu puedas estarte organizando bajo el esquema de administración de proyectos: una buena arquitectura de proyectos y diseños. Y de ahí...con la parte de UML, se le envía a los desarrolladores, y como es un lenguaje universal (UML), pues todos lo captan y empiezan a generar el código, por eso nosotros podemos estar asegurando que trabajan bajo el mismo estándar todos y cada uno.... pueden terminar un modulo mientras que los otros esta haciendo otro modulo, cuando terminan eso le ayudan a los que no han terminado. Tienen esa capacidad de rápidamente integrarse a tareas para poderlos sacar en forma paralela todo el proyecto" (Líder MIXE JA2).

El líder JA2, al igual que JH0, nos dice que la división social del trabajo es importante para un mejor control de la calidad y del conjunto de la cadena de desarrollo:

> "Hay un departamento de aseguramiento de calidad y lo que es testing (...) El testing son jóvenes que se dedican exclusivamente a hacer todo el tipo de pruebas que se requieran para que la funcionalilidad de cada uno de los módulos o de los programas o todo con lo que estés trabajando, este funcionando de una forma integral. Y a parte el aseguramiento de calidad desde que tengan un rendimiento o sea el performance... en la forma en que se esta dando el resultado sea oportuna, que sea veraz, que sea claro para el usuario. Entonces ese tipo de aseguramiento de calidad va desde los procesos, la forma de llevar a cabo el desarrollo del código y los beneficios o las bondades que le estas proporcionando a la parte del usuario. Entonces cada uno de ellos ya tiene tareas muy especificas y con eso te esta asegurando el aseguramiento de la calidadW" (Líder MIXE JA2).

En esta división social, la postura de los líderes es tener el control del *saber hacer* del programador, este discurso del control lo observamos cuando se les pregunta a los directivos que nos dijeran cual era su apreciación del programador dentro del proceso de trabajo:

> "Yo veo al programador como un componente, en realidad tiene que haber una persona que analiza los procesos, una persona que define, analiza y diseña los procesos, una persona que diseña y define la base de datos, otra persona que programa los procesos que se han establecido y otra persona que genera las salidas: consultas, reportes etc. (...) (gerente DYYA RF1).

Otras empresas tienen equipos de trabajo flexibles, por ejemplo MIXE y CISE, nos decían líneas atrás, que si un programador terminaba su modulo podía integrarse rápidamente a otro modulo, esta *flexibilidad operativa* se explica en el hecho de que los programadores conocen los módulos que integran el proyecto, conocen cuales son las directrices que "cosifican" el proyecto, de tal forma que puedan integrarse a otro modulo, o bien entre equipos de trabajo. Definiré *flexibilidad operativa* como la capacidad de un programador de generar líneas de código para un modulo X y desplazarse a otro modulo para continuar generando algoritmos para el modulo X2 o bien modulo Y.

7.4.2.- Empresas tipo 2

Al igual que para las empresas del tipo 1, la incertidumbre en el diseño y desarrollo del código (*interpretación sígnica*) está dada por el hecho que si el *texto sígnico* refleja los requisitos señalados por el cliente o, en su defecto, la interpretación es deficiente (doble hermenéutica).

Para el líder de proyecto DC2 de la empresa CATI, el problema en la falta de cumplimiento de los requerimientos, se debe a que no se tiene personal capacitado, con suficiente experiencia y habilidades para generar *interpretaciones sígnicas* eficientes (P21 CATI DC2). Para el programador y líder de proyecto CM3 de la empresa MEKI el trabajo más eficiente de un líder de proyecto consiste en proporcionarle al cliente lo que necesita:

> "Yo creo que el trabajo fino, insisto, es del líder el que diga quiero "Que funcione así", (...) tratar de darle al cliente lo que está pidiendo, por que muchas ocasiones si no esta bien definido, eso es lo que lleva a muchos problemas el proyecto, porque hay que hacer otro diseño, por que hay reprogramarlo, cuesta mucho más esfuerzo" (Programador MEKI CM3).

Más adelante este mismo programador reconoce que en la empresa él ha visto que sucede seguido que no se entregan a tiempo y no cumplen los requisitos del cliente los sistemas informáticos que desarrollan:

> "...si por ejemplo el líder nos pasa ese requerimiento con fallas, que no este bien definido entonces lo hacemos pero ya después nos lo regresan "No, sabes que es que así no, necesito esto y esto" Entonces uno lo tiene que volver a hacer, entonces es como que más trabajo para uno como programador... sería que el proyecto no se entregue a tiempo y a veces no cumplen con los requisitos es lo que más constantemente se da aquí en la empres"(Programador MEKI CM3).

Son varios y diversos los argumentos por los cuales no se puede entregar a tiempo un proyecto, no cumplir los requisitos. Para el gerente RS0, con maestría en el CINVESTAV y dueño de la empresa TADI, con una antigüedad de más de 15 años en el desarrollo de sistemas es la deficiente evaluación de los requisitos que se hace del sistema a desarrollar:

> "...el problema de no entregar a tiempo los sistemas, no es otro más que la mala evaluación (...) que se hace para hacer una propuesta. Estas hablando de que realizas unas entrevistas, un cierto bosquejo, un diseño y... con base a esa experiencia pues evalúas el tiempo o las horas de programación que se requieren, entonces esa evaluación... es muy importante y es la que te da pauta para que puedas entregar a tiempo un desarrollo (gerente TADI RS0).

No pasemos por alto que la evaluación es un sistema heurístico entre cliente-desarrollador-cliente. Lo que nos lleva a preguntarnos ¿Hasta donde influyen las capacidades y habilidades de quien entrevista?. Para los gerentes y líderes de proyecto RS0 y DC2, es el cliente quien no proporciona la información suficiente.

> "(...) el cliente también muchas veces no recibe lo que realmente estaba solicitando. En primera instancia a lo mejor el diseño es muy bonito y dice: ¡ah

está padrísimo !. Lo que pasa es que no sabe (...) a la hora de que funcionen con veinte usuarios accesando a la base (de datos) al mismo tiempo truena" (Gerente TADI RS0).

Otro gerente, señalaba lo mismo para con el cliente:

"De un proyecto pequeño, realmente los principales problemas surgen por el mal diseño, el mal diseño surge por la mala información que proporciona el cliente y al enfrentarla a la realidad es cuando surgen ya los cambios específicos" (Gerente CATI DC2).

Para la Ingeniería del Software, la incertidumbre en el diseño, funcionalidad y fiabilidad de un programa de software, reside en la falta de implementar metodologías ya probadas, así como una falta en seguir un plan y buenas prácticas en el proceso de desarrollo. Por ejemplo, nótese que el Gerente y Líder de proyecto RS0 nos explica que si bien es cierto es importante una organización mínima en el proceso de trabajo:

"llevamos una cierta metodología que nos ha funcionado, es análisis estructurado y diseño estructurado, desarrollo de prototipos. ¿Qué es esto? Es un análisis, es una recopilación de los requerimientos. Diseño, pues hacemos ciertos diagramas... y lo que sería la propuesta del sistema en sí. Y el manejo de prototipos es decir básicamente esto sería el primer prototipo" (Líder TADI RS0).

En este mismo párrafo reconoce que no existe una metodología que se aplique al detalle, ya que la confianza a los programadores, que además fueron sus compañeros de universidad, le permite tener certeza que el diseñó y análisis que se haga será el correcto. Para esta empresa TADI, el diseño y análisis, así como la organización del trabajo, está mas en función de la experiencia acumulada de los programadores de la empresa misma. La acumulación de habilidades a partir de programas ya desarrollados por la empresa, de tal forma que podemos decir que la empresa a través del tiempo ha acumulado una serie de repositorios de algoritmos, una especie de "librería electrónica" que el programador puede reutilizar códigos:

"si alguien nos dice: oye quiero programa así, así, es muy probable que tengamos problemas en un ochenta, noventa por ciento. Entonces si yo le digo a Martín (que es uno de nuestros programadores) oye sabes que aviéntate esto, "ah pues si, haber cómo esta, que análisis, aja pues sabes qué, esto ya lo tenemos". Pues estás hablando de que todo ese software ya lo conocemos o esos módulos por así llamarlos, ya los conocemos. Entonces implementarlos no es ya tan difícil. No se está hablando de algo nuevo (...), si no hay una buena

documentación el mantenimiento, cuando entra gente nueva, pues va a ser más difícil, pero como te digo es ya mucho aquí los años que hemos tenido de desarrollo de los módulos que tenemos para nosotros ya no es tan problemático es parte" (Gerente TADI RS0).

El gerente RS0 no cuenta con una división del trabajo establecida, si bien señala que es el método de análisis estructurado y entrega de prototipos, reconoce que es una organización horizontal, en el sentido que es el cliente, él como gerente y el programador quienes diseñan y establecen los requisitos. Distinto como nos comenta el gerente DC2 de la empresa CATI, quien señala que la forma de trabajo como se organizan es:

"...la forma de trabajo es dividida en tres: Arquitectos, Desarrolladores y Documentadores. Existen otras divisiones que se pueden manejar pero yo así la manejaba. Los arquitectos se encargaban de la programación de alto nivel, de la programación de integración. Los programadores se encargaban de la parte repetitiva, de la programación. Los documentadores se encargaban de las pruebas y la documentación, para esto la tecnología que adoptamos es trabajar sobre UML para poder diseñar todos los componentes o sea esto le lleva al estándar prácticamente RUB, de hecho en el *file process*, que es la forma de llevar a buen termino un desarrollo y documentarlo correctamente por que uno de los mayores problemas de los programadores siempre es documentar. Nunca hacen ese trabajo bien" (gerente CATI DC2).

Podemos indicar que la experiencia, habilidades y destrezas en y de los programadores es fundamental para el desarrollo del sistema informático. El informante clave RS0, primero expuso a detalle lo que era una metodología estructurada, sin embargo a medida que avanzó la entrevista reconoció que "te soy honesto, no tan a detalle eh, no tan a detalle, nunca ha existido", se basa mas en la confianza con la que se interactúa, ya sea con el cliente o con el programador. (P20 TADI RS0) que en una estricta metodología. Otro aspecto, importante de las empresas tipo 2, es el numero de programadores que se pueden mantener en nomina, por ejemplo la empresa CATI (entrevistada en Mayo de 2005) para Agosto de 2006 (segunda visita) había suprimido el área de desarrollo, bajo el argumento que era muy costoso mantener una nomina de desarrolladores. El gerente y ex líder DC2 nos comento que la estrategia de la empresa y, es la tendencia mundial subcontratar los procesos de desarrollo.

7.4.3.- Empresas tipo 3

Una de las singularidades de las empresas tipo 3 es que desarrollan los programas con software libre disponible en Internet; sin embargo, el diseño y desarrollo de software cumple aparentemente el mismo proceso que el de la empresas tipo 2; con la particularidad que el Gerente y/o Líder de la empresa, una vez que define los requerimientos del cliente, puede obtener módulos e incluso proyectos similares de Internet, y presentárselos al Cliente. En otras palabras, existe un amplio abanico de posibilidades que el desarrollador reutilice módulos disponibles y le presente al cliente como un prototipo funcional, sin embargo, aún así persisten las condiciones de la *interpretación sígnica*, aúnque los *textos sígnicos* estén disponibles en *open source:*

> "...lo que haces es implementar un software que ya estaba desarrollado y a lo mejor probado y hasta funcionando en otros lados, de ahí puede saltar a ser una empresa de desarrollo, cuando digamos ya se encuentran con que a cierto software que ellos estaban implementando hay que hacerle nuevos cambios, que lo pueden adaptar digamos a la manera de trabajar de la mayoría de los clientes (...) y entonces aquí la lógica seria, si se puede hacer las dos cosas: desarrollar y dar la consultoria, porque van de la mano, (...) Pero vamos a decir que la tendencia ahora ya no solamente es mover cajas, es no mover caja de hardware y de software, sino darle servicio al cliente, darle desde "haber tu en vista de las necesidades que tienes necesitas este producto o este hardware o software y te conviene implementarlo así"... y llevar al cliente de la mano, desde sugerirle, a lo mejor venderle, instalarle, configurarle, capacitarlo y ver como va trabajando, ese es el nuevo modelo, ya no solamente es... me dedico a vender, a comprar y vender, ¿por qué?, porque el cliente ya te dice que tu les des la solución precisa. (Líder WALS JG2).

La forma en que se organizan las empresas tipo 3 se inicia porque el encargado del proyecto, que por lo regular es el gerente y un programador con experiencia, quienes evalúan las necesidades con el cliente:

> "El proceso aquí seria evaluar la necesidad del cliente, que es lo que requiere, que ya tiene funcionando, si hay algún sistema que actualmente ya hace algo o de plano no tiene nada, si hay que automatizar todo (...) la segunda parte seria evaluar si hay un software OPEN SOURCE que satisfaga su necesidad, evaluarlo haber si existe, hasta que grado satisface la necesidad del cliente, si se puede instalar tal cual esta ó si hay que hacerle modificaciones, hay que hacerle mejoras, hay que, en este caso rediseñar la base de datos, rediseñar la interfase, agregarle nuevos módulos, cambiar la funcionalidad un poco y en base a eso ya se le puede preparar una propuesta al cliente. Decirle sabes que: hay esto, se te puede adaptar, te va a costar tanto, lo vamos a tener en tanto tiempo, funciona así y así; presentarle probablemente un demo del software si es que ya existe

para que lo vea y presentarle la propuesta; tanto tecnológica como de servicio, obviamente como económica, para que vea cuanto va a invertir, que va a recibir y que beneficios va a obtener con ese sistema" (Líder WALS JG2).

En este proceso, nos preguntamos ¿Hasta donde es una ventaja el implementar Software Libre? O bien ¿Qué tipo de contingencias representa el utilizar Software libre existente en la red?[89] Ya sea de mantenimiento, actualización o implementación:

"Exactamente ésta es una de las bondades del software libre, que no es un sistema cerrado, al cual la empresa tiene que adaptarse, sino al ser un sistema abierto precisamente es una de las ventajas del software libre, es que yo puedo adaptarlo a mis necesidades, tengo acceso al código fuente, al diseño de la base de datos, a la documentación, tengo a parte la experiencia digamos de otras implementaciones de otros usuarios que se han dedicado a lo mejor al mismo proceso de instalación, configuración y adaptación y... yo me retroalimento también de esa información, entonces lo que hacemos es adaptar el software a las necesidad de la empresa y no al revés, porque ese a sido uno de los defectos del software cerrado, que la empresa tiene que a veces utilizar un campo que no es para registrar un dato o a lo mejor no se ajusta a su proceso de producción, a su proceso de venta, entonces es donde viene el conflicto entre cómo lo hacemos y cómo me lo pide el sistema para registrarlo o para poder llevar a cabo el proceso. (Líder WALS JG2)

La incertidumbre aún prevalece en la implementación de programas basados en software libre:

"Varios tipos de pruebas y muchas de ellas, por premura se llevan...hasta la fase de producción, pero la principal validación es que el sistema haga lo que debe de hacer... en las pruebas de realización, para hacer la entrada y debe tener tal salida, después haces pruebas de... VAYON ... pero igual tiene limitaciones, pruebas las áreas que se te ocurren, (...) y por ultimo, las pruebas de eficiencia... el sistema tiene que entregar el resultado que debe de entregar, en un tiempo aceptable, con un consumo de recursos aceptables y ahí entras en varias iteraciones de optimización, incluso en algún momento te debes de asesorar con otras personas de igual especialización, que están en menor contacto con el sistema, que puedan recomendar alguna otra manera u otras maneras de optimizar un procedimiento que este haciendo "cuellos de botella". (Programador DINS 03).

Según el experimentado líder de proyecto y programador JG2, los cambios en el software disponible en la red no se le hacen muchos cambios, de tal manera que, a través del tiempo, es más sencillo modificar, mantener y localizar quien asesore a la empresa:

[89] Estas dos preguntas nos pueden guiar para conocer las diferencias cualitativas entre swp y swl.

"Te puedo decir que realmente las modificaciones son digamos ajustes, como... con que servidor va a trabajar, con que lenguaje va a trabajar, algunas personalizaciones como el nombre de la empresa, el logo, a lo mejor quitar algunas secciones, cambiar algunos colores, realmente no es muy significativo los cambios que hacemos en software libre a la hora de implementarlo" (P24 WALS JG 02).

7.5.- Síntesis de un proceso de trabajo ágil y fluido

La Ingeniería del Software a propuesto una serie de metodologías, técnicas, herramientas e incluso a sugerido la presencia de nuevas formas de ingeniería como son la Ingeniería de la Usabilidad o Ergonomía del Software; Ingeniería de Requisitos; Arquitectura del Software (Gonzalo, 1991; Marck y Rigby, 1994; García y Sánchez, 1994) e incluso recientemente se empieza a discutir, si es que no estamos frente a una nueva ciencia de la tecnología de la computación, versus Ingeniería del Software[90]. En términos generales la Ingeniería del Software propone que para el buen término en el desarrollo de un sistema informático deben seguirse una serie de etapas que identifiquen las necesidades del cliente, éstas necesidades deben ser traducidas como requerimientos del software que se pretende desarrollar, éstos requisitos transformados en un diseño sencillo y comprensible y el diseño se traduzca en líneas de código (interpretación *sígnica*). Donde el conjunto de símbolos algorítmicos en su conjunto respondan a las necesidades planteadas por el cliente.

Dicho proceso se le conoce como *ciclo de vida* del proyecto, planteado así, parece sencillo, sin embargo la complejidad es de tal grado, que para cada una de esas diferentes fases o etapas, se proponen una serie de nuevas formas de ingeniería. Por ejemplo, para la primera etapa, que es la de contacto con el cliente, se propone la Ingeniería de Requisitos y la Ingeniería de Usabilidad o Ergonomía del Software[91]; para la segunda etapa que corresponde a la del Diseño, se propone la Arquitectura del Software. El área propia a la generación de las líneas de código, sería el espacio propio de la Ingeniería del Software: espacio en el cual se implementarían pruebas de calidad, cumplimiento de tiempos, revisión de la documentación

---

[90] Blum, B. I. Beyond Programming: To a New Era of Design. Oxford University Press, 1996.; Cornford, T y Smithson, S. Project Research in Information Systems. A Student's Guide. Ed. MacMillan, 1996; véase los trabajos del Grupo de Investigación KYBELE, de la Universidad Rey Juan Carlos, en Madrid, España.

[91] Nielsen J. (2003). PR on Websites: Increasing Usability. March, 2003 Jakob Nielsen's Alertbox. Disponible http://www.useit.com/alertbox/20030310.html ; Nielsen, J.; Mack, R.L. (1994). Usability Inspection Methods. John Wiley & Sons, New York, NY.; Granollers, T.; Perdrix, F.; Lorés, J.; Grupo de Investigación GRIHO. Universidad de Lleida, España.

del proceso seguido en cada uno de los módulos. La ingeniería de la Usabilidad comprende aquellos aspectos relativos a la interfaz grafica del sistema informático sea comprendida por el cliente. Es decir que ergonómicamente el software se adapte a las necesidades del Cliente. Visto a grosso modo parecería así un tanto cuanto esquemático el desarrollo de un sistema informático, también expresa burocracia, rigidez y, altos costos.

> "....la compañía grande IBM que te cobra 10 veces lo que vale el desarrollo, van a tener retraso, no importa los cubren, van a tener algún problema que se les complique, no importa contratan a un programador mas talentosos.... porque... dentro del costo pagaste diez veces el proyecto; ellos pueden absorber todos los retrasos, todas las inconsistencias y sí te pueden ofrecer una garantía de terminación y entrega de satisfacción. Sin embargo... las compañías mexicanas, incluso compañías grandes no tienen dinero para pagarle ha alguien como IBM... ¿que es lo que acaban haciendo? los clientes...pues acaban cayendo con una PYME que desarrolla software, que generalmente o muchas de las veces es el primer proyecto de un tamaño grande que les traen.... y por la propia inexperiencia... lo proyectan mal.... se vuelve imposible que terminen el proyecto" (Programador DINS 03).

El objetivo de la Ingeniería del Software es que el desarrollo del Sistema Informático se entregue a tiempo, se cumplan los requisitos señalados por el usuario, se cumpla con parámetros mínimos de calidad, que el sistema funcione y se le pueda dar mantenimiento y actualización en un tiempo razonable. Sin embargo, una de las principales características estructurales de la industria es que son equipos de trabajo pequeños, localizados en empresas que, para el caso de México, son pequeñas y medianas empresas, donde en promedio entre 85 y 88 por ciento tienen menos de 50 trabajadores por empresa. Porcentaje que es similar al de la industria del software mundial. Esta descripción de las etapas del proceso de trabajo en el Software es EL TIPO IDEAL que citan la mayoría (unos mas otros menos) de los libros de la ingeniería del Software (Pressman, 2001). Sin embargo, aún en las empresas mejor organizadas, dichas etapas no están claras, son confusas. Existe una serie de irregularidades e indeterminaciones del tipo técnico y metodológico en las fases citadas.

Parte III

Configuraciones simbólicas en el saber –hacer, interacciones y prácticas sociales embebidas entre juegos y consensos, negociaciones y conflictos, saber y poder

Capitulo VIII

El saber como poder y el saber hacer como aprendizaje:
Juegos y consensos, conflictos y resistencias en el proceso de trabajo

8.1.- Introducción

En el proceso de trabajo del software a la medida del cliente es fundamental especificar la lista de requerimientos que el líder de proyecto, gerente o programador de la empresa transformará en un diseño modularizado. Baldissera señala que la consideración del cliente y/o del usuario final en el diseño y desarrollo de programas de software se presenta como una "revolución del usuario"[92] (Baldissera, 1988; en Castillo, J.J. 1993). En ultima instancia, un tercer jugador en el proceso de trabajo en la industria del software es el cliente quien participa en mayor o menor medida como un agente activo en el proceso de trabajo, desde el inicio del proceso e incluso en algunos módulos considerados como "complicados" el cliente está presente. La participación del cliente en el proceso de trabajo del software a la medida es importante para convenir sus necesidades (requisitos de diseño) y desarrollar una interfaz grafica que refleje dichas necesidades. En esta *interacción social* deben observarse: a) comprensión de las necesidades del cliente y proyectarlas en un diseño; b) diseñar dispositivos gráficos (interfaz) y procesos sencillos, prácticos; c) una interfaz gráfica con símbolos usables; d) facilidad para corregir errores, acceder información a través de iconos y ventanas, sin complejidades para el usuario; e) crear iconos alternativos al texto escrito, por ejemplo en lugar de "enviar por correo electrónico" utilizar el icono: , para que el software desarrollado represente, ejecute y procese la mayor información posible solicitada por el cliente. En este proceso se ponen en juego un conjunto de símbolos del saber-hacer y el saber como poder, se involucran una serie de *configuraciones subjetivas* que intervienen en el proceso de trabajo, como los conocimientos formales e informales, habilidades y destrezas, autoaprendizaje y disposición en la búsqueda de información; consensos y alianzas, resistencias y desobediencias; entre aquellos agentes que intervienen en el proceso de desarrollo del software a la medida.

[92]Baldissera, Alberto, 1991; Máquinas antropomórficas y mentes artificiales: interacción y cooperación entre hombre y calculador en los sistemas complejos en: Juan Jose Castillo (comp.) La automoción y el futuro del trabajo. Diseño del Trabajo y cualificación de los trabajadores, 2ª edn. MTSS, Madrid., España.

8.2.- Espacios del saber como poder

En el proceso de trabajo del software se ponen en juego una serie de conocimientos formales e informales, un conjunto de habilidades y destrezas en torno al significado de apreciar, discernir o descifrar soluciones informáticas, proceso que podemos señalar como *rutinas y juegos específicos de relaciones de poder*, entendiendo por poder el sentido que señala Foucault (1993) que es un conjunto de *relaciones de poder*, el cual no se adquiere mediante el despojo o la expoliación, no es que se arrebate o comparta; sino que existe en el momento que se ejerce, que entra en conflicto el saber y el poder[93]. Foucault (1993, 1976, [2005]) señala que el poder sólo existe en el momento de su ejercicio, cuando se busca algo y es conseguido, obtenido, logrado las relaciones de poder desaparecen, es decir el poder sólo existe en el acto, en las interacciones transitorias y continuas entre los individuos. El poder no se posee, no es monolítico, consiste en una serie de juegos específicos en el que todos los participantes actúan en diferentes niveles, con distintas intensidades e intenciones.

Para el caso del software, consideramos que cuando el cliente hace contacto el agente-desarrollador (sea el gerente, líder de proyecto, ingeniero de software ó el programador experto) indaga quien y bajo que costo desarrollarán un software, que solucione los problemas empresariales en el menor tiempo y costo posibles. El agente-desarrollador le interesa primero retener al cliente como tal, y ajustarse a sus necesidades, obviando una serie de contingencias, por ejemplo ¿Existen posibilidades reales para desarrollar el software que plantea el cliente?; ¿Posee la empresa la experiencia, recursos humanos y técnicos para el desarrollo del software que se solicita?; ¿Es viable el proyecto, tanto económicamente como técnicamente?; ¿Los requerimientos del cliente, es realmente lo que necesita?. Consideraciones que están implícitas en el proceso de trabajo y, en conducir a buen término el proyecto.

Si bien es cierto en el capitulo anterior y en lo que va del presente, hemos situado al proceso de trabajo del software como si estuviesen separadas las fases de diseño del software y la creación cognitiva de los algoritmos, es decir por un lado las entrevistas entre el administrador del proyecto y el cliente y, por otro lado, la transformación de los requisitos en líneas de código (LDC); dicha separación no es tal[94], por el contrario, coexiste un proceso

[93] Foucault, M. 1993, Microfísica del poder, Ediciones La Piqueta, Madrid, 1993.

[94] Excepto en las grandes empresas con más 1,000 programadores en las cuales las divisiones entre el área de diseño, desarrollo, calidad, documentación son mas jerárquicas, sin embargo aún así se observa que el trabajo es fluido. Castillo, J.J. (2007), El trabajo Fluido, Madrid, España.

fluido (Castillo, J. 2007) entre un proceso y el otro; al respecto el programador JO3 señala que existe un ir y venir entre el administrador del proyecto y el cliente, entre el administrador y los programadores; entre programadores y tester de calidad; proceso fluido, ágil que implica una serie de *interacciones simbólicas* entre el administrador del proyecto y los programadores:

> "El cliente se podría decir que participa en todo el proceso, pero no directamente con el programador, sino que participa con el arquitecto y con el administrador del proyecto... el arquitecto es el que...va a solucionar las necesidades, el administrador es el que tiene que ver con el cliente...juntas, reuniones, entregables, ver si hay alguna modificación, - los proyectos a veces se extienden y pueden producir modificaciones en cualquier momento- para ajustarse a la realidad..." (Programador JO3 MIXE-MT1)

La participación del cliente viene a marcar una ruptura con el tradicional proceso de trabajo de la economía fordista-taylorista, el espacio cerrado en que se convertía el proceso de trabajo, zona exclusiva entre patrón y trabajador es irrumpida por el cliente, como actor clave que participa en mayor o menor medida en el proceso de trabajo. Sin embargo, debemos aclarar que en la industria del software a la medida la participación del cliente es heterogénea, con diferentes grados de implicación. Por ejemplo, en las fábricas de software empaquetado, de uso masivo el cliente es considerado como usuario final del producto, la *práctica social* mas común entre este tipo de empresas es el *testeo masivo* o *debuging free* que consiste en la liberación del software que se está desarrollando en formatos beta (software en fase de prueba), éste es probado (testeado) tanto por otros programadores (en búsqueda de vulnerabilidades, bugs, fallas, etc.) como por usuarios invitados, para que comprueben la usabilidad o no de la interfaz grafica, así como la operabilidad interna del sistema[95] y otros posibles errores en el producto desarrollado por las fábricas de software. Sin embargo, en el software a la medida, la participación del cliente ocurre desde el inicio del diseño de los requerimientos del diseño y es quien evalúa y aprueba el interfaz gráfica del software. Al respecto, el líder de proyecto y dueño de una pequeña empresa con menos de 35 empleados, 4 programadores y 6 consultores en tecnologías, pero líder en el mercado de los ERP

[95] Beta: Es el proceso formal de solicitar información y comentarios sobre los resultados del software todavía en programación. En el desarrollo de aplicaciones, es la segunda parte de las pruebas que se realizan del software por los usuarios finales. La fase Beta en el desarrollo de un programa, se puede decir que es el prelanzamiento del software. Estas versiones son distribuidas a una gran audiencia de Internet, para llevar al programa a un entorno real de trabajo, y proveer a los usuarios un anticipo del próximo lanzamiento. Este procedimiento significa grandes ahorros para las empresas, ya que en esta etapa se identifican fallas en el sistema (bugs) que el área de calidad no localizo. Esta es una ampliación del proceso de trabajo más allá de las paredes de la empresa.

administrativos en Latinoamérica, señala que el diseño de interfaz grafica es elemental en el éxito de un sistema informático:

> "...ves interfaces de usuario, que están concebidas pensando en el dato, no en el usuario; pensando en la información, no en el proceso... llega el usuario se sienta enfrente de la computadora y para registrar un tipo de transacción tiene que navegar por dos, tres, cinco o siete pantallas para registrar una sola cosa" (Líder DYYA RF1).

El líder RF1 desaprueba que las empresas desarrolladoras de software no profundicen en torno a las necesidades graficas del usuario (interfaz gráfica) por estar concentrados en el desarrollo del código, es decir no contemplan el desarrollo del software en todo su conjunto. Al respecto Hammer y Champy (1994:35)[96] señalan que el proceso de desarrollo de un software "es una colección de actividades que toman uno o mas tipos de entradas y crea una salida que es de valor para el cliente". La *colección de actividades* en el argot de la industria del software es la participación de distintos actores en el ciclo de vida del proyecto[97] que hemos denominado con el nombre de *constelación de relaciones subjetivas* y un conjunto de *practicas sociales de aprendizaje*, que en la mayoría de los ciclos de vida coinciden en las etapas de conceptualización/formalización y procesamiento/ implementación del sistema. Lo cual no significa, según Hammer y Champy (1994), que todas las empresas de negocios de software implementen todas fases de: a) Análisis de requisitos; b) Diseño basado en un proceso c) Desarrollo de los módulos; d) implementación, sino que pueden utilizar diversos enfoques, como orientado a tareas, a personas, a estructuras entre otros enfoques como el método ágil, donde la comunicación entre agentes es fluida, horizontal, sin jerarquías, sin divisiones rígidas o formales.

Para el caso de las empresas del Valle de México observamos en las entrevistas realizadas que en el discurso se plantean tres etapas de trabajo yuxtapuestos: 1) Análisis de requisitos y diseño del proyecto, integrado por el cliente y el agente desarrollador; 2) Proceso de programación y verificación de la calidad, interactúan el diseñador, programador, tester, documentador, entre otros trabajadores cognitivos; 3) Implementación del software desarrollado, participan el diseñador, programador responsable, el cliente y usuarios finales

---

[96] Champy, J.; M. Hammer (1994): Reingeniería, Ed. Norma. España

[97] Se entiende por *ciclo de vida* al periodo de tiempo en que se concibe, desarrolla e implementa el Software. El ciclo de vida no es único -ver el capitulo 2- existen distintos ciclos de vida, por mencionar algunos: en cascada, desarrollo incremental, evolutivo, en espiral, etc. Pressman, R. (2002:45).

del programa. Sin embargo, en términos informales el proceso no esta dividido, es fluido, dinámico, ágil, sin fronteras rígidas entre cada uno de las tres etapas del proceso de trabajo.

Esquema 9
Espacios interactivos de saber y de poder
en el proceso de trabajo del software

Fuente: elaboración propia en base a las entrevistas.

En el esquema 9 se identifican tres espacios de poder en el proceso de trabajo, que ya definimos en el apartado anterior en términos de espacio mas no de relaciones de saber y poder. A continuación abordaremos dichos espacios como: i) *interacción simbólica*, hace referencia al primer espacio de poder que sucedería en la definición y especificación de requerimientos, tareas o rutinas que deberá ejecutar el software a desarrollar, participa el cliente y el agente desarrollador que formalizará dichos requerimientos; ii) *interacción sígnica*, hace referencia al segundo espacio de saber, entre el diseño de los módulos que integraran el software ha desarrollar y la transformación de la información contenida en los requisitos o rutinas en *textos signicos* que son aquellos algoritmos que resuelven el problema planteado por el cliente; iii) *interacción gráfica*, hace referencia al tercer especio de poder, son los resultados obtenidos, no en términos de eficiencia algorítmica, sino orientadas más al cumplimiento de que se ejecuten las tareas asignadas, terminar a tiempo y documentar el proceso. Así como la evaluación final que hacen los usuarios finales del software desarrollado en la fase conocida como implementación del software desarrollado (curva de aprendizaje y curva de aceptación social del software implementado).

Espacios de poder y de saber que no necesariamente están separados, por el contrario están yuxtapuestos horizontalmente, comprendidos en una *fluidez cognitiva* en el proceso de trabajo y por una *constelación de interacciones subjetivas*. Consideramos como aparente la

separación entre diseño y ejecución, porque obedece más a un discurso formal impuesto por las teorías del *managment*. Sin embargo, en las empresas tipo 1, como daremos cuenta mas adelante prevalece un discurso de una gerencia de control y poder en el proceso de trabajo, se gestiona un proceso de "restarle poder al trabajador" mediante la implementación de diversas *normas disciplinares* que aglutina los distintos aportes teóricos de las distintas ingenierías: Ingeniería del software, Ingeniería de la usabilidad, Arquitectura del software, etc. y, por *herramientas procedimentales*, agrupamos los diversos enunciados de las ingenierías, como las heterogéneas metodologías, métricas de calida, diferentes ciclos de vida para desarrollar software, etc.; En los tres tipos de empresa se yuxtaponen los espacios de poder embebidos en una *constelación de interacciones significativas* que intervienen en el proceso de trabajo.

Entre dichos espacios convergen un conjunto de conocimientos de saber y de poder; en un contexto de *fluidez cognitiva* que atañe una transformación de información en conocimientos formulados como *operaciones subjetivas* del saber, en un contexto no limitado o rígido, por el contrario en un espacio flexible, dinámico, ágil que denominamos *fluidez cognitiva,* donde los espacios definidos configuran *dinámicas relacionales del saber-hacer como poder*; que denominamos *interacción simbólica* al conjunto de acuerdos, y convenios, en la definición de los requerimientos entre el cliente y el líder de proyecto, gerente ó programador (dependiendo de la organización de la empresa) establecen, negocian o consensan un acuerdo de las rutinas que debe contener el software a desarrollar; el segundo espacio que designamos como de *interacción sígnica* tiene que ver con el proceso de *interpretación sígnica,* que hace referencia a los *fluidez cognitiva* entre administradores, programadores y tester para definir, crear y generar las rutinas de código que resuelven el problema planteado (*textos sígnicos*).

Las *interacciones sígnicas,* se llevan a cabo en un contexto de colaboración, conflictos y resistencias, consensos, arreglos y juegos, que están embebidas en una *constelación de relaciones significativas* generadas a través de un proceso de intencionalidades, percepciones del trabajo, de sentidos y conflictos dirigidos, formales e informales, expresados a través de una serie de juegos de saber y de poder entre aquellos agentes que interaccionan en el proceso de trabajo, como son el diseñador, líder de proyecto o el administrador de la configuración; o bien los trabajadores cognitivos como son los programadores, desarrolladores, tester,

documentador en un contexto de *dinámicas relacionales* que definimos como *interacción simbólicas, sígnicas y gráficas.*

8.2.1.- Interacción simbólica: contingencia latente y juegos de saber resolver

En el diseño del software, es importante que el agente que lleva a cabo las entrevistas con el cliente las realice de tal forma que la lista de requerimientos que emana de éstas entrevistas se apegue a lo que el cliente realmente necesita; una segunda condición es que estos requerimientos sean "traducidos" en el diseño del software a desarrollar; proceso que implica una sistematización de los contenidos en un conjunto de requerimientos y tareas que los programadores deberán a su vez resolver en algoritmos. Este proceso interpretativo implica una doble *interacción simbólica* entre el cliente y el desarrollador; el cliente enuncia a través de representaciones y sentidos simbólicos los requerimientos para el software que desea; el diseñador en las entrevistas re-configura los requerimientos a través de una serie de símbolos y códigos propios al saber del diseñador, enlistando y condensando con el cliente la lista de requerimientos del software. Este contexto es atravesado por una contingencia latente, que la ingeniería del software denomina *aflicción del software*, éstas contingencias significan un conjunto de incertidumbres que son intrínsecas al diseño y al proceso de *interpretación sígnica* de los requerimientos del cliente. Al respecto, las estadísticas de la ingeniería del software estiman que en la década de los noventa aproximadamente 40% de las fallas en los sistemas de software se debían a falta de información en la lista de requerimientos; en el argot del desarrollo del software es común enterarse del efecto de "*teléfono descompuesto*" entre diseñadores y desarrolladores, aludiendo a los problemas de interpretación de las necesidades del cliente, o bien por una deficiente comprensión de los requerimientos, lo que se traduce en incumplimiento de tareas, atraso en la entrega, incremento en los costos, etc.

Para Foucault (1993) en "microfísica del poder", el poder no es monolítico, sino que consiste en una serie de *juegos específicos* en el que todos los participantes actúan en diferentes niveles y con distintas intenciones e intensidades. Para Foucault no se puede pensar en términos del poder al estilo de Taylor, en el que por un lado está la administración o los gerentes con un poder disciplinario, señalando *normas* y herramientas que deben ser cumplimentadas; por el contrario, Foucault señala que el poder nunca es enteramente controlado por alguien o algunos, sino que en cada situación, contexto y escenarios se

reconfiguran nuevos recursos inesperados de juegos de poder (Foucault:1993). En este sentido, consideramos que el primer espacio de poder del proceso de trabajo en la industria del software convergen una serie de *operaciones subjetivas* entre el cliente (ó aquellos que contrataron el desarrollo del Software) y el agente que está describiendo los requisitos ó tareas que el software debe ejecutar.

La *interacción simbólica* está impregnada por una serie de contingencias latentes: ¿Las entrevistas con el cliente fueron suficientes o se requiere de mas planificación?; ¿El mecanismo que se llevo a cabo en las entrevistas fue eficiente?; ¿Se entrevistaron a los usuarios finales del software?; ¿Se evaluó el hardware y software del cliente?; ¿El cliente entrevistado posee suficiente información de la empresa donde se aplicara el sistema informático solicitado?, ¿Fueron suficientes las preguntas realizadas al cliente?[98]. Preguntas que forman parte del guión de la entrevista, información que es de forma en la *lista de requerimientos* del software a desarrollar; sin embargo, en el fondo intervienen las experiencias y habilidades comunicativas, así como las destrezas cognitivas para representar mediante iconos textuales (texto escrito) las representaciones simbólicas del cliente; la disponibilidad de información y capacidad de *interpretación sígnica* (símbolos del cliente en iconos textuales, que son los requerimientos), además de la fluidez cognitiva del entrevistado como del entrevistador; de tal forma que las incertidumbres sean mínimas cuando se consensa con el cliente la lista de requerimientos. Al respecto, el programador CM3 que labora en una empresa que se especializa en el desarrollo de software para palm-top señalaba que las incertidumbres están presentes en el ciclo de vida del proyecto, se traducen en incumplimientos de tareas o en el peor de los escenarios, que el proyecto se "caiga" es decir que se cancele "...que el proyecto no se entregue a tiempo y que a veces no cumpla con los requisitos, es lo que más constante que se da aquí en la empresa" (programador CM3, MEKI).

Las incertidumbres y contingencias posibles en los requerimientos también son afectadas por agentes intermediarios, como bien señala el programador DINS-03 quien es freelance y experto en programas de seguridad: comenta que en la industria del software existen mediadores denominados "consultoras de tecnologías de la información" que fungen como "clientes" y en realidad están subcontratando procesos de software y son ellos los que

[98] El cliente puede ser el gerente administrativo y no necesariamente conocer lo suficiente las necesidades del sistema a desarrollar; bien, puede ser un conjunto de gerentes que no tengan suficiente información de las necesidades de la empresa como el de comprar, el de mercadotecnia, etc.

se hacen pasar como "el cliente" para especificar los requerimientos, y cuando se hace entrega del programa informático, este puede no cumplir con las expectativas del cliente final, porque el subcontratista no especifico detalladamente algunos requerimientos (programador freelance DINS 03). Por su parte el líder JH0, con maestría en ingeniería de software y gerente del área de capacitación de la empresa CISE con mas de 300 programadores, señalaba que si no existe en la empresa un área de análisis de los requerimientos es factible que no se comprendan eficientemente las necesidades reales de lo que quiere el cliente o usuario; por tal motivo el diseño de los requerimientos será incorrecto y se desarrollará un software incompleto (Líder CISE JH0).

Las posibles contingencias no sólo están presentes en la licitación de requerimientos, también influyen aspectos relativos al hardware que emplee el cliente, así como el "grado" de conocimiento suficiente o no de quien esta redactando los requerimientos o conduciendo la entrevista. Al respecto el programador CM3 de la empresa MEKI, especializados en programación para agendas electrónicas del tipo PDA, señalaba que en alguna ocasión la empresa ofreció el desarrollo de un software que controlaría una cámara fotográfica a través de unos controladores API de Windows[99], implementados a través de una PDA, no se cumplió del todo con el requerimiento señalado, porque el sistema Windows instalado en la PDA no contenía los API, ya que sólo los contiene el software Windows que está instalado en la computadora de escritorio, así que la solución no fue la prometida, por falta de suficiente información por parte del desarrollador:

> "...buscar una solución alterna que era manejando de forma externa a la cámara o sea no con el API de Windows, entonces se le dijo al cliente que esa solución le podíamos dar, que a lo mejor no vamos a tener tanto control en la cámara como quisiéramos, pero podía salir el proyecto" (Programador, MEKI CM3)

El comentario del programador CM3 es importante no sólo porque explica que muchos de los sistemas informáticos que se utilizan funcionan, pero ello no quiere decir que sean eficientes "*a lo mejor no vamos a tener tanto control en la cámara como quisiéramos, pero*

[99] La Interfaz de Programación de Aplicaciones, cuyo acrónimo en inglés es API (Application Programming Interface), es un conjunto de funciones residentes en bibliotecas (generalmente dinámicas) que permiten que una aplicación corra bajo el sistema operativo Windows. Las funciones API se dividen en varias categorías: Depuración y manejo de errores, E/S de dispositivos, DLLs, Comunicación entre procesos, Manejo de la memoria, Monitoreo del desempeño, Manejo de energía, etc. http://es.wikipedia.org/wiki/Win32

salió el proyecto". Por otra parte, nos ejemplifica que estamos en presencia de un proceso de *trabajo fluido*, donde la participación del cliente es importante (en mayor o menor medida, dependiendo de la complejidad del software y de la organización de la empresa desarrolladora) así como *saber resolver el problema* por parte de los agentes, agilidad que se sujeta a una serie de configuraciones subjetivas que intervienen en el proceso de trabajo.

Otros problemas que se presentan, es que el cliente modifique las requerimientos una vez iniciado el proyecto, es decir que solicite nuevos requisitos del programa informático, que no habían sido acordados, lo cual tiene implicaciones si sólo si: a) los cambios no afectan la estructura del diseño, de lo contrario se hacen a un lado, o bien pueden significar mayores costos o una controversia con el cliente; b) si se aceptan los cambios -dependiendo de la complejidad- puede significar un costo extra, lo cual también remite a una negociación con el cliente; c) dependerá del grado y avance en el desarrollo del software, si está avanzado el proyecto posiblemente regrese a modificar partes del sistema lo que es caro o conflictivo, más allá de un costo extra, puede significar problemas "de configuración interna".

> "... ya el cliente dice «sí me gusto...pero esto cámbialo», oye pero ¿sabes qué? te va a costar porque es tiempo...en lugar de avanzar en otro proyecto pues nos vamos a tener que regresar «ok, no pues déjalo así...o sí cámbiale». Entonces normalmente se le da así como que una prueba y ya después ya todos los módulos" (programador GG3 MIXE-T1).

El programado GG3 de una empresa con mas de 100 programadores que desarrollan software a la medida, señalaba que para evitar problemas en el diseño de los requerimientos, se les explicaba al cliente mediante "demos" o "pantallazos" –son diseños gráficos sin contenido- el contenido grafico de las pantallas, que no contiene información pero se utiliza para establecer una serie de símbolos como el tipo de letra, colores, líneas, iconos, etc. Sin embargo, cuando se desarrollan algunos módulos se le presenta al cliente para que de "el visto bueno" en cuanto a contenido y lógica de la información en la interfaz gráfica. Otro ejemplo relativo a los cambios del cliente lo comenta el líder CM3, quien además de promover la industria del software mexicana en California, Texas y Arizona (EEUU), así como en Europa (España) es gerente de una empresa tipo 2, señala que tuvo una experiencia con un cliente, que en un principio llegaron a un acuerdo en los requisitos que cumpliría el programa informático a desarrollar, pero, después de seis meses de trabajar en el proyecto, llegaron otros integrantes de la empresa y solicitaron mas requerimientos, pero no estaban dispuestos a pagar mas de lo

acordado, motivo por el cual se canceló el proyecto (Líder RS0 TADI MT2). En las entrevistas que se implementan para la licitación de requerimientos, el administrador o líder del proyecto deben descubrir posibles "vacíos de información", identificar potenciales conflictos de intereses, conflictos que pueden ser tanto con los usuarios, como con el entorno de hardware y software con el cual inter-operara el sistema a desarrollar

> "En ocasiones... muchos clientes entre ellos no se ponen de acuerdo, has de cuenta que el de sistemas y el gerente de mercadotecnia piden cosas que no se ponen de acuerdo, el de sistemas dice «No, hay que hacerlo así», ya se lo lleva al de Mercadotecnia o al de Producción «Oye mira este es el software»... «Ahhh...es que mira esto no funciona así, no me gusta» (programador CM3 MEKI PT2)

Este espacio de poder está impregnado de contingencias y eventualidades que conduce a fallos, errores que forman parte de la "*aflicción del software*", como respuesta las *normas disciplinarias* constituidas por diferentes ingenierías intentan establecer un conjunto de procesos estándar que aclaren las contingencias que envuelven la conceptualización y formalización de los requerimientos del software a la medida. Como es el caso de la recientemente creada Ingeniería de Requisitos del software que es definida como "un conjunto de actividades en la cual se intenta comprender las necesidades exactas de los usuarios del sistema software, para traducir tales necesidades en instrucciones precisas y no ambiguas, las cuales podrían ser posteriormente utilizadas en el desarrollo del sistema" (Loucopoulos 1995, en Pressman, 2002)[100]. La Ingeniería de Requisitos, como disciplina, busca identificar el propósito, dirección y alcance del sistema a desarrollar, consiste en un conjunto de *dinámicas e interacciones simbólicas* que pretenden aprehender los requerimientos de un software a desarrollar y convertir los requerimientos (representaciones simbólicas del cliente) en un conjunto de signos textuales que representen una descripción completa y documentada de las necesidades del sistema. La lista de requerimientos representa un conjunto de *interacciones simbólicas* entre las necesidades reales de los clientes-usuarios y las representaciones de quien realizo las entrevistas. Sin embargo, éstas propuestas de la ingeniería de requisitos aplican en grandes fábricas de software, ya que según las empresas entrevistadas éstas propuestas son burocráticas, rígidas, difíciles y costosas de implementar.

[100] Loucopoulos, P; Karakostas, V. (1995); System Requirements Engineering McGraw-Hill, 1995, citado por Pressman 2002.

> "Si de hecho creo que las partes más finas es eso, o sea el comprender que es lo que quiere el cliente, ya sea que el líder nos lo diga o el cliente directamente....como programador nos facilita un poco la vida por que lo hacemos una vez, o sea, se hace una vez y ya quedo, pero si por ejemplo el líder nos pasa ese requerimiento con fallas, que no este bien definido, entonces lo hacemos pero ya después nos lo regresan «No, sabes que, es que así no, necesito esto y esto»...deben estar seguros, debe tener toda la información necesaria, debe saber lo que en realidad quiere el cliente, para que uno lo pueda hacer bien" (programador CM3 MEKI- PT2).

La ingeniería del software ha orientado sus esfuerzos en proponer un "proceso de desarrollo sistemático y disciplinado del sistema informático" que minimice las contingencias a través de implementar distintos tipos de ciclo de vida del proyecto y del producto que sea eficiente; sin embargo, como ya explicamos, los programadores y líderes se orientan más por las experiencias en proyectos similares practicados con anterioridad, se guían por la acumulación de saberes que en aplicar un método estándar, ya que suponen implícitamente que si el método implementado dio resultado en un proyecto, no tiene porque fallar en el siguiente. Sin embargo, está suposición no es del todo cierta, ya que cada cliente plantea una serie de problemáticas diversas.

Este espacio de poder, que corresponde a la conceptualización y formalización de los requerimientos o requisitos no sólo se corresponden con aquellas solicitudes establecidas por el cliente, sino que existen una serie de factores externos e internos que deben ser considerados al momento de proponer el conjunto de documentos que integran los requisitos del sistema[101]. Organizar los requisitos del sistema implica la existencia de una *fluidez*

[101] Requisitos de comunicaciones del sistema: Son de carácter técnico relativos a las comunicaciones que deberá soportar el sistema software a desarrollar. Por ejemplo: el sistema deberá utilizar el protocolo TCP/IP para las comunicaciones con otros sistemas. Requisitos de interfaz de usuario: Se especifica las características que deberá tener el sistema en su comunicación con el usuario. Por ejemplo: la interfaz de usuario deberá ser consistente con los estándares definidos en IBM's Common User Access. Se debe ser cuidadoso con este tipo de requisitos, ya que en esta fase de desarrollo todavía no se conocen bien las dificultades que pueden surgir a la hora de diseñar e implementar las interfaces, por esto no es conveniente entrar en detalles demasiado específicos. Requisitos de fiabilidad: deben establecer los factores que se requieren para la fiabilidad del software en tiempo de más uso del sistema. Se mide la probabilidad del sistema de producir una respuesta satisfactoria a las demandas del usuario. Por ejemplo: la tasa de fallos del sistema no podrá ser superior a 2 fallos por semana. Requisitos de entorno de desarrollo: Se especifica si el sistema debe desarrollarse con un producto específico. Por ejemplo: el sistema deberá desarrollarse con Oracle 7 como servidor y clientes Visual Basic 4. Requisitos de portabilidad: Definen qué características deberá tener el software para que sea fácil utilizarlo en otra máquina o bajo otro sistema operativo. Por ejemplo: el sistema deberá funcionar en sistemas operativos Windows 95, 98 y Windows NT 4.0, siendo además posible el acceso a través de Internet usando cualquier navegador compatible con HTML 3.0. Amador Durán T, y Beatriz Bernárdez J. Metodología para la Elicitación de Requisitos de Sistemas Software,

cognitiva entre cliente-empresa de software-cliente, generándose distintas versiones de la lista de requerimientos. Por ejemplo el líder de proyecto DC2, indicaba que si el cliente posee cierto grado de cultura informática disminuyen las incertidumbres, lo cual no quiere decir que desaparezcan:

> "...depende mucho de la cultura del cliente. Sí el cliente esta en algún nivel de cultura informática se minimizan los errores pero no se eliminan. Sí tienen menor cultura informática puedes llegar a tomar decisiones que afectan directamente la posibilidad de generar software correspondiente". (Líder DC2 CATI-PT2)

El programador y líder de proyecto MZ3 de una microempresa que se especializa en software administrativo a la medida y otros programas específicos para empresas micro, señala que se firma un contrato con el cliente, contrato que comprende no sólo el costo de desarrollo, sino también el de capacitación de los usuarios finales y, que en caso de incumplimiento en los requerimientos éstos se corrigen, siempre y cuando no sean requerimientos que no estaban inicialmente en el contrato (programador MZ3; líder TADI MT2). Para el programador GG3 de la empresa MIXE la licitación de requerimientos representa formas complejas para corroborar que los requerimientos sean los adecuados para el cliente, por ejemplo a través del diseño de prototipo rápido, o bien mediante un esquema "vacío" es decir, se presentan pantallas sin contenido, sin datos, solo a nivel de esquemas. Y, en caso de que haya un repositorio de módulos equivalentes a los que solicita el cliente, se presenta un prototipo de diseño rápido. Sin embargo, estas soluciones temporales no son lineales, sino que representa un proceso heurístico, casuístico, fluido. El programador GG3 de la empresa mediana MIXE, señala que en un proyecto que estaban desarrollando por capas (ciclo de vida), la primer capa se integraba por 214 pantallas de interfaz gráfica, la segunda capa corresponde al desarrollo de la base de datos y la tercera, el desarrollo del código que entrelaza los datos que se agregan a las pantallas con la base de datos.

> "ahorita no se ha codificado nada, pero en interfaz gráfica tu ya puedes ver el sistema, si tu quieres verlo funcionar que si le aprietas un botón te va a aparecer tal pantalla...Ahorita estamos comenzando a trabajar una parte de base de datos... hacer el diseño, evaluación de todas las tablas que tenemos de Excel, ver qué campos si se meten, cuales no (reutilizar código) qué campos te van a servir como consulta..." (Programador GG3 MIXE)

Versión 2.1, Departamento de Lenguajes y Sistemas Informáticos, Facultad de Informática y Estadística, Sevilla, España, Octubre de 2000.

En la etapa del diseño, Davis (2005); Pressman (2002) y otros expertos en Ingeniería del Software, proponen una serie de modelos, sin embargo, la mayoría de nuestros entrevistados coincide en que el diseño debe:

- Considerar diferentes enfoques alternativos acorde a la lista de requerimientos.
- No inventar nada que ya este inventado. Hace referencia a la reutilización de código, subrutinas, incluso módulos, librerías, etc. que ya se hayan desarrollado previamente por la empresa.
- Que la modularidad del sistema presente uniformidad, integración, fiabilidad.
- Admitir cambios, que no se trate de un diseño cerrado.
- Querer solucionar los requisitos con código, no es respetar el diseño.
- Valoración continúa del diseño, en el proceso de desarrollo del Software.
- Revisar continuamente el diseño, para evitar errores de concepto.
- Reducir en lo posible lo abstracto del diseño.

Cuando el diseño es presentado por un agente o por un programador que este al frente del área de diseño, es cuado podemos decir que da inicio el segundo espacio de saber, punto en el cual la lista de requerimientos o módulos del sistema, deben ser traducidos a símbolos, iconos, algoritmos que, dan por resultado una tarea o acción que demanda un módulo, un conjunto de requisitos. Proceso que no es individual, está inmerso en una serie de *constelación de configuraciones subjetivas* individuales o colectivas; virtuales o cara a cara; cooperativas o de resistencia; donde se ponen en juego conocimiento formal e informal.

8.2.2.- Interacción sígnica: normas disciplinares y herramientas procedimentales

En el proceso de trabajo del software para que un programador resuelva un problema mediante algoritmos, convergen toda una serie de cualidades: investigar, comprender y reflexionar, así como juzgar y decidir aquello que se considere eficiente; procedimiento que implica en un solo acto el saber-hacer como un solo procedimiento. Foucault (1976) señala que el poder no es omnipresente, por el contrario se produce y reproduce a cada instante, en todos los puntos de la red de interacciones; el poder está en todas partes, no porque lo englobe todo, sino porque proviene de todas partes. Para Foucault el poder circula, transita entre los individuos, no se aplica a ellos (1976:113 ss.). En el desarrollo del software las decisiones individuales fluyen en el proceso, decisiones que significan y representan saber y poder de decisión. En este sentido, consideramos que en el espacio del procesamiento de datos, en el proceso de "traducción" de la información disponible en la lista de requerimientos, es decir el proceso de conversión en *signos (lenguajes de programación) a través de símbolos* (lista de

requerimientos) en *textos signicos* (rutinas de algoritmos) que resuelven los requerimientos señalados por el cliente, es lo que hemos denominado *interacción sígnica*. El *texto signico* es conjunto de rutinas y subrutinas de algoritmos lógicos, que a su vez son representación simbólica de sentidos, consideraciones y pretensiones dentro del trabajo y frente al trabajo por parte de los actores laborales, significan una serie de prácticas con *representaciones y sentidos del aprendizaje* que confluyen a través de una *constelación de interacciones significativas* entre los actores que intervienen en el proceso de trabajo; *constelaciones* que son divergentes, con cooperación y resistencias, boicot y consenso, arreglos y conflictos, es decir, entre los diferentes espacios del proceso de trabajo coexisten diversas *configuraciones subjetivas*.

Al estilo de Foucault, dichas configuraciones concurren en el proceso de trabajo del software a la medida, se suceden en coyunturas específicas con relaciones de poder intensas y tensas cuando un agente (administrador del sistema, arquitecto de software, líder de proyecto, programador experto, etc. dependerá del grado de sofisticación en la organización de la empresa) interacciona con el programador con un cierto fin: que se comprenda y aprehenda el *problema a resolver*, problema que es planteado a partir de la lista de requerimientos, *interacción sígnica* que representa una serie de relaciones de poder tensas. Al respecto el líder RF1, gerente y dueño de la empresa DYYA, señala que para él los programadores en ocasiones son muy "delicados", difíciles y, que quizá tengan derecho a ser así, porque al final de cuentas son "artistas", en el sentido que los programadores crean y fabrican el código (Líder DYYA RF1), pero más adelante, en la entrevista acomete contra los programadores, a quienes señala que resuelven los problemas sin haber internalizado el mismo (falta de identidad creativa) "...el programador a base de pegarle a las teclas, ... hasta que funcione la mugre...sin realmente haber profundizado en el problema..." (Líder DYYA RF1).

El programador de software JO3 comenta que le gusta *resolver problemas* por su "cuenta propia y a su manera", no le gusta interaccionar con otros programadores, trabajar en equipo o tener que documentar todas sus actividades "...¿para que documento?...que todo el mundo las conozca... como se dice...propiamente programo, pero sin alguna bitácora o algo por el estilo" (Programador MIXE JO3). El programador señala que documenta lo menos necesario, aludiendo al argot de la informática que señala una ley no escrita "si se entiende, para que lo documento". *Práctica social de ocultamiento del saber hacer*, es decir, mediante la dinámica del *juego* de documentar lo mínimo, de *ocultar* el proceso del *como* se resolvió el

problema. *Juegos* que representan intenciones y representaciones tensas que se constituyen en el proceso de trabajo como relaciones de saber-hacer y saber como poder, que están embebidas en la programación del software. En términos Foucaultianos, en la medida que se *saben* resuelven los problemas planteados por el cliente se disipa *el poder* del programador, sin embargo, se erigen nuevas relaciones de saber y de poder frente a nuevas tareas; o entre quienes participan en el desarrollo del proyecto o entre aquellos que utilizan el software; el cliente como usuario final también esta incluido en los juegos del saber como poder.

Por ejemplo, en el tercer espacio de poder (*interacción gráfica*) señalado en el esquema 9, en la implementación del programa informático en las instalaciones del cliente o usuario final, este rompe con rigideces contextuales donde se implementa el software desarrollado, se enfrenta a un contexto de resistencias y conflictos por los usuarios finales, ya que el software implementado en las instalaciones del usuario final rompe con otros *espacios de poder* y con rutinas embebidas entre la cultura laboral de los usuarios finales (Baethge y Oberbeck: 1986 [1996]; Perrini: 1988 en Castillo:1993). En otras palabras, según Foucault el poder del diseñador se relativizará frente al conjunto de programadores que desarrollan los algoritmos, una vez que los programadores *saben resolver* los problemas planteados en el diseño, relativización que no necesariamente acontece en el proceso de trabajo, no es que se esfume el saber como poder, que se disipe el saber-poder como señala Foucault; sino por el contrario consideramos que es un conjunto de conocimientos que se acumulan como *saber hacer* y esta acumulación de experiencias se traduce como poder, es decir la *acumulación de saberes,* como conjunto de conocimientos y aprendizajes (experticia), forma parte de las intencionalidades y representaciones subjetivas de los agentes que participan en el proceso de aprendizaje. No es que haya una "suma cero" de saber y poder; por el contrario consideramos que al interior de las *constelación de relaciones subjetivas* que intervienen en el proceso de trabajo se acumulan, reproducen y reconfigura el saber hacer como poder.

Los espacios de poder definidos en el esquema 9 no son rígidos ó sistemas obligatorios del proceso de trabajo, por el contrario se constituyen sólo como un referente abstracto de la *fluidez cognitiva* que cosifica los espacios de saber-poder intrínsecos en el desarrollo de software. Hummer y Champí (1994) señalan que algunas empresas practican enfoques divergentes de este método, realizan procesos integrales centrados en tareas, personas o rutinas, donde no existe esta separación de tareas; por ejemplo los métodos ágiles, la

programación extrema, programación en software libre, entre otros métodos donde el proceso de trabajo es interactivo, mas que una modularización, se resuelven los problemas planteados en una espiral de tarea-problema-resolución en el momento mismo que se identifica un requerimiento, están yuxtapuestos los espacios de poder ,una vez resuelto cada requerimiento se implementa en "casa" del cliente para así probar y comprobar los progresos de cada requisito del proyecto.

En el procesamiento de datos del software las relaciones de poder que se establecen entre programadores y diseñadores está embebido en una *constelación de relaciones subjetivas* que representan una serie de acciones individuales y colectivas, que son presionada y constreñida por estructuras; donde las estructuras son representadas por un conjunto de *herramientas procedimentales* como los métodos, métricas de calidad, ciclos de vida, etc. que presionan el *como* deben licitarse los requerimientos del cliente y, por otro lado influyen en la acción de programadores para *saber resolver* los problemas planteados en los requerimientos. Al respecto el programador CM3 de la empresa MEKI, señala que el líder o el programador experto es el que especifica *como* deben funcionar los módulos, cuales son los requerimientos que contiene el módulo, pero cuando dichos requerimientos se interpretan en *textos sígnicos* es el programador responsable quien *resuelve* acorde a la acumulaciones de habilidades y destrezas (experticia), más la posiciones de éste en los flujos de aprendizaje y las interacciones concretas y virtuales de las que forma. *Saber-resolver* no necesariamente implica el cumplimiento de los requerimientos del cliente, sino que coexisten una serie de contingencias que limitan dichos cumplimientos; por ejemplo que los requerimientos elaborados por el cliente no sean suficientes, que la interpretación de los símbolos del cliente en *textos sígnicos* (algoritmos) no sea eficiente, que el programador no interpretó correctamente el requerimiento o bien se resiste, boicotea o "resuelve a medias" el requerimiento; también puede modificar intencionalmente el diseño(Programador MEKI CM3).

La solución de problemas por distintos procedimientos es normal dentro del proceso de trabajo, ya que como señala el líder de proyecto RS0 gerente y dueño de TADI con estudios de maestría en el CINVESTAV, que si bien es cierto el líder especifica los requerimientos de los módulos, ello no significa que el líder tenga un conocimiento claro de *cómo y porque* se codificarán esas funcionalidades "...si alguien nos dice: «oye quiero que programes así y así» es muy probable que tengamos problemas en un ochenta ó noventa por ciento" (Líder RSO

TADI). Es decir, el responsable en *saber resolver* el problema planteado es el programador no el líder. En la medida que el líder de proyecto explica e interacciona con los requerimientos que contendrá el programa informático le permite diseñar las funcionalidades, pero la interpretación sígnica, es decir la conversión del diseño en algoritmos, es responsabilidad del programador. Sin embargo, el diseñador o programador responsable, si participa en las *dinámicas subjetivas* que pueden traducirse como *aprendizajes significativos* entre el líder del proyecto y los programadores, donde la exposición de requerimientos por parte del líder de proyecto no significa una imposición a los programadores del *como* se resolverá el problema planteado, pero si existen *herramientas procedimentales y normas disciplinares* que constriñen y presionan el *saber resolver* en el interior del proceso de trabajo, ello no significa que efectivamente el programador documente el *que hacer* y *como hacer,* del proceso de trabajo; el programador, quien a partir de la *interpretación sígnica* resolverá o propondrá una solución de los problemas a resolver, es quien crea una serie de algoritmos acorde a auna serie de proceso, como la acumulación de aprendizajes tácitos y formales, así como las *dinámicas subjetivas* de compromiso, emociones y disposiciones para resolver el problema planteado.

La coexistencia de *herramientas procedimentales y normas disciplinares* caracterizan la diferencia tácita entre la programación del tipo artesanal con respecto de la programación moderna, ésta última está contextualizada por un conjunto de *herramientas procedimentales* como son lenguajes de programación, metodologías de desarrollo orientada a objetos, técnicas UML, RUB, etc. que permiten diseñar un plan o modelo para el análisis, desarrollo. documentación e implementación de programas de software. Las *herramientas procedimentales* constituyen una serie de procesos y métodos que permiten a los agentes que intervienen en el proceso de trabajo planificar un sistema de software sin que necesariamente tenga que tener conocimientos de programación (división del saber hacer). Al respecto el líder y experto en programación JA2, de la empresa exportadora de software con 100 programadores en el DF, pero a nivel nacional con mas de 600 programadores, señalaba que en la programación artesanal el desarrollo de software se constituía como "*caga negra*" donde sólo el programador conocía el contenido y la forma de operar del código implementado, si otro programador accedía a leer el código era casi imposible de comprenderlo, mas aún de modificarlo, sin embargo con *herramientas procedimentales* como UML, RUB, métodos orientados a objetos, etc. la complejidad disminuye:

"Antes lo hacíamos muy tradicional y por eso para poder entrar y comprender las líneas de código era muy problemático... ahora se ha hecho más fácil con la parte de lo que es UML, con esa metodología se ha disminuido la complejidad..." (Gerente, MIXE JA2).

En el mejor de los casos los líderes o administradores de proyecto implementan *herramientas procedimentales* como Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) que es el lenguaje de sistemas de software (aún no es un estándar oficial) que utiliza un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML que ofrece un incipiente estándar para describir una "planificación" del sistema (modelado del proyecto), incluyendo aspectos conceptuales como procesos de negocios y funciones del sistema, lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. Otro método para llevar a cabo "planificaciones" en la administración de proyectos, son las certificaciones PMI (Project Management Institute) que entre las más frecuentes están: Asociado en Gestión de Proyectos Certificado (CAPM –por sus siglas en ingles-) y Profesional en Gestión de Proyectos (PMP –por sus siglas en ingles-)[102]. Otra herramienta que pretende establecer estándares es el Proceso Personal de Software (PSP –por sus siglas en ingles) que es una pequeña versión de CMM, propuesto en 1995 por Humphrey[103] (Humphrey:1995; Sommervilley:1994; Pressman:2003; etc.). El conjunto de *herramientas procedimentales* como PMI, UML, PSP, junto con el Proceso Unificado Racional (Rational Unified Process RUP)[104], es un proceso de desarrollo de software que junto

---

[102] Project Management Institute (PMI) ofrece dos certificaciones: A) Un Asociado en Gestión de Proyectos Certificado (CAPM®): es aquel que ha demostrado una base común de conocimientos y términos en el campo de la gestión de proyectos. Se requieren 1,500 horas de trabajo en un equipo de proyecto o 23 horas de educación formal en gestión de proyectos para conseguir esta certificación. B) Un Profesional en Gestión de Proyectos(PMP®) ha experimentado una educación específica y requerimientos de experiencia, ha aceptado ceñirse a un código de conducta profesional y ha pasado un examen designado para determinar y medir objetivamente su conocimiento en gestión de proyectos. En el 2006, el PMI reportó más de 220,000 miembros y cerca de 200,000 PMPs en 175 países. Más de 40,000 certificaciones PMP expiran anualmente, ya que un PMP debe documentar experiencia en proyectos en curso y educación cada tres años. http://es.wikipedia.org/wiki/Project_Management_Institute

[103] El Proceso Personal de Software (Humphrey:1997, An introduction to the Personal Software Process) se caracteriza porque es de uso personal y se aplica a programas pequeños de menos de 10.000 líneas de código. Se centra en la administración del tiempo y en la administración de la calidad a través de la eliminación temprana de defectos. En el PSP se excluyen los siguientes temas: Trabajo en equipo, Administración de configuraciones y Administración de requerimientos. http://es.wikipedia.org/wiki/Personal_Software_Process

[104] El Rational Unified Process, RUP es un proceso de desarrollo de software, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP, a diferencia de UML, no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización http://es.wikipedia.org/wiki/RUP

con el Lenguaje Unificado de Modelado (UML) y en menor medida las certificaciones PMI (CAPM y PMP) constituyen un conjunto *buenas prácticas normativas* que representa un conjunto de procesos estándar en el desarrollo de software. Las *buenas prácticas* son aquellos procesos que se apegan a estándares e implementación de metodologías y métricas de calidad que representan un "proceso robusto".

> "... ahorita ya cuentas con varias herramientas para eliminar esa parte que muchas veces tenias que volver a leer, debías tener alto conocimiento técnico; ahora con UML o algunas herramientas (RUP) ya no necesitas ser desarrollador para saber que es lo que va a hacer el programa, no tienes que decirle al desarrollador que haga... no conoces nada de códigos, sólo «ah mira pues aquí entra la orden de compra, hace esto y ya» (Gerente, MIXE JA2).

Las *herramientas procedimentales* permiten que el diseñador o líder de proyecto no sea experto en programación; sin embargo, esto no significa que las empresas adopten los estándares citados, ya que reconocen que hacerlo es limitar la propia versatilidad de la industria del software. El programador JH0, comenta que si bien es cierto que existen heterogéneas *herramientas procedimentales* para la administración de proyectos, ello no significa que se pueda adoptar una sola herramienta especifica para el desarrollo de un sistema informático, ya que se corre el riesgo de forzar un proceso de trabajo que no le corresponde[105].

> "...porque cada proyecto se ataca de diferente manera, inclusive depende del ciclo de vida, si escojo un ciclo de vida en cascada, a la mejor ese es secuencial en la utilización de las disciplinas; pero si yo escojo un ciclo de vida iterativo - incremental, es otra manera de como se buscan las disciplinas, no puedo... tengo que ser flexible, no me puedo casar con una herramienta" (programador CISE JH0)

Consideramos que el proceso para desarrollar código se corresponde con un período de *construcción social de la ocupación.* Esta industria está en proceso de reconocimiento social de la ocupación desempeñada por los programadores, integrada por micro y pequeñas empresas, quienes no implementan complejas metodologías caras y complicadas, ya que consideran que no tienen mucho impacto en el control de la contingencia, además de que estas empresas se preocupan más en sobrevivir que en aplicar *herramientas procedimentales* o

[105] Precisamos que algunas investigaciones realizadas en empresas que operan como fabricas de software que contratan a miles de programadores, donde implementan éstas herramientas-inmateriales la división del trabajo es tácita, en el sentido que dichas fabricas operan como maquiladoras de software, en las cuales solo desarrollan parte del código, sin que ello signifique que el programador conoce todo el proceso del software mismo (Dayasindu y otros:2002; Castillo:2006)

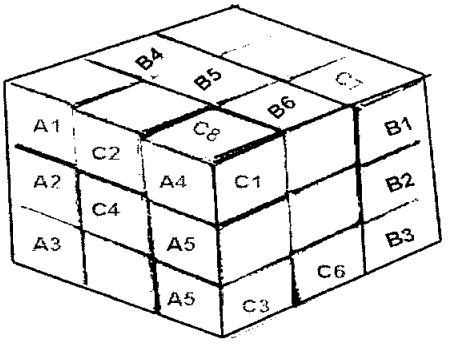
certificaciones costosas. Entre los programadores y los líderes se señala más un concepto de "proceso ágil" en el sentido de resolver, solucionar, desarrollar el software de acuerdo con los procesos, esquemas o tipologías que se llevan a cabo en la empresa a iniciativa de los propios agentes que participan en el desarrollo del software en lugar de implementar costosas herramientas y metodologías.

8.2.3.- Fluidez cognitiva: Espacio de aprendizaje y conflicto en el texto sígnico

Supongamos que el líder de proyecto Juan presenta el proyecto Mouse, el cual se conforma de los módulos A, B, y C (ver esquema 10), los cuales a su vez se subdividen en un conjunto de tareas: A1, A2, A3; B1, B2, B3; C1, C2... C6. Se asigna al equipo de programadores de Juan los módulos A y B, y al equipo de María el módulo C. Supóngase que Juan como líder explica los requerimientos que solicito el cliente a los programadores, en esta exposición se dialoga, consensa y colabora en la asignación de tiempos y tareas a resolver en los módulos, así como la negociación de quien participa en los módulos que integran el proyecto. Sin embargo esta asignación dista de ser bajo un espíritu de colaboración o cooperación, por el contrario el desarrollo de las rutinas de cada uno de los módulos no se lleva en forma aislada o libre de conflicto, coexisten interacción entre los programadores, que se traduce en relaciones de poder.

Esquema 10

Fluidez cognitiva en el proceso de trabajo del software



Fuente: Elaboración propia en base a las entrevistas

El *saber resolver* a tiempo o arreglárselas para resolver las tareas asignadas que representan los módulos, implica una serie de *dinámicas subjetivas* hacia dentro del proceso de trabajo (cara a cara) como hacia fuera, pero desde el lugar de trabajo (pantalla a pantalla) como fuera del proceso de trabajo (tiempos de vida personal). Es decir, el proceso de trabajo

supone una yuxtaposición de los tiempos de producción y tiempos de vida del programador que abordamos a continuación.

En el apartado anterior dimos cuenta de la perspectiva espacial relativa a la licitación de requerimientos de las funcionalidades del software como un espacio de *relaciones de saber y de poder* donde no existe un solo camino para resolver un problema, por el momento citamos dos ejemplo de lo que implica el espacio de *relaciones de saber y de poder* entendido como un proceso de *fluidez cognitiva* entre los espacios del trabajo cognitivo. El primero hace referencia a la interacción simbólica, que en el caso del esquema 10, es la asignación de tareas entre ambos equipos de trabajo, por ejemplo el equipo María debe *saber resolver* los módulos correspondientes en un tiempo *T1* y el equipo de Juan en un tiempo *T2*, el software a desarrollar se integra en un proceso tenso de *resolver a tiempo* los módulos, donde cada módulo esta interrelacionado (interoperabilidad). En la programación artesanal se asignaba módulos a los programadores y se dejaba que resolvieran el problema, sin darle importancia a la documentación y comentarios paralelos del código (documentación del saber-hacer), lo cual implicaba que cuando había que proveerle mantenimiento o actualización al programa sobrevenían una serie de problemas: que la empresa que desarrollo el programa ya no existía (mortandad de microempresas de software); que el programador o programadores claves que habían desarrollado el programa ya no laboraban (rotación laboral): que no se comprendía el desarrollo del código (código espagueti); etc. Sin embargo la etapa moderna supone la implementación de *herramientas preocedimentales y normas disciplinarias* que proveen condiciones óptimas en el *saber hacer* y *resolver a tiempo* bajo buenas prácticas. Sin embargo, aún continúa la *"aflicción del software"*.

El gerente y líder JA2 de la empresa MIXE, comentaba que en la etapa artesanal del desarrollo del software, comprender la lógica de las líneas de código era muy problemático, lo cual se aligero un poco con la aplicación de *herramientas procedimentales* como UML (Líder JA2 MIXE); al respecto, lo secunda el gerente y líder DC2, de la empresa CATI, para quien la forma de disciplinar a los programadores en la documentación del código es UML ó RUP, por medio de la cual, tanto el usuario final (el cliente) como el programador que vaya a dar mantenimiento o actualización del sistema comprenda el *que y como se hizo* las rutinas:

> "...eso ya cambio muchísimo con UML, RUP y pues es parte de las mejores prácticas del como hacer las cosas, también es muy importante las memorias técnicas de las cosas, no solamente la parte de la programación sino donde va a

funcionar eso y con que ambientes y con que circunstancias y con que condiciones" (Líder CATI DC2).

Entre la programación moderna y la artesanal consideramos que es *el saber-hacer* el aspecto central en discusión, entendiendo el *saber-hacer* como *saber-resolver* las tareas asignadas a través de rutinas algorítmicas que a su vez están embebidas en una *constelación de configuraciones subjetivas* que intervienen en el proceso de trabajo, como son los *juegos de saber y de poder*, consensos y negociaciones en el resolver, entre otros arreglos sociales que mas adelante abordaremos. Sin embargo, es importante precisar algunas limitaciones en la representación del poder como lo concibe Foucault, para quien el saber no es equivalencia de poder (Sexología Tomo 1:124), sino que el poder es indisociable del saber; planteamientos que explican parte de lo que hemos abordado entre un estadio artesanal y el moderno, específicamente en aquellos espacios mediados por la *fluidez cognitiva* donde las formas, contenidos y transformaciones del saber es un proceso fluido, en el cual la circulación y transformación del *saber-hacer* transcurre como arquetipo de poder, en el entendido que *saber-resolver* un problema mediante *textos sígnicos* implica un determinado proceso cognoscitivo complejo; complejidades que se construyen y confrontan entre una serie de *constelaciones de configuraciones subjetivas* que intervienen en el proceso de trabajo.

Estructuras diversas ciñen la acción que llevan a cabo los individuos que participan en el proceso de trabajo, acciones que implican una re-configuración en las relaciones de *saber-hacer y de poder* coxistentes en el proceso de trabajo; lo cual no quiere decir que dichas relaciones de *saber* sean las más eficientes u óptimas, empero mientras el algoritmo resuelva los problemas planteados o bien que se cumplan las tareas asignadas al programador el proceso de trabajo continúa, es en este acto en el que los individuos-programadores establecen relaciones de poder en el proceso de trabajo, ya que éste se reconfigura continuamente, no es estático. Al respecto los líderes y gerentes de empresa expresaron que lo primero que observan en un programador que contratan, son las habilidades, básicamente conocimiento no formal "...de cómo resuelven o han resuelto los problemas, la practicidad sobre todo de cómo resuelven" (Líder JM2, DYYA-PT1); después que demuestren a través de su trayectoria profesional como han resuelto otros problemas y, por último las credenciales formales de estudios que poseen. En otras palabras, predomina el *saber-resolver* sobre credenciales formales.

Las relaciones de *poder* entre los programadores es temporal, así como lo fue para el líder de proyecto, como bien señala Foucault, el poder no es fijo, sólo transita entre los individuos, porque ante un nuevo proyecto de software el programador recurre nuevamente a una serie de procesos cognitivos, habilidades, destrezas, etc. para nuevamente *saber-resolver* el problema planteado, lo cual implica nuevamente una *interacción entre saberes* y *relaciones de poder* entre los que participan en el proceso de trabajo. Por tanto el *saber-hacer* en el proceso de trabajo se presenta frente al otro como poder, lo cual no es causal, ya que los programadores reconfiguran su "posición" de *saber* en cada proyecto e incluso por tarea resuelta, pero es frente a los otros que se reposicionan en términos de poder. Por ejemplo para que un programador senior se le reconozca como tal, no es la antigüedad en el puesto de trabajo, es la experticia, la *acumulación de saberes* el que otorga estatus, no son las credenciales formales de estudio, es la habilidad para solucionar problemas, es la destreza con la cual propone soluciones, es la abstracción cognitiva del procedimiento planteado. Al respecto de esta abstracción y la no existencia de un solo camino para resolver, señalaba el líder de proyecto RS2 gerente y dueño de una microempresa, que cuando estaba estudiando la maestría en el Tecnológico de Monterrey el profesor de programación y director de operaciones, le solicito que desarrollará en algoritmos la secuencia de Fibonacci, lo resolvió en cinco líneas de código, tiempo después le pregunto al profesor porque le había puesto ese problema, contestándole éste, que se podía resolver en dos líneas de código[106] (líder y programador RS2, TADI-MT2). Ejemplo que nos permite ilustrar que el *saber-resolver* no obedece a un sólo contexto cognitivo finito, sino que se constituye frente a otro como poder por la consistencia del algoritmo planteado, poder que reside en la "*lógica de programador*" para desarrollar algoritmos eficientes.

Otro ejemplo de poseer "*lógica de programador*" lo ofrecen dos programadores entrevistados en Hermosillo, Sonora. El programador FB0, señala que un conferencista de una materia que se llamaba estructura de datos, redujo una algoritmo de 500 líneas de código a 5 líneas, el algoritmo era ininteligible, pero si funcionaba, a lo cual se ufanaba el conferencista "lo que me gusta es que los hijos de %&%$· no me entienden" (Programador FB0-Hermosillo). Otro programador JN0 señalaba que en la clase de inteligencia artificial,

[106] En matemáticas, la sucesión de Fibonacci es la sucesión infinita de números naturales: 0, 1, 1, 2, 3, 5, 8, 13, etc. donde el primer elemento es 0, el segundo es 1 y cada elemento restante es la suma de los dos anteriores. A cada elemento de esta sucesión se le llama número de Fibonacci http://es.wikipedia.org

desarrollaban algoritmos genéticos de 200 líneas de código y un compañero de clases lo resolvió en 10 líneas de código. Se utilizan éstos algoritmos genéticos para resolver problemas de combinaciones aleatorias[107]:

> "por ejemplo el problema de las 8 reinas, que tienes en el tablero del ajedrez, son millones de combinaciones para que 8 reinas no se ataquen, entonces haces un algoritmo genético para resolver ese problema en milésimas de segundos, para que saque todas las combinaciones posibles, nosotros la sacábamos en 7, 8 segundos y él las sacaba en menos de 1 segundo, entonces...te quedas oye, aja, y por mas coco" (programador JN0, Hermosillo).

Por el momento no ampliaremos el significado de esta "*lógica de programador*" (le dedicaremos en el siguiente apartado un amplio espacio) pero lo utilizaremos para señalar que hay varios caminos para resolver un problema, proceso que no es individual del todo, aunque el acto de desarrollar el algoritmo es individual, la solución del problema atañe a una serie de relaciones de poder basado en el saber-hacer y saber-resolver, ya que es este saber-poder el que está presente en el proceso de trabajo, ya sea en formas de juegos de poder, juegos informales como consensos y arreglos sociales que constituyen el sistema de relaciones de poder que forman parte de la *fluidez cognitiva* del proceso de trabajo.

8.3.- Juegos y arreglos del *saber como poder* en el trabajo cognitivo

En el apartado anterior dimos cuenta del proceso denominado *fluidez cognitiva* que hace referencia a las *constelaciones de configuraciones subjetivas* y las *dinámicas de interacciones* que intervienen en el proceso de trabajo y se objetivan como una serie de *juegos de poder* como son el *arreglárselas, saber resolver* y *arreglo social* para resolver a tiempo, entre otros tipos de consensos y negociaciones que mas adelante abordaremos; por lo pronto señalamos que las *interacciones de saber y poder* no es que se desvanezcan o disipen posteriores a la interacción, como anuncia Foucault; por el contrario suponemos que si bien es cierto acontece una serie de *juegos y relaciones de poder* entre los programadores no resulta una "suma cero"

[107] En los años 1970, John Holland, sugirió una de las líneas más prometedoras de la inteligencia artificial, la de los algoritmos genéticos. Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados. http://es.wikipedia.org/wiki acceso 15 Noviembre de 2007.

para los "jugadores" que participan en el proceso de trabajo; consideramos que hay límites en la propuesta de Foucault, en términos del *saber que se traduce en poder*; "traducción" que desde la perspectiva de Foucault sólo existe en la interacción entre individuos y posteriormente a esta interacción el resultado es "suma cero", es decir que *las relaciones de poder*, una vez resuelta la situación que origino la interacción entre *saber como poder*, se disipan las relaciones de poder. Esto no ocurre en el proceso de trabajo del software a la medida, por el contrario suponemos que en esta interacción acontece una re-valoración del saber-hacer de los individuos en el proceso de trabajo. Entendiendo por revaloración del saber-hacer a todo un proceso de dinámicas subjetivas del saber-hacer que se acumulan en términos de aprendizajes y conocimientos.

Suponemos que el *saber-hacer* se traduce en poder, proceso ya señalado en el apartado anterior, retomamos algunos aspectos de la teoría del poder de Foucault para explicar en términos de un *saber-hacer* que se construye o conforma en el contexto de la *fluidez cognitiva* que no se limita al proceso de trabajo sino que trasciende las paredes de la empresa, empero consideramos que la propuesta foucaultiana posee limites en cuanto a que considera al poder como momentáneo al evento de la interacción entre individuos, como si el individuo no tuviese intereses que le motivan a dicha interacción; por el contrario consideramos que el individuo posee motivaciones, intenciones, emociones que le llevan a continuar o no con las interacciones. Al respecto Burawoy ([1979], 1989) señala que en las relaciones de producción capitalistas intervienen una serie de acciones individuales, gerenciales, patronales que pueden monopolizar el poder, que mantienen el poder no necesariamente centrado en las paredes o circunscrito a las *dinámicas subjetivas* de los individuos de la empresa como señalaba Foucault; por el contrario es un poder que se recrea, valoriza y reproduce en el proceso mismo del trabajo a partir no sólo de las interacciones[108]. Ambos planos de la subjetividad las agregamos en las *dinámicas subjetivas* y las distintas *operaciones subjetivas* dan origen a las *constelaciones de configuraciones subjetivas* que intervienen en el proceso de trabajo como fuera de él.

[108] Para el proceso de trabajo del software podemos llamarle: juegos por resolver a tiempo un problema, entendiendo por ello, que en el proceso de trabajo del software en mayor o en menor medida, los programadores acorde a una serie de acciones subjetivas y reglas formales e informales resuelven los problemas cognitivos planteados por el diseñador del sistema informático.

El planteamiento del *saber como poder* no es que tienda a cero o que desaparezca, no es que haya un "borrón y cuenta nueva" posteriores a las interacciones sociales, como ambiguamente plantea Foucault, por el contrario consideramos que en las relaciones de poder, convergen una serie de *dinámicas subjetivas* como son los intereses, intenciones, motivaciones, representaciones entre los individuos que intentarán o no monopolizar el poder. Burawoy señala que el monopolio del poder, refleja determinados conocimientos tácitos y explícitos como habilidades, destrezas, experiencias, emociones, sentimientos, formalidades en el proceso, etc. (Burawoy, [1979], 1989:102ss). Conocimientos que refuerzan el *saber-poder* de quien monopoliza (el taylorismo es el mejor ejemplo). Si hay quien monopoliza, está claro que existe aquel que se le arrebata el *poder*, es decir, se rompe con el planteamiento de Foucault, quien señalaba que el *poder* no se arrebata, no se quita, sino que se generaba y culminaba en las interacciones de los individuos. La noción de monopolio del poder señalado por Burawoy, para el caso de las *dinámicas subjetivas* entre programadores de software es importante porque nos permitirá conocer cuales son y como se configuran *los juegos por el poder* y el *desplazamiento del conflicto* por el saber-hacer, entre quien intenta monopolizarlo y los que detentan el *saber-hacer* en la programación. Sin embargo, también es cierto que en este proceso de monopolización concurren contradicciones y luchas, enfrentamientos y resistencias entre empleador y empleado, entre quien diseña el programa y quien lo desarrolla, entre quienes desarrollan el programa y donde se implementa el mismo, así como entre quienes utilizan el programa. Es decir, el conflicto y la resistencia es un componente más en el proceso de trabajo, el cual no se circunscribe sólo al proceso cognitivo, también se expresa en actitudes, conductas e indisciplinas del programador frente al líder ó gerente; Eduwards, señalaba que el conflicto cobra diversas connotaciones, que las abordaremos en el capitulo siguiente.

En el proceso de trabajo cognitivo concurren paralelamente *los juegos de resolver a tiempo* y las *prácticas subjetivas* que le atañen, suponemos que se embeben en una serie de acciones y dinámicas individuales como las de resistencias, conflictos y confrontaciones, unas soterradas y otras ocultas, que se objetivizan en acciones como *ocultar código, no documentar el código, documentación insuficiente del código*; entre otras prácticas de índole subjetivo, como el *sabotaje*, que se traduce en *renunciar al problema, no resolver a tiempo, no cumplir con tiempos*, etc. Suponemos que éstas prácticas se reflejan como pugnas, conflictos y

resistencias que se suceden no sólo por las luchas por el monopolio del poder, sino que se configuran como una respuesta o contraste soterrada al conjunto de *herramientas procedimentales* del tipo UML, RUP, PSP, PMI, CMMI, ISO/IEC 15504, IEEE, MOPROSOFT, etc. que promueve la Ingeniería del Software; también se erige ante las *normas disciplinares* representadas por un conjunto de métodos y normatividades promovidas por las diferentes profesiones del software como es la Ingeniería del Software, Ingeniería de Requisitos, Arquitectura del software, entre otras; resistencias y conflictos que se materializan como respuesta frente a las estructuras que promueven e intentan separa el *saber hacer* como poder en la programación de software.

La implementación y asentamiento de las *herramientas procedimentales y normas disciplinares* como la "buena nueva" para la industria del software, se generan tensiones y conflictos; al instituirse como una serie de estructuras que ciñen y cohesionan el proceso de trabajo. Sin embargo, quizá el mejor ejemplo de esta pugna por el poder, pero entendido a nivel de ocupación enfrentada por el control del saber-hacer son las *comunidades de aprendizaje virtual* en Internet, tanto las de software libre, como los espacios virtuales de comunicación como son los blogs, wikis, y otros espacios interactivos donde los programadores se congregan para compartir el conocimiento y establecer relaciones de aprendizaje, con amplios repositorios de información, espacios de comunicaciones del *saber hacer*, etc. espacios virtuales que ofrecen información, asesorías y soporte a programadores (relaciones de trabajo pantalla a pantalla en tiempo real o virtual) procesos que se erigen frente a la coerción estructural de las *herramientas procedimentales*, y *normas disciplinares*.

8.3.1.- El sentido subjetivo del trabajo como aprendizaje

A diferencia del proceso de trabajo en la manufactura, en la industria del software no hay un esfuerzo corporal exhaustivo, repetitivo del todo, o monótono con tiempos y movimientos detallados en manuales implementados por la gerencia. Sin embargo al igual que en la manufactura hay un desgaste si bien no físico si de índole cognitivo, lo cual no quiere decir que en la manufactura no haya desgaste mental, basta mencionar el stress productivo, la monotonía, etc. En el software, la fatiga cognitiva coexiste *un sentido subjetivo del trabajo como aprendizaje.* Una primera consideración para este *sentido subjetivo* es que para el programador el proceso de trabajo representa un aprendizaje, una ampliación del *saber hacer*

ó fortalecimiento del aprendizaje acumulado; al respecto el programador GG3, de la empresa MIXE, señalaba que presento exámenes para ingresar a laborar en dos empresas y en ambas fue requerido para que se presentara a trabajar, eligiendo a MIXE porque supuso que aprendería mas en la empresa MIXE por los proyectos informáticos que ésta empresa está desarrollando.

> "... fui a mis exámenes a las dos. Afortunadamente en las dos me quede...¿Y ahora qué hago?, ¿dónde me quedo? Yo lo ví a futuro y me incline por éste proyecto... «aquí voy aprender más, aquí tengo mayor posibilidad de subir de puesto...creo que a futuro es más posible que avance, viene desde cero el proyecto...»" (Programador GG3 MIXE)

El *sentido subjetivo del trabajo como aprendizaje* el programador le percibe y comprende el aprendizaje como un saber-hacer que representa habilidades y destrezas para *saber resolver* problemas. Al respecto el programador GG3 señala que es importante el aprendizaje acumulado, porque constituye experticia y aquel programador con experticia estará en condiciones de por mejores condiciones laborales:

> "...si me dicen te doy ocho (mil) digo bueno, quiero un aumento, -les voy a reclamar- les voy a pedir más, pero sí no me lo dan, sí me quedo, porque por el hecho de que estoy aprendiendo y eso me va servir, para que si en otro proyecto, llego y les digo: quiero quince (mil) ahora y, ellos me dicen que no, yo me voy a buscar un proyecto en donde me den más dinero. Porque digo OK, yo merecía haber ganado doce (mil) me diste ocho (mil) ya tengo más experiencia y a lo mejor es lo que me podrían decir sabes qué, no te voy a dar los doce que quieres o los doce que estas estimando porque no tienes experiencia" (Programador GG3 MIXE)

Otro elemento que consideramos en el *sentido subjetivo del trabajo como aprendizaje* es aquel que se construye en el plano de las *prácticas subjetivas* de los individuos que hacen referencia al conjunto de *satisfacciones relativas* como el cumplimiento de intereses, intenciones o motivaciones que conducen a los individuos a intensificar o reducir las interacciones en las *relaciones de saber-poder*. El conjunto de *satisfacciones relativas* estarían dadas por las percepciones de familiarización con los conocimientos utilizados; aprendizaje de nuevos conocimientos (dentro o fuera del proceso de trabajo), habituación a los sistemas de producción en la empresa, etc. La insatisfacción por el aprendizaje puede conducir a reducir el componente de poder en proceso de trabajo o bien renunciar a la empresa o "ciclarse" en el trabajo.

El conjunto de *satisfacciones relativas* y la acciones subsumidas en las practicas laborales y extralaborales emprendida por los individuos, constituye parte fundamental de las *dinámicas relacionales* y paralelamente son parte integrante del *sentido subjetivo del trabajo como aprendizaje* ambas forman parte de una serie de juegos, consensos y negociaciones que acontecen en el proceso de trabajo (Burawoy); es decir se traducen en una forma de juegos de poder, con tensiones, pugnas y conflictos, que denominaremos por el momento como juegos por *resolver a tiempo*[109].

En esta industria el trabajo podríamos definirlo como una *fluidez cognitiva* donde *saber- resolver a tiempo* un problema dado es medular, lo cual no quiere decir que la solución sea la mas eficiente (incertidumbres que ya definimos en el anterior apartado), la variable tiempo es una condición del proceso de trabajo, medido el tiempo en horas-hombre que en muchos de los casos sin negociadas para resolver algoritmos (en algunos casos se impone el tiempo por parte de la empresa). Al respecto, señalaba el programador de plataformas educativas en software de código abierto AM5 de profesión psicólogo, que los programadores requieren poner en prácticas capacidades de orden superior como la abstracción para resolver problemas:

> "...un programador, lo que tiene que hacer es solucionar problemas... la solución de problemas, la abstracción son de orden superior, la memoria y el recuerdo es de orden inferior...uno de los problemas más frecuentes en la solución de problemas, es que uno llega a una saturación cognitiva, es decir, uno no encuentra la solución del problema y le da vueltas y vueltas, hasta que algo dentro de nosotros, ocurre, que se reacomoda...algo que nosotros llamamos gestalt, es decir, reorganizamos el medio ambiente para obtener una nueva visión del problema" (Académico AM5 ENEX)

El *juego de resolver a tiempo,* implica varios temas, uno de ellos es el conflicto que no abordó del todo Burawoy como si lo hicieron Edwards y Thompson. Sin embargo, en el presente apartado, interesan comprender las *reglas informales* que forman parte del juego de *resolver a tiempo*; juego que se constriñe a una serie de complejidades que tienen que ver con el hecho que los programadores se resisten a formalizar (documentar el proceso) cual es el procedimiento abstracto (algoritmos) que se implemento para solucionar un problema. Esta formalización se concreta mediante un conjunto de *textos sígnicos* que son las líneas de

[109] Lo cual no quiere decir, que las satisfacciones relativas sean el conjunto de las interacciones que convergen en el proceso de trabajo del software, sin embargo nos concentraremos en este punto, ya que el anterior apartado se lo dedicamos a la interacción cliente-líder de proyecto.

código, mismos que deben ser consistentes con la lógica hacia el interior del algoritmo y hacia el resto de las soluciones que se presenten (compatibilidad con otros módulos), donde tanto la ausencia de inconsistencias como de incompatibilidades representan una condición del *significado de resolver a tiempo*, que no es otra situación que cumplir con resolver el problema y entregarlo en las fechas señaladas.

> "Pues mi principal preocupación es que hagamos las cosas en los tiempos planeados, que se sigan los estándares que se pidieron (cumplir con los requisitos), que nadie los este evitando... si veo que algo aunque lo estén haciendo y funcione, si yo se que se puede hacer mejor... decirles como debería hacerse, te digo la programación es libre, se le deja su criterio pero se puede asesorar a la persona en como podría mejorar." (Programador JO3 MIXE)

Una tercera situación del *sentido subjetivo del trabajo como aprendizaje* tiene que ver con el hecho que *resolver a tiempo* un problema es que no necesariamente quiere decir que el *texto sígnico* sea inédito, sino que un porcentaje importante se reutiliza. Según Pressman (2002), se estima que en un sistema informático poco mas del 25% es software reutilizado; una de las *normas disciplinarias* de las Ingenierías del Software es reutilizar lo mas posible algoritmos a través de producir liberarías de código que estén documentadas, o bien elaborar repositorios de código, con las cuales sería posible disminuir costos, posibles atrasos en tiempo o errores en el código desarrollado; ya que las bibliotecas y repositorios de códigos reutilizados demuestran ser consistentes, compatibles y no presenta errores de interoperabilidad (Programación orientada a objetos). Lo cual no quiere decir que sea el código idóneo, pero funciona. Al respecto el programador experto MZ3 señalaba que la asignación de tiempos de trabajo es por experticia acumulada o destrezas en la reutilización de código (Programador MZ3 TADI), sin embargo la reutilización de código en algunos casos implica problemas, fallos, ineficiencias:

> "... siempre se reutiliza (código) de hecho la lógica de una alta, de una baja o de un reporte es que siempre la estas reutilizando... aunque hay veces que lo hace ineficiente, bueno normalmente lo hace un poco ineficiente.... supón que tu tenias un menú de cierta forma y entonces le metes alguna imagen o algún otro tipo de menú, una barra de herramienta, etc. entonces siempre reutilizas lo que ya hiciste, yo creo que lo rico de la programación es eso, que siempre estas reutilizando y normalmente estas reinventando, bueno no reinventando sino creando cosas nuevas" (programador MZ3 TADI).

El reutilizar código no limita la abstracción cognitiva de *saber resolver* un problema o que minimice los tiempos para resolverlo, ya que la propia reutilización implica "saber" leer el código, tener presente el código desarrollado, y poseer habilidades y destrezas para *saber resolver* los inconvenientes, como la falta de documentación del código, detectar el código oculto, entre otras dificultades.

Las *prácticas sociales* simbolizan una serie de juegos y consensos de resistencia, conflicto y boicot, para que "otros" no "accedan al *saber hacer* de como se resolvió determinado problema". Por ejemplo en la entrevista con el experto freelance DINS quien señala la existencia de programadores "Shadow", que solucionan problemas complejos que no resuelven los equipos de las empresas, entonces se subcontrata el programador shadow. Comenta que en una librería de código que se reutilice casi el 95% del código desarrollado y se comprenda, pero puede haber un 5% realmente complejo y confuso de comprender:

> "Hay 1000 pedazos de código... donde 900 líneas son librerías (reutilización) y las ultimas 100 líneas son la aplicación, pero el cachito de 25 líneas me pongo a leerlo y digo « "&*# que hizo este» sin embargo, hay veces, incluso sucede dentro de compañías... que contrataron a otra compañía que les hizo un modulo y nadie se atreve a moverlo... porque nadie se atreve a tratar de entender que es lo que sucedió ahí... entonces depende mucho del coeficiente intelectual de la persona que esta leyendo el código... o lo que quiso escribir la otra persona, necesitas un razonamiento lógico un poco mas avanzado, un planteamiento matemático un poco mas avanzado para aprender a leer código" (programador SAR3 DINS).

Una cuarta circunstancias del *sentido subjetivo del trabajo como aprendizaje* es el hecho que *saber resolver* implica un doble juego hermenéutico, que tiene que ver con el proceso de abstracción *que resolver* y *como resolver*. El primero hace referencia a una serie de consensos en convenir entre el líder de proyecto y el programador en consensuar el contenido del requerimiento que se resolverá (cuales son contenidos); el segundo tiene que ver con la interiorización y aprehensión del *como resolver* (como crear los contenidos). El *que resolver* está implícito en la lista de requerimientos, la información contenida en los requerimientos es reinterpretada por el programador, donde la reinterpretación del *como* se resuelve representa un problema, porque ello significa acudir a una *constelación de configuraciones subjetivas* para saber resolver la acumulaciones de saberes, junto a relaciones de poder, etc. La solución implica una serie de *reglas no formales en el juego de resolver pero* es través de las *interacciones sociales* y virtuales y no donde se ingresa al juego de resolver, a

los arreglos sociales; pero en este juego las *dinámicas sociales* del aprendizaje, experiencias, habilidades, destrezas, etc. forman parte activa del juego, que hace alusión al proceso de internalizar, interiorizar, aprehender el problema a resolver.

> "muchas veces sabes que las cosas se pueden hacer, a veces no tienes el conocimiento total de como se hace, pero tienes los principios, sabes por donde empezar y si necesitas ayuda en ciertas cosas vas y lo consultas, ya sea en un libro, ya se en Internet y de ahí ya vas viendo como se resuelven las cosas..." (Programador JO3, MIXE)

8.3.2.- Las reglas informales en el *saber resolver a tiempo*

En 1939 Roehtisberger y Dickson (citador por Burawoy, [1979], 1989:105) señalaban que "los trabajadores tiene sus propias reglas y su propia lógica, que en la mayoría de los casos están en contradicción con las que les han sido impuestas" en este mismo sentido Pierre Tripier en 2003[110] planteaba en el 4° congreso de sociología del trabajo que:

> "...para descubrir el origen de un defecto, se necesita la cooperación entre ingenieros y obreros, y para lograr esta colaboración cada uno debe respetar el saber del otro, el entendimiento pragmático, local y practico de los obreros, el saber abstracto, general y matemático de los ingenieros. La *igualdad cognitiva* no significa que uno puede ocupar el papel del otro, sino que cuando hay un defecto de resolver, solo la colaboración de los dos permite resolverlo, el ingeniero puede entonces aprender del obrero y la reciprocidad debería existir." (Tripier, 2003:12)

Esta cita de Tripier, en cierto sentido es similar a Foucault, en el entendido que la *igualdad cognitiva* no significa que el obrero puede ocupar el lugar del ingeniero y viceversa, no es que haya una lucha cognitiva entre ambos, sino que es a partir de una colaboración que resuelve el problemas que suscitó dicha *interacción sígnica*, después se disuelve al estilo señalado por Foucault, quien indica que el *poder* no intenta ocupar el lugar del *saber*, sino que ambos existen y coexisten uno frente al otro, señalamientos que denotan más una *alineación del saber frente al poder*, del obrero frente al ingeniero, que colaboran y cooperan para resolver problemas. Esta aportación de Foucault tiene limites, cuando observamos que las relaciones laborales no están exentas de consensos, juegos, negociaciones que irrumpen esta alineación, son las reglas informales no escritas, como el *juego de saber resolver* que irrumpen

[110] Mercier, Delphine y Pierre Tripier, 2003, El Neo-Management y la ceguera Organizacional. Ponencia presentada en Asociación Latinoamericana de Sociología del Trabajo, 4° Congreso La Habana 9-13 de Septiembre de 2003.

las relaciones de poder en el proceso de trabajo, proceso que Burawoy explicó categóricamente en 1979, y después otros investigadores han explicado de una forma sobresaliente como Nonaka y Takeuchi, Reynaud, Dubar, entre otros, que coinciden en señalar que en el proceso de producción (manufactura, servicios, etc.) coexisten otros tipos de conocimiento, uno que esta relacionado con el *saber-hacer* como parte integrada de la practica cotidiana, del *saber resolver* problemáticas particulares del proceso de trabajo (conocimiento tácito) y, por otro lado, el de la gerencia que permite contextualizar la problemática presentada en el proceso de trabajo mediante normas y procedimientos formales (conocimiento explícito). Lo cual no quiere decir que ambos conocimientos sean excluyentes, o que estén separados, de hecho ambos están yuxtapuestos sea en el trabajador o en la gerencia, son complementarias, cooperativos; construyéndose una interacción colaborativa que Taylor y Ford no separaron, no dividieron del todo, sólo la redujeron, la restringieron, pero no la desaparecieron (Tripier:1991).

Una quinta y última consideración del *sentido subjetivo del trabajo como aprendizaje*, es el supuesto de la *fluidez cognitiva* entre aquellos agentes que participan en el proceso de trabajo. Fluidez que va más allá del planteamiento de Tripier. El concepto de *fluidez cognitiva* que ya explicamos en el esquema 10, nos permite explicar algunos aspectos del proceso de construcción de los algoritmos entre los espacios de producción, de las *interacciones sígnicas* del saber hacer y de las interacciones de poder. Saber resolver los problemas planteados en los requerimientos no es aislado, no es al estilo de la programación artesanal, por el contrario, como señalábamos párrafos arriba, la programación moderna está mas estrechamente relacionada con la colaboración, pero no del tipo de la cuarta ola gerencial del managment, sino una colaboración generada desde abajo por los propios agentes que participan en el desarrollo del software, una colaboración conflictiva por un lado y cooperativa por otra; una colaboración frente a frente en el proceso de trabajo y pantalla a pantalla, tanto dentro del proceso de trabajo como fuera del proceso de trabajo. Es decir una colaboración construida socialmente a partir de interacciones individuales, colectivas, concretas en el trabajo y abstractas en el proceso de creación que forman parte de la las comunidades simbólicas.

Entendiendo por estabilidad al hecho de reducir la incertidumbre presente, que no haya conflictos de configuración, operabilidad e interoperabilidad en los módulos resueltos. En otras palabras *saber resolver* los problemas que plantea el programa informático, involucra

una serie de *juegos por resolver* los problemas, solución que implica fluidez de información, conocimientos y de saberes entre los espacios de poder (diseño y desarrollo), ya que si no existe un conjunto de interacciones y relaciones de poder entre ambos espacios cognitivos que permitan resolver los módulos o rutinas especificas acorde a las fechas del proyecto informático, de lo contrario el proceso de hace conflictivo en el sentido de que la incertidumbre aumenta. Al respecto el líder de proyecto JM2 de la empresa DYYA con presencia en Latinoamérica, señalaba que no se le deja al programador sólo en la resolución del problema, sino que se consensan los tiempos de desarrollo, los contenidos y posteriormente se resuelve como podría desarrollarse el sistema:

> "... incluso...pues... lo hacemos en consenso... no dejamos de que «oiga haga esto»...hay una consideración entre ambas áreas (diseño y desarrollo) «oye si lo ponemos acá ¿vemos el mismo resultado?, ¿mantenemos?, ¿esta limpio?, etc.»... «no pues... adelante...entonces...» se queda como una definición completa... después de esa reunión de esa sesión hay una definición completa del... cómo se va a resolver" (Líder JM2 DYYA)

Ahora bien la distribución de tareas (rutinas de código por resolver) y los tiempos asignados para resolver los módulos que integran el sistema informático se sucede en un contexto de negociación con consenso entre gerentes, lideres/administradores de proyecto, programadores y entre programadores.

> "normalmente cuando hace una propuesta de un proyecto, la mayoría de las veces es importante ver con el programador que posibilidades tiene, porque aunque uno ya tiene la experiencia y dice «no pues esto va a estar en treinta días» -por el cierto conocimiento que ya tienes de lo que sería el desarrollo y los tiempos de programación-. Si es importante involucrar... al programador, para que este conciente de que si estamos definiendo una propuesta a treinta, sesenta días o un año, ese período se pueda cumplir, es decir yo trato de involucrarlo...digo «oye traigo esta propuesta nos vamos a aventar treinta días ¿se puede o no se puede, tu que piensas?...» es negociar con el programador" (Líder RS2 TADI).

Consenso y negociación, juegos y arreglos (Burawoy), en el sentido que los trabajadores y gerentes establecen una serie de juegos basados en primera instancia o partiendo de una regla no escrita: *fluidez cognitiva*; fluidez que no significa que el diseñador deba conocer el *como y porque* se va resolver un problema, sino que sólo conoce el *que* resolver, pero el *como y el porque* es tarea de los programadores, es decir fluidez en términos de la información que fluye del diseño hacia el proceso de trabajo como un conjunto de

requerimientos que deben ser traducidos en *textos sígnicos*; interpretación que ser traduce en conocimientos, conocimientos que se amplían como aprendizajes a través de una serie de juegos de saber hacer y saber poder. Al respecto Ditton, señala que las reglas al interior del proceso de trabajo se fortalecen con la participación activa de los estratos menores de la gerencia e incentivando la presentación de juegos (citado por Burawoy, [1979], 1989:106). Burawoy concluye y demuestra en su amplia obra, que cuando se llevan a cabo los juegos éstos no son independientes y no se conspira contra la empresa; por el contrario se lleva a cabo un proceso complejo entre gerentes y trabajadores, para que cada uno de los "jugadores" desarrolle habilidades, destrezas y conocimientos necesarios para *arreglárselas* en el cumplimiento de las exigencias productivas (numero de piezas, cantidad de operaciones técnicas, etc.)[111].

> "Con la anterior gerente no había negociación por que ella conocía muy bien los procedimientos y conocía bien la estimación y sabía cuanto nos tardábamos cada uno...no había que decirle «me voy a tardar tanto»... Con la nueva gerente no es igual porque ella no estuvo involucrada en el proyecto desde el principio, entonces con ella si tengo que plantearle que es lo que estoy haciendo y porque me tardo tanto. No es una negociación, sino explicación, sino informarle a ella para que vaya viendo como estimamos en ese proyecto" (programador CB3 DASA)

La *fluidez cognitiva* esta compuesta por información y comunicación, además de la *constelación de configuraciones subjetivas* que intervienen en el proceso de trabajo, aquí juega un rol importante el líder de proyecto quien no solo "vigila" el desarrollo-solución de los problemas planteados por el programa informático, es el interlocutor con la empresa para re-estimar los tiempos de trabajo en caso de atraso, es quien justifica, amplia, restringe o reprime al programador en caso de atrasarse en la solución informática. El programador explica, si el problema es complejo y requirió más tiempo del estimado, si tiene errores en el desarrollo y no le es posible solucionarlo, etc. (programador CB3 DASA-ET1). En el *juego de saber resolver a tiempo*, como estamos dando cuenta interaccionan los diseñadores del proyecto, administradores, líderes, programadores, documentadores, tester de calidad, entre otros. En este juego, el *saber resolver a tiempo* un algoritmo exige capacidad de abstracción, habilidad cognitiva para solucionar, es decir los programadores en el desarrollo de software deben

[111] Con el fin de remitir a la complejidad del juego de resolver a tiempo, el siguiente sub apartado (8.2.1) lo dedicaremos al concepto de uso común entre los programadores de software "tener lógica de programador" como una especie de "caja negra" que no se puede explicar del todo pero resuelve el problema.

solucionar el problema, en interacción con los otros, pero son los programadores quienes deben *arreglárselas* para escribir las líneas de código y así *saber resolver el problema*; es verdad que puede participar un programador mas hábil, o bien acudir a librerías, ayudas en comunidades virtuales en Internet, etc. pero en el proceso son los programadores quienes crean el algoritmo, son los responsables de la solución planteada. Este concepto prestado de Burawoy, el de *arreglárselas* nos permitirá explicar parte del juego informal de *saber resolver a tiempo*.

> "si no se entregan las cosas en los tiempos en los que ya se planificaron pueden haber penalizaciones...la empresa puede tomar ciertas reconsideraciones conforme a sus programadores y pues no tenemos un contrato fijo y no somos indispensables. Tanto podemos tener un proyecto del inicio hasta al fin, como podemos salir si no somos capaces de desarrollar las tares que nos asignan (programador JO3 MIXE)

El *arreglárselas* para *saber resolver un problema* tiene que ver con el uso de las *herramientas procedimentales* que definimos al principio, con los lenguajes de programación, con el uso correcto de las plataformas tecnológicas, e incluso una negociación densa entre el líder o el diseñador del módulo con el programador, bajo la posibilidad de que el planteamiento del diseñador no siempre es el más idóneo:

> "... el modulo que a mi me asignaron era un poquito complejo, como me lo había presentado el arquitecto y en base al lenguaje (de programación) y la tecnología que estábamos usando (Bases de datos) no era la mejor forma de hacerlo, platique con el arquitecto y le hice ver que la mejor forma era por otro camino y me dieron luz verde... los tiempos que yo tenia asignados los rebase, entonces estaban con la premura...el modulo se tardo unas dos semanas más de lo que debía, (aunque el módulo era para dos proyectos)... tenia ya la presión sobre mí.... termine el modulo, lo presente y la implementación termino en 5, 6 minutos; dijeron «a caray, valió la pena, pero en este proyecto se retraso»...dijeron «bueno haber, impleméntalo en el segundo proyecto, "haber si es cierto"», entonces, llego, lo implemento en el siguiente proyecto y lo termino en 5 minutos, me dicen «oye esta súper rápido de implementar», "sí por eso me tarde dos semanas mas, ahora puedes reutilizar este otro modulo en cualquier otro proyecto". Ahí vieron el valor de esas dos semanas, cosa que no contemplaron desde la proyección o desde el diseño, y ese es un ejemplo de cómo llega a afectar una decisión del programador al diseño de un arquitecto (programador freelance BB3 BICS)

Burawoy ([1979], 1989:106) demuestra que no basta una orientación instrumental para llevar a cabo el *arreglárselas* en el proceso de trabajo, sino que existe todo un conjunto de

reglas informales que constriñen el juego; al respecto gerentes y lideres no sólo colaboran en el cumplimiento de las reglas sino que participan activamente en el juego, por ejemplo ampliando las horas de trabajo de un módulo, cuando observan que el programador no cumplirá con los tiempos de entrega, agregan un remanente de horas a favor del programador, para que así el programador cuente con un tiempo "extra" para prevenir cualquier incidente en el cumplimiento de *resolver a tiempo*, además que este remanente de horas en caso de atraso no significaría costos para la empresa, puesto que ya están estimados; otro *arreglo social* no escrito es el hecho de permitir que el programador cuente con acceso al chat en horas de trabajo, tenga cuentas personales (Yahoo, Hotmail, etc.); el programador por su lado, cede espacios de su vida fuera del trabajo, resuelve parte del problema en los espacios domésticos o de recreación; investiga, se informa autoaprende por cuenta propia en libros, tutoriales, chat-room, blogs personales, entre otros tipo de actividades virtuales. Por ejemplo, existen foros de discusión que ofrecen espacios de discusión especializada, sin importar el modo de desarrollo de software que se utilice los programadores o empresas (de código privado o libre), por ejemplo http://www.foromsn.com/index.php?Ver=Foro&Id=6&VerEtiqueta=34 que ofrece distintas discusión en torno a las *herramientas procedimentales*, lenguajes de programación como: Visual Basic, Java, C y C++, PHP, SAP, Batch, Delphi, MySql; Bases de Datos como: Binario/Ensamblador, JavaScript, MS-Dos, .Net; etc. entre otras ligas de interés.

El programador resuelve parte del trabajo en espacios virtuales, labor que no necesariamente se realiza dentro de la jornada de trabajo. Al respecto el programador CM3 de la empresa MEKI-PT2 señalaba que cuando no le es posible resolver parte del problema acude primeramente a los foros de programadores, de no encontrar alguna solución que le oriente, se remite a Blogs o wikis personales o espacios de discusión de programadores de una serie de colegas que ya están en la "lista de preferidos" que mas frecuenta (CM3, MEKI-PT2); por su lado el programador JO3 de la empresa MIXE, señala que en Internet existen diversos foros de discusión de variadas empresas desarrolladoras como son SAP, JAVA ó Microsoft, entre otras empresas que ofrecen tutoriales para programar en sus respectivas plataformas tecnológicas de desarrollo, también ofrecen cursos y seminarios para certificarse en éstas plataformas. Los programadores acuden a este tipo de foros de forma cotidiana, o bien cuando no les es posible solucionar X problema, ya que aseguran que siempre hay otro programador que ya se enfrentó a un problema parecido. En dichos foros existen listas de discusiones,

donde se puede leer y examinar el *como* se resolvió parte del problema o un proceso similar, ya que en caso de que no haya una respuesta satisfactoria o no exista parte del problema, se "postea" la pregunta, las dudas, señalando la problemática a resolver, para que alguien asesore o aporten posibles respuestas, "tipo lluvia de respuestas"; sin embargo se reconoce que no habrá respuestas satisfactorias, sino parte de la respuesta (JO3 MIXE-ET1), este conjunto de *operaciones subjetivas* virtuales y reales, ya sea hacia el interior del trabajo como hacia fuera, ya sea cara a cara o pantalla a pantalla, conforman parte de un juego: El *arreglo social*.

8.3.3.- Arreglárselas para saber resolver

Burawoy ([1979], 1989:107) señala otra característica importante en la participación del juego en el proceso de trabajo el "arreglárselas", tiene que ver con el deseo de participar en el juego "...nace tanto de la necesidad de trabajar y atenerse a las exigencias del proceso productivo en el lugar de trabajo como de la aparición de necesidades radicales o de una nueva concepción del trabajo..." los trabajadores participan en el juego para obtener un conjunto de *satisfacciones relativas* que tienen que ver con el rompimiento de la monotonía, el desgaste físico, minimizar el aburrimiento. Para el caso de la industria del software señalamos que prevalece un desgaste cognitivo, un conjunto de relaciones de saber y de poder basados mas en las habilidad y destrezas de *saber resolver* problemas mediante proceso abstractos.

Como ya señalamos consideramos que la *fluidez cognitiva* entre los espacios laborales (cliente, gerentes y trabajadores) permite no sólo una colaboración y participación de los agentes, sino también un flujo de información y conocimiento entre dichos espacios, fluidez que esta embebida en una serie de conflictos, resistencias, consensos, negociaciones que hemos denominado como *constelación de configuraciones subjetivas* que intervienen en el proceso de trabajo; Burawoy ([1979], 1989:106) resalta el hecho que la colaboración entre gerencia y trabajadores se traduce en apoyos mutuos, a través de consensos, negociaciones, colaboraciones; Al respecto el programador CB3 de la empresa DASA, señalaba que acuden a los foros, blogs o wikis en Internet, donde diferencian las distintas posibilidades de resolver: aprecian y conversan los diferentas procedimientos al leer la información que resulta de la charlas virtuales; después de reflexionar la información disponible el programador será quien tenga "el último algoritmo" (programador CB3 DASA). Podemos decir que los programadores establecen *nuevas formas de aprendizaje situado* en forma virtual, ya que en los foros, blogs o

wikis se aprende interactuando, ya sea leyendo, aportando, resolviendo dudas, etc.; en otras palabras acontece una retroalimentación del conocimiento como aprendizaje, el cual se sucede no resolviendo el problema, sino en una serie de diálogos virtualizados, que se traducen en "tips" para quien acude a dichos foros (CB3 DASA-ET1).

> "casi diario ando buscando información relativa a cosas que no conozco...hay muy pocas cosas que sé y muchas que desconozco, entonces veo como lo han hecho otras personas, y veo si lo hicieron bien o lo hicieron mal, o si lo puedo mejorar yo, o si me sirve eso para lo que estoy haciendo, a veces, si es que todavía no tengo la solución" (programador JO3 MIXE)

Este *arreglo social* es por ambas partes, por un lado la gerencia ó el líder de proyecto reconocen que la solución de un problema lógico no se resuelve necesariamente en el periodo de tiempo de la jornada de trabajo, sino que la cognición del problema se extiende más allá de los limites de la jornada y fuera de las paredes o espacios de la empresa, quizá los mejores ejemplos son los programadores freelance, o los amplios repositorios de código existente en los Chat-room de programadores (Microsoft, Debian, Ubuntu, etc.) o las explicaciones sencillas entre programadores para compartir soluciones a problemas planteados en Internet. A continuación extraemos una dialogo entre un programador junior (programador con escasa experiencia) que está desarrollando un software a la medida para una bodega de préstamos de herramienta y se enfrenta a un problema y acude al foro:
http://www.todoexpertos.com/categorias/tecnologia-e-internet/programacion/respuestasa,
donde le apoya un programador experto en la posible solución del problema posteado:

> 22/11/2007 Usuario pregunta:
> Administró la Bodega de Préstamos de Herramientas en una Empresa de Mantención. Todo el control de préstamos lo llevo con un formulario en papel que traspaso a Excel. No he tenido problemas pero quiero automatizar el servicio y poder llevar un registro más fácil de las herramientas que solicitan, devuelven o tienen pendientes de devolución. ¿Me podrías dar una pauta por donde empezar, o si tienes algún ejemplo de donde extraer ideas te lo agradecería, Saludos. Juank
>
> 22/11/2007 Experto:
> Por su puesto que si. Yo tengo varios sistemas, mi especialidad es fileMaker y por lo que veo quieres un control de vales. Lo único es que son archivos que necesitan de 20 megas para ser enviados. El archivo no es grande es muy pequeño pero el problema es que necesitas FILE MAKER para "correrlo".
>
> RE: 23/11/2007 Usuario:
> Muchas Gracias por contestar, con respecto a Filemaker tengo una versión, me parece que es la 8. Agradecería mucho que me pudieras ayudar, mi correo es juancarlos.lemunir@gmail.com si me puedes enviar esos ejemplos mencionados para poder adaptarlo a lo que necesito te lo agradeceré un montón, este tema me atrasa mucho en mi

trabajo y me queda muy poco tiempo para otras cosas; si es muy "pesado" lo puedes comprimir con hacha o winrar.

Re 23/11/2007 Experto:
Excelente yo tengo las versiones 3, 5, 5.5 developer, 6, 7, 7 developer , 8 y 8.5 developer. Te voy a enviar un programa de vales que tengo espero te sirva.

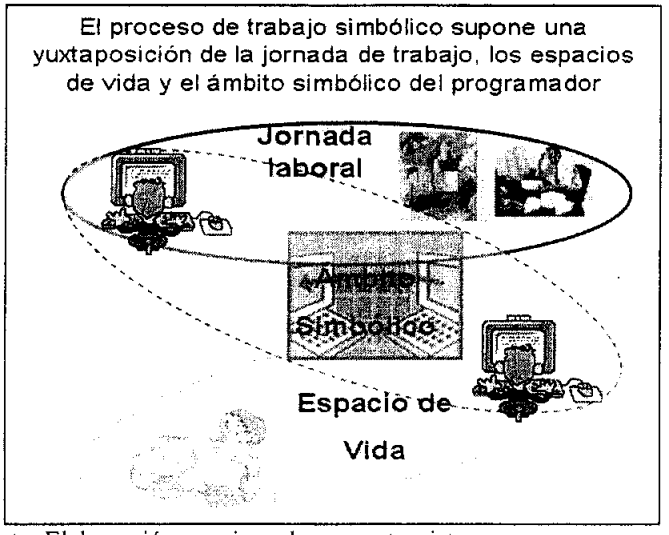
RE 23/11/2007 Usuario:
Excelente, muchas gracias por el Archivo, una consultita con respecto al sistema de Vales, ¿Existe la posibilidad de que al borrar quede en una especie de Historial del Cliente y que cuando busque el cliente me muestre solo lo que debe?

Re: 23/11/2007 Experto:
Por supuesto que si, varias soluciones, pero por cuestión de tiempo te voy a contar de 2.
La mas fácil es que le pongas un campo que contenga 2 valores como ACTIVO / CANCELADO ó algo por el estilo; de esta forma cuando le des buscar solo mostrara los ACTIVOS. La otra forma es crear un archivo gemelo del vale y utilizar la opción IMPORTAR desde el que ahora será histórico, y si tienes el filtro activado para CANCELADOS se va todo al histórico. Te recomiendo la primera forma por ser más fácil. La otra no es complicada pero si se requiere un poquito de tiempo el llevarla a cabo de forma totalmente automatizada que con solo darle un botón elimine del archivo actual y mande al histórico.

Re: 23/11/2007 Usuario:
Muchas Gracias, me ha sido de mucha ayuda, aquí seguiré perfeccionando el sistemita, espero poder contar con tu ayuda más adelante.

El ejemplo expuesto, nos permite observar el proceso de *aprendizaje dialógico* a través de una Chat-room, el intercambio entre un programador junior y un experto, por lo pronto retomamos que, el programador es quien presionado por el tiempo y por la necesidad de *arreglárselas* para resolver el problema que acude a los foros, foros que se constituyen como un *arreglo social* mas allá de las fronteras del proceso de trabajo. Mas adelante volveremos sobre este tipo de comunidades de aprendizaje virtual, que siguiendo a Edwards ([1982], 1987) y Thompson, consideramos que contribuyen al proceso social de la *construcción social de la ocupación*. Ahora bien, la búsqueda en la solución de un problema implica no sólo el juego de *saber resolver a tiempo* y las reglas no formales de *arreglárselas* y los *arreglos sociales* sino también la participación de la gerencia en el juego a través del hecho que el proceso de trabajo supone una yuxtaposición de la jornada de trabajo y una ampliación de la jornada de trabajo que comprende los espacios de vida y de recreación del programador. Superposición que no es ignorada en el proceso de trabajo, proceso que supone una ruptura entre las fronteras del trabajo y no trabajo, ampliándose la jornada de trabajo hacia los espacios de vida del programador (véase esquema 11).

Esquema 11



Fuente: Elaboración propia en base a entrevistas

Para el programador CM3 de la empresa MEKI-PT2, señalaba que el compromiso de *saber resolver a tiempo* la tarea, implica una estimación del tiempo "...si es sencillo, si lo saco", lo cual puede no ser cierto, por tanto significa trabajar mas de ocho horas, desvelarse y la empresa no paga horas extras (programador CM3 MEKI-PT2), mas adelante comenta -como si fuese una ventaja- que ahora ya poseen computadoras portátiles para hacer el trabajo en casa y resolver los problemas a tiempo.

> "...la mayoría los programadores o mas bien todos ya nos dieron lap top´s, entonces antes era «lo que acababas hoy más o menos y ya» pero ahora esta el compromiso «Ah lo tengo que entregar mañana» pero ahora, tienes la facilidad de llevarte la lap top, sólo tienes que llegar a tu casa, abrir la lap top «y a darle» ya en la noche, para entregar las cosas a tiempo" (programador CM3 MEKI).

Para Burawoy, los juegos establecidos por trabajadores y manos medios en el piso de la fábrica no se constituyen contra la empresa, como señalan Elton Mayo y Cornelius Castoradis (Burawoy, [1979], 1989:107), no obedecen a luchas del tipo instrumental, que tengan por objetivo límites definidos por necesidades salariales; están otro tipo de consideraciones, sentidos y percepciones hacia el trabajo, nos referimos a los deseos, motivaciones y necesidades radicales de una nueva concepción del trabajo, de un código alógico así como la obtención de *satisfacciones relativas*. Estas ideas de Burawoy en parte nos auxilian para comprender el proceso de trabajo cognitivo del programador de software, en el entendido que las empresas estudiadas, están más interesadas en el juego de *saber resolver a tiempo,* bajo las normas informales de *arreglárselas*; donde el *arreglárselas* representa un

trabajo fluido entre los que intervienen en el proceso de *saber resolver*; donde dicha *fluidez cognitiva* no necesariamente corresponde al contexto de la jornada de trabajo, sino que también implica romper con la regulación de la jornada de trabajo, supone una yuxtaposición entre mundos de trabajo y mundos de vida del programador. Por ejemplo un programador que por razones de falta de experiencia, información disponible, etc. no alcanza a solucionar un problema especifico, es decir que estuviese "ciclado" (término cotidiano entre los programadores para señalar que no les está siendo del todo fácil resolver algún algoritmo del módulo) motivo por el cual acuden a otro individuo, el cual puede ser el compañero de trabajo, el líder del proyecto, el programador senior, el programador-virtual, (contacto que no necesariamente es cara a cara, o en tiempo real), comunicación que puede ser del tipo sincrónica y también diacrónica; por ejemplo puede acudir con el compañero de trabajo, con el programador senior responsable del equipo de trabajo, o acudir a Internet y acceder a información disponible en blogs, portales del tipo wiki, etc. consiguiendo la información que le nutre, ayuda u otorga ideas o conceptos para resolver parcial o totalmente el problema. Este conjunto de decisiones-acciones del programador van más allá de una alineación instrumental, se colocan más en el aspecto de las satisfacciones relativas que ya denominamos al principio de este apartado como sentido subjetivo del trabajo como Aprendizaje por parte del programador, medida subjetiva que podría determinar si continúa laborando o no en la empresa (ver esquema 11).

Otro consenso no formal es *lógica de programador* regla que esta contxtualizada por las *constelación de configuraciones subjetivas* que intervienen en el proceso de trabajo como los deseos, motivaciones y necesidades radicales de una nueva concepción del trabajo, de un pensamiento dialógico que tiene que ver con la obtención de *satisfacciones relativas*, que en conjunto son resultado de un contexto de relaciones de poder, luchas, contradicciones, consensos y negociaciones que se propagan no solo en el proceso de trabajo, sino que se expanden mas allá de la frontera de la jornada de trabajo y abarca los tiempos de vida del programador y los ámbitos simbólicos en los que interactúa. Entendiendo a estos ámbitos simbólicos como aquellos espacios privados de autoaprendizaje, espacios virtuales de interacción que conforman parte de su saber-hacer, espacios de poder que no necesariamente comparte con los otros programadores. El conjunto de consensos y juegos a través de reglas no formales como el *saber resolver a tiempo, arreglárselas, arreglo social* que no está libre de

conflicto, resistencias y luchas por el saber hacer y el control del poder sobre el proceso de trabajo; resistencias que mas tarde abordaremos como *conflicto no dirigido*, soterrado, oculto-presente entre los juegos de resolver; otro conflicto es la *resistencia a documentar* el proceso o metodología del código, o bien el *complot* intrínseco en la elaboración de líneas de código como: *ocultar código*. Conceptos que construimos basándonos en Edwards P.K. y Hugo Scullion ([1982], 1987) y en Baethge Martín y Herbert Oberbeck ([1986], 1995) y abordaremos en el siguiente capitulo.

Capitulo IX

Subjetividad, estructuras y acciones en el proceso de trabajo:
Una reflexión desde la subjetividad del conflicto como resistencia

9.1. Introducción

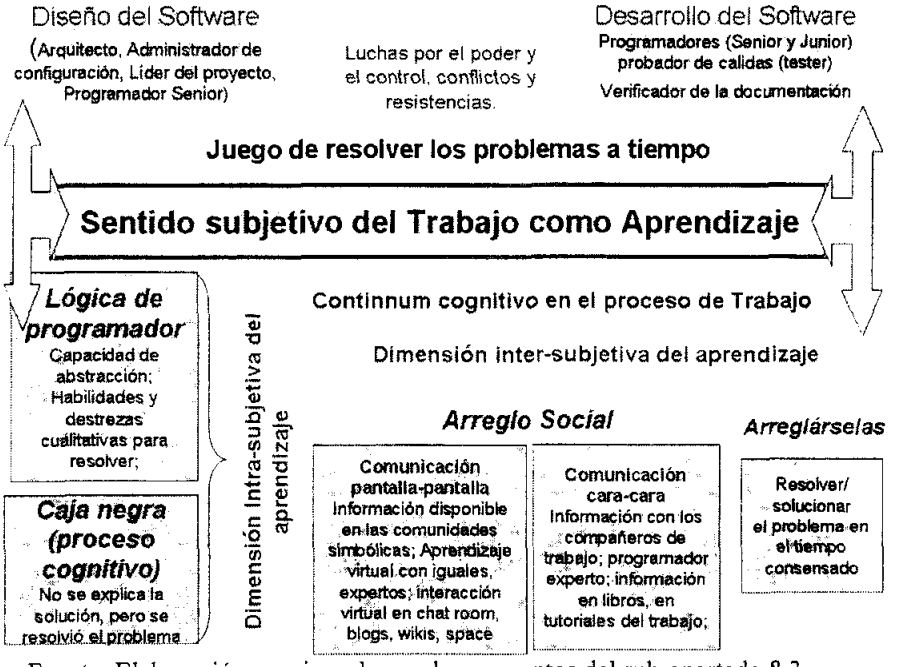
En el proceso de trabajo del software como trabajo cognitivo actúan una serie de *constelación de configuraciones subjetivas* que se mezclan en el proceso de trabajo, por ejemplo los intereses y representaciones, sentidos y emociones que guían al trabajador, es decir las intencionalidades que asume frente al trabajo. Por ejemplo, en las entrevistas con los programadores, señalaban que consideran la empresa como una ampliación del aprendizaje, donde si bien es cierto la empresa no les reconocía su iniciativa de autoaprendizaje, su labor fuera del espacio del trabajo, su creatividad en el trabajo, ya sea mediante estímulos económicos o de reconocimiento social, no importaba, porque la empresa les estaba dando la oportunidad de aprehender, de desarrollarse. En las entrevistas líderes y gerentes señalaban que los programadores estaban "adquiriendo" conocimiento y éstos una vez que *saben hacer*, que conocen las *prácticas del saber hacer*, renuncian y se emplean como freelance o bien buscan empleo en otras empresas. También señalan que una vez que los programadores *saben hacer*, se valorizan y negocian con la empresa otro tipo de prestaciones, o bien empiezan a disminuir el rendimiento, ya sea como boicot o resistencia a seguir laborando bajo las condiciones que consideran no apropiadas "se creen *primas donnas*".

Otro tipo de comentarios relativos a la representación del trabajo para los programadores tiene que ver con la resistencia y conflicto, unas veces implícito otras explícito. Por ejemplo, cuando el programador hace como que documenta el *saber hacer* de algún algoritmo complejo, solo escribe unas líneas para "cumplir", pero estos comentarios son inconclusos e ineficientes. Otras veces tiene que ver con el incumplimiento de tiempo y tareas asignadas, donde el programador lleva al limite el tiempo, cuando quizá pudo haber resuelto los comandos o tareas asignadas con anticipación. Este conflicto de llevar al limite el tiempo asignado para resolver -según las entrevistas- tiene que ver con la inconformidad, apatía hacia el trabajo, entre otros conflictos que se generan en el interior del proceso de trabajo, como otros que presionan desde el exterior.

9.2.- Configuraciones intrínsecas al sentido subjetivo del aprendizaje

Dos de las reglas informales de las que expresamos en el apartado anterior como "arreglárselas" y "arreglo social" se construyen bajo un amplio contexto de *dinámicas subjetivas* individuales y la suma o agregación de las acciones de los agentes que intervienen directa e indirectamente, virtual o concretamente en los espacios laborales conforman un enredado de interacciones que hemos denominado como *constelaciones de configuraciones subjetivas que intervienen en el proceso de trabajo*, las cuales podríamos señalar que están presentes en mayor o menor medida, con diferentes grados y matices, con claros-obscuros entre actores, llámense gerentes, administradores del sistema informático, programadores, tester, etc., que se enfrentan, colaboran, interactúan en el espacio laboral como fuera de éste, en un contexto de *fluidez cognitiva*, no en el sentido que señala Tripier, que los individuos que participan se complementan y colaboran en el proceso de trabajo (Tripier:1998 y 2003), sino una *fluidez cognitiva* que esta contenida por consensos y negociaciones informales, por conflictos y resistencias que no están ausentes de arreglos y consentimientos.

Esquema 12
Constelación de configuraciones subjetivas
que intervienen en el proceso de trabajo



Fuente: Elaboración propia en base a los conceptos del sub-apartado 8.3

Por ejemplo, la presión del *resolver a tiempo* y el *saber resolver* las tareas implica una serie de *arreglos sociales* y *arreglárselas* para proveer una solución algorítmica, solución

cognitiva que embebe habilidades y destrezas que denominamos *lógica de programador* que representa no solo acumulación del *saber hacer* sino una posición de *poder* en el proceso de trabajo, este contexto esta cohesionado tanto por el *sentido subjetivo del trabajo como aprendizaje,* por las *configuraciones subjetivas* descritas, así como por una *sumisión voluntaria* (consentimiento) de los agentes que intervienen en el proceso de trabajo (ver esquema 12). Sumisión más no alineación que le confiere a los agentes una definición de "jugador" individual en el proceso de trabajo, individualización que le significa un grado subjetivo de autonomía en el proceso de trabajo, independencia que se ve reflejada en la toma de decisiones individuales para seleccionar una ruta cognitiva, trayecto cognoscitivo que no necesariamente es la mas eficiente, pero en caso de resolver el problema planteado dentro de los limites del tiempo establecido y cumplimentar con los estándares y procedimientos de la empresa y las reglas informales de *"saber resolver en el tiempo consensado"* le confieren al desarrollador autonomía en el juego y a su vez una satisfacción de "control" sobre el proceso de trabajo.

> "Normalmente son entregables, por objetivos. Si a mi me dicen «hay que entregar esto mañana a las seis de la tarde» y yo sé que hoy y mañana no lo voy a acabar con mi horario normal, pues me voy más tarde hasta avanzarle lo mayor posible y, al otro día llego temprano y se entrega...no nos pagan hora extras. Como se cobra realmente por objetivos, nosotros no tenemos un horario" (programador GG3 MIXE).

El programador senior MZ3, de la empresa TADI, comentaba que el cumplimiento de la jornada de trabajo no es rígida, que él podía resolver los compromisos en su hogar o en otro espacio laboral, que no necesariamente es en la oficina, ya que el compromiso adquirido es que "tienes que cumplir para que se acabe el proyecto" (programador MZ3, TADI), así mismo señalaba que no se vigilan los progresos de los programadores, sino que donde hay un programador hay un proyecto, es decir "Aquí la observación es «se tiene que sacar el proyecto, para poder cobrar»...." (programador MZ3, TADI). En otras palabras coexiste un sistema de *prácticas implícitas* como el *arreglo social* de *resolver a tiempo*, al ser una microempresa se asigna a un programador por proyecto, no se vigila, se consensa que se tiene que terminar el proyecto en tiempo y forma, para contar con recursos y cubrir la nomina. Ahora bien, no es que la *sumisión voluntaria* signifique que sea pacifica, sin contradicciones, al estilo de Burawoy o de cooperación entre iguales al estilo de Tripier; por el contrario consideramos que existen otras prácticas informales que tienen que ver con la posición

intrasubjetiva del programador frente al trabajo, que hemos identificado: *lógica de programador* y *caja negra-cognitiva* ésta última está subsumida en la primera, son indisolubles, ambas se erigen como un conflicto soterrado, como una resistencia conflictiva, un boicot permanente, invisible, pero presente, que se sucede entre tanto entre programadores como programadores frente a empleadores. A continuación señalemos como se construye socialmente éstas dos prácticas informales en el contexto de *la constelación de configuraciones subjetivas* que intervienen en el proceso de trabajo cognitivo.

9.2.1.- dinámicas subjetivas: Lógica de programador

El *saber resolver*, implica a nivel cognoscitivo solucionar bajo cierto método que esta embebido de experiencias, habilidades, destrezas individuales (intra-subjetividad), conjunto de cualidades que se requieren para intervenir en el juego de *saber resolver a tiempo*, también se integra por cualidades externas de interacción con otros individuos (inter-subjetividad). Ambos espacios subjetivos en el individuo se yuxtaponen, están imbricados en la subjetividad del programador cuando resuelve un problema, dimensión subjetiva que hemos denominado *lógica de programador*. Para el líder de proyecto JM2, la *lógica de programador* es importante en el desarrollo de un sistema, en el entendido que no se resuelvan los problemas dos veces, sino que se lleve a cabo un solo proceso (ideal del manual de *buenas prácticas*), no importa que ello signifique tardarse mas tiempo del estimado, lo cual a su vez implica una negociación con el programador, que documente o resuelva de la mejor manera:

> "...la capacidad de abstracción, realmente eso es lo importante. Y también, que se note y se vea en cuanto a la eficiencia, pero te voy a decir en qué sentido... que no le guste resolver el mismo problema dos veces, sino que realmente cuando arme o plantee una solución de desarrollo, realmente sirva para futuro, vamos, que ya nunca lo vuelva hacer, porque muchas veces al desarrollar, desarrollas para el momento... «que flojera estar haciendo esto todo el tiempo, mejor lo declaro aquí en variables, que quede esto y cuando lo vuelva a utilizar, lo saco y lo uso y se acabó» (documentar el proceso) me tardé mucho más, porque muchas veces puedes resolver el problema en cinco minutos, pero se te vuelve a presentar y otros cinco minutos, y otra vez se te vuelve a presentar y otros cinco minutos, y a la larga te puedes estar meses en la sumatoria. Aquí en ese mismo problema, tal vez no le dedicaste cinco minutos, le dedicaste una hora, pero jamás le vuelves invertir nuevamente...Entonces de esa manera de practicidad o eficiencia es lo que estaríamos buscando como cultura o forma de pensar realmente" (Líder JM2 DYYA)

Esta cita nos permite observar que el líder de desarrollo JM2 aboga por una cultura del tipo ideal "resolver para siempre", olvidándose de los intereses, sentidos y representaciones del programador frente a la pantalla, así como de las contingencias implícitas al trabajo cognitivo; el manual de las "buenas prácticas" señala un *tipo ideal* en programación, con un trabajador que documente el proceso, explique *como* desarrollo los algoritmos, que describa la intencionalidad del *texto sígnico* (conjunto de símbolos compuesto por signos) que representa una serie de rutinas que resuelve el problema planteado en el diseño, la explicación del *porque y el para que* de éstos textos sígnicos permitirá, en el futuro, proveer de mantenimientos y actualización al texto sígnico; sin embargo las "buenas prácticas" relegan los intereses e intencionalidades del programador. Al respecto, el líder de programación RS2 y dueño de la empresa TADI, señala que la *lógica de programador* no es evidente, esta embebida entre las líneas del código que comprende un algoritmo, abstracción que no se ejercita en la escuela, sino a través de la dinámica laboral (Líder RS2 TADI-MT2); la *lógica de programador* no se circunscribe a describir el *saber hacer* a través de la documentación del código o comentarios alusivos al *texto sígnico*, sino que esta embebido en las representaciones e intencionalidades del mismo, están implícitas tácitamente en los pensamientos y abstracciones que implica el *saber resolver*:

> "...el director de operaciones del ITESM, me dice «...sabes que desarróllate la secuencia Fibonaci», y entonces desarrolle el código y me la avente en cuatro o cinco líneas... -y dijo- «pues... esta bien»... y ya después platicando le comento «porque me pones a...» -me contesta- «se puede hacer en dos línea» -le contesto- «no inventes» -dice- «...sí...» y, que se pone a hacerla en dos líneas. Entonces ahí es donde te digo, que mucho tiene que ver la lógica del programador o sea lo que puedes hacer a lo mejor en cincuenta líneas, alguien lo hace en cinco, mucho tiene que ver los tiempos..." (Líder RS2 TADI).

El concepto *"lógica de programador"* representa una serie de *dinámicas subjetivas* como son las habilidades y destrezas cualitativas para *saber resolver* un problema en el tiempo que se consenso; sin embargo, este proceso está constreñido por un contexto de incertidumbre, ya que dicha solución no se está seguro de que sea eficiente u óptima, ya que no esta exenta de fallas en el sistema (bugs); problemas de interoperabilidad con otros módulos, etc., (contingencias de la *aflicción del software*), este contexto es una presión cognitiva, porque forma parte del conjunto de incertidumbres presentes al proceso de trabajo. Al respecto el gerente y líder DC2 de la empresa CATI, señalaba que no considerar las

incertidumbres de un proyecto, puede significar la quiebra de áreas de trabajo, porque los constantes retrasos e incumplimientos se traducen en crisis. DC2, señalaba que los programadores no sólo no documentan los procesos, sino que están mas interesados en "resolver retos propios" y, en caso de no existir nuevos retos renunciaban a la empresa (Líder DC2 CATI), este *sentido hacia el trabajo como aprendizaje* implica un conjunto de *satisfacciones relativas*, que de no ser percibidas por los programadores en algunos casos significa rotación laboral o bien una disminución de la *interacción creativa* en el *juego de resolver a tiempo*. DC2, comenta que "...no todo se puede interpretar al detalle...cuando tienes la idea del código..." este comentario hace referencia al hecho que no todo texto sígnico se puede describir mediante documentación, existen pedazos de código que simplemente funcionan, pero son una "complejidad" comprender el *como* operan, donde dicha operabilidad se convierte en una especie de *"caja negra-cognitiva"* en la cual, incluso el mismo programador no se atreve a re-codificar o documentar. Un ejemplo nos lo señala el programador SA3:

> "...sí se puede leer (el código) para que lo documento, si alguien sabe programar va a saber lo que dice aquí y corregirlo...sin embargo, hay esos pedacitos, hay esos fragmentitos de 25 ó 50 líneas (de código) que muchas personas le tienen miedo, porque son pocos lo que se atreven a meterle mano y sin embargo, puede llegar una persona con suficiente capacidad intelectual y decir «... en este fragmentito de 25 líneas está el problema que les esta dando lata en los últimos 2 años, lo voy a reescribir...», igual a veces lo reescribe mas limpio, a veces mas enredado, pero con la corrección..." (Programador SA3 DINS).

Consideramos que el proceso de trabajo la *reflexividad cognitiva* junto con las dinámicas de *interacciones (simbólicas, sígnicas y gráficas)* están ceñidas a un contexto que podríamos denominar como *tiempos y pensamientos* no sujetos a *normas disciplinares* o técnicas de las *herramientas procedimentales* que generen un sistema de *prácticas estándar*, suponemos que la *reflexividad cognitiva* implica *dinámicas subjetivas*, como el *sentido común*, *satisfacciones relativas*, intereses y sentidos del programador frente al trabajo y los otros que le permite *saber resolver* problemas, que junto con las *destrezas y habilidades acumuladas* para aprehender -ya sea un nuevo lenguaje, una nueva herramienta lo que permite desarrollar una *lógica de programador*. En cambio el líder JG2 de la empresa MIXE señala que el programador debe tener habilidades para comunicarse con los gerentes y con el

cliente para señalar *que* es lo que esta haciendo y *como* lo esta haciendo (Líder JG2 WALS-MT3). Otro líder JA2 de la empresa MIXE, va más allá en términos de la *lógica del programador*, al señalar que ya existen *herramientas procedimentales* como UML, PMI, RUB, entre otros métodos que le dan sentido al *camino* que debe seguir el programador cuando desarrolla el código o componentes del programa y en caso que el programador no tenga esta *lógica de programador* es mejor contratarlo en otra área del proceso de trabajo como es en administración, en documentación, pero no en desarrollo, ya que "si no tiene la *lógica de programación* nunca se le va a desarrollar, cuesta muchísimo trabajo, para que le sigues haciendo creer y que desperdicie su tiempo en estar haciendo cosas que él no tiene habilidad" (Líder JA2 MIXE).

9.2.2.- Barrera cognitiva en el *saber hacer*: la caja negra

No hay una solo camino para saber resolver, no existe un único procedimiento para conformar los algoritmos estas diversas rutas posibles configuran la *lógica de programador* y distinguir la reflexión por otro programador no es sencillo, porque esta sujeta a una serie de dinámicas subjetivas que hemos denominado *caja negra-cognitiva* en el entendido que este concepto no es explícito sino que está embebido en el desarrollo de las líneas de código y encierra una serie de acciones individuales no visibles, como el ocultamiento de información, el *saber-hacer* autolimitado intencionadamente, etc,. Gerentes y programadores reconocen que existen "pedazos" de código que no están documentados eficientemente, incluso que no se mencionan en la documentación (código ciego, ocultar código, etc.); al respecto el gerente y líder RS0 de la empresa TADI, señala que el resolver un problema sin que el programador pueda explicar el *como* esta elaborado el algoritmo se debe a "un conjunto de habilidades y experiencias...no es más que la forma de resolver problemas de la manera más sencilla" (Líder/gerente RS0 TADI-MT2), pero no se puede describir el como se resolvió. La regla informal de *caja negra-cognitiva* esta embebida en el *saber hacer*, es parte del *poder* como saber, no se percibe sino que esta invisibilizada en el proceso de trabajo:

> "...ese buen programador por mas explicito que pueda ser, si su lógica va mas allá de la lógica dé los demás programadores, jamás le van a entender... tu programas todos los días, vez algoritmos todos los días, lo tienes en tu mente, lo tratas de explicar y... no es que seas egoísta, lo que pasa es que no te van a entender, porque usas algoritmos tuyos usas formas de pensar tuyas... por aquí la lógica es otro" (Programador FA3 ENEX).

El programador FA3, comenta que en la clase de inteligencia artificial, desarrollaban algoritmos genéticos de 200 líneas de código y un compañero de clase sabia resolver el problema en 10 líneas de código, "por ejemplo el problema de las 8 reinas que tienes en el tablero del ajedrez, que son millones de combinaciones para que 8 reinas no se ataquen, entonces haces un algoritmo genético para resolver ese problema en milésimas de segundos, para que saque todas las combinaciones posibles, nosotros la sacábamos en 7, 8 segundos y él, las sacaba en menos de 1 segundo, entonces...te quedas «oye, aja», y por mas coco..." (programador FA3 ENEX).

Consideramos que la *lógica del programador* es intrínseca a un proceso de dinámicas subjetivas que denominamos *acciones individuales intra-subjetivas e inter-subjetivas* que están penetradas por un *saber hacer* tácito al proceso de abstracción y solución que no es del todo interpretativa, sino que está embebida en el *saber-hacer como poder* como una especie de *"caja negra" del saber-hacer como poder del programador*; subjetividad que bien podemos señalar como una *barrera cognitiva* superpuesta entre las relaciones de saber como poder entre las interacciones cotidianas en el proceso de trabajo; entre los actores que intervienen en el proceso de trabajo aún existiendo una aparente *fluidez cognitiva* entre los que diseñan el software y los que desarrollan al código, *barrera cognitiva* que no necesariamente implica restricciones a la subjetividad, por el contrario, ésta se erige como un complejo de dinámicas sociales definidas al interior de las *constelaciones de configuraciones subjetivas* que intervienen en el proceso de trabajo, este conglomerado subjetivo de sentidos y representaciones del programador frente al trabajo es construido como un *sentido de aprendizaje* de los programadores frente a la pantalla del trabajo; otro ejemplo es la disposición del programador a laborar fuera de los espacios de la jornada de trabajo, sea como autoaprendizaje, investigación personal del *saber resolver* una tarea pendiente, informarse que portales de información virtual ofrecen cursos o tutoriales de interés personal, etc.

La barrera cognitiva representa también una serie de conflictos, resistencias, boicot, que se extiende no solo en los espacios laborales, sino que comprende también los espacios de vida y recreación del programador. Barrera que comprende conflictos institucionalizados a nivel macro, como es la implementación de *herramientas procedimentales* y *normas disciplinarias* que implican una serie de conflictos abiertos y dirigidos. Conflictos que consideramos como potenciales para la conformación social de una ocupación profesional y

estructuración de comunidades simbólicas de trabajadores que a su vez son constreñidas por una primigenia división social de trabajo (Diseño/requisitos- Desarrollo/Calidad-Implementación); división que consideramos es promovida por las ingenierías citadas a través implementar y proponer como salida a la crisis "el mejor camino" de las *herramientas procedimentales* y *normas disciplinarias* que influyen en un conjunto de métodos y procesos *(que hacer)*; sin embargo, como hemos dado cuenta la reflexividad cognitiva del *texto sígnico* representa una acción creativa del programador que se traduce como *saber-hacer y saber como poder* en el proceso de trabajo.

9.3.- Abriendo la caja negra: El conflicto atrás del sentido subjetivo del aprendizaje

Desde la teoría constructivista vygotskiana se habla de internalización del aprendizaje como el proceso de apropiación de los instrumentos psicológicos por parte del individuo-aprendiz, gracias a la *interacción social* frente a los otros más diestros y hábiles. Vigotsky niega que la actividad externa donde se interacciona con los otros y frente a los otros y la actividad interna sean idénticas, pero afirma aún así su interconexión. Por tanto el proceso de internalización del aprendizaje exige una reelaboración por parte del sujeto aprendiz. No hay que olvidar que este concepto no esta del todo acabado, como bien señala Rugoff (1990) que "el concepto de internalización se ha usado en diferentes planteamientos teóricos, desde la interiorización de la acción sobre los objetos (Piaget), hasta la interiorización de la actividad social (Vigotsky) y la interiorización del modelo, en la teoría del aprendizaje social (Aronfreed, 1968; Bandura. 1986; Zinchenko, 1985). Sin embargo, para el caso que nos interesa, que es el proceso de transferencia de la actividad "externa" al plano "interno" ha recibido una insuficiente especificación en la mayoría de las perspectivas, que han abordado este término (Rogoff, 1993)[112]. Partiremos de Vigotsky (1995) para quienes cualquier aprendizaje implica el entendimiento e internalización de los *símbolos y signos* de la cultura y grupo social al que pertenece el (los) individuo (s). Para Vigotsky, los aprendices o alumnos se apropian, embeben, interiorizan las prácticas y procedimientos culturales a través de la interacción con miembros más diestros, con experiencias y habilidades diferentes, proponiendo para explicar este proceso la *Ley genética general del desarrollo cultural*, donde:

[112] Roggof B (1993) Aprendices del pensamiento. El desarrollo cognitivo en el contexto social, paidos, Barcelona.

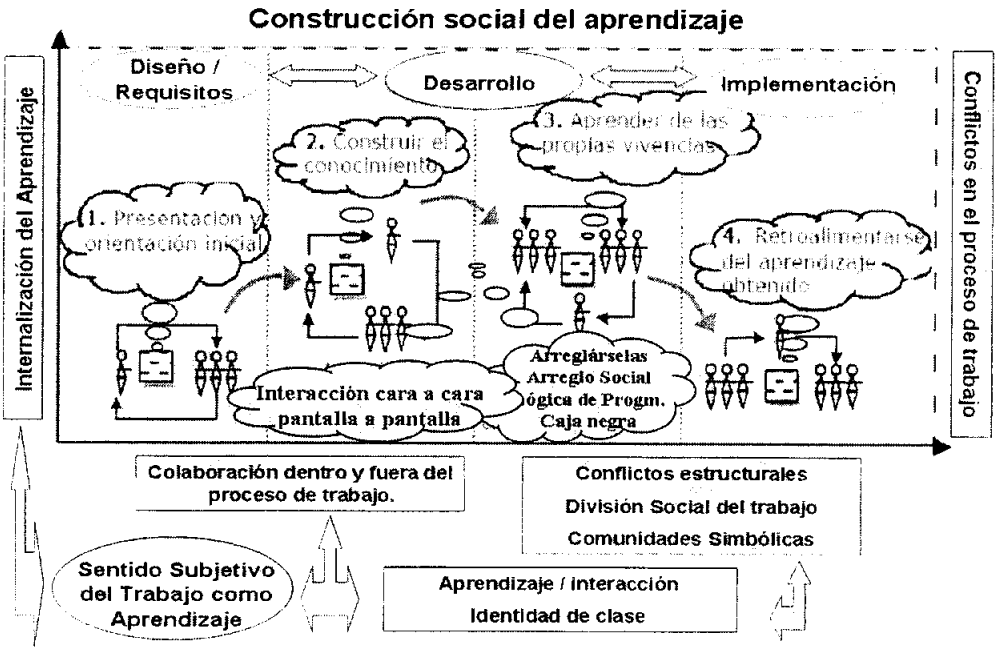
- Considera como insuficiente el concepto de Aprendizaje Asociativo ya que considera que existen rasgos específicamente humanos. No reducibles a asociaciones tales como la conciencia y el lenguaje.
- El conocimiento no es un objeto que se pasa de uno a otro, es algo que se construye por medio de operaciones y habilidades cognoscitivas que se inducen en la interacción social.
- El desarrollo intelectual del individuo no puede entenderse como independiente del medio social en el que esta inmerso el individuo.
- El desarrollo de las funciones psicológicas superiores se dan primero en el plano social y después en el nivel individual.
- La transmisión y adquisición del conocimiento y patrones culturales es posible cuando la interacción (plano inter-psicológico) se llega a la internalización (plano intra-psicológico).
- La internalización, es el proceso de pasar lo inter-personal (inter-subjetivo) a lo intra-personal (intra-subjetivo).

Vigotsky insiste en el origen social de las prácticas, señalamiento que afirma el hecho que los procesos psicológicos se forman en y atraviesan por una fase social que proviene de la actividad que establece el sujeto con los objetos y en contacto con otros individuos (interacción). La tesis de Vygostsky establece que el conocimiento es una construcción socio-histórica, ésta tesis da lugar a formular las bases de la "ley genética del desarrollo cultural", en la cual Vigotsky señala que en el desarrollo cultural del aprendiz o el alumno, toda función aparece dos veces: primero, en el ámbito social y más tarde, en el ámbito individual; primero entre personas (inter-subjetivo) y después en el interior del propio aprendiz (intra-subjetivo).

Para Vigotsky, las funciones superiores como la reflexión, la atención, el pensamiento, etc. se originan como *interacciones sociales* (Vigotsky, 1979:94), en este sentido consideramos que el conjunto de *configuraciones subjetivas* que hemos expuesto y denominado como: *arreglárselas, arreglo social, lógica de programador y caja negra cognitiva* se constituyen a partir de una serie de consensos, negociaciones e interacciones sociales, sean individuales, colectivas, en comunidad, virtuales, etc. Es decir, en principio las interacciones sociales que llevan a cabo las personas como una categoría inter-psicológica, después son interiorizadas por el individuo y aparecen como aprendizaje, como categoría intra-psicológica. Vigotsky, considera que las funciones psicológicas superiores (reflexión, atención, pensamiento, etc.) son relaciones sociales internalizadas y, la internalización es un proceso de *interacción creativa* donde ciertos aspectos de la interacción social de la actividad donde se desarrolla el individuo ó aprendiz (plano externo) pasan a interiorizarse, a gestarse en

el plano interno del sujeto; este proceso de *interacción creativa* está basado principalmente en la comunicación y la reflexividad cognitiva, y supone inter-subjetividad. Para Vigotsky, la intersubjetividad proporciona el fundamento de la comunicación y, la comunicación de una experiencia o una idea requiere relacionarla con un tipo de fenómeno ya conocido, es decir generalizar el fenómeno para comunicarlo, y la comunicación a su vez supone inter-subjetividad (Bruner:1991 en Coll:1999)[113]. El párrafo anterior nos permite fundamentar el concepto que se construyo el del juego de *saber resolver* a tiempo, entendido como la *interacción creativa* entre los que intervienen en el proceso de trabajo para el desarrollo de un programa informático, se conduce éste en el plano inter-subjetivo, es decir entre individuos y, en el plano intra-subjetivo que acontece cuando el programador internaliza el problema, se apropia de la problemática a resolver, aprehende e interioriza el problema a resolver para solucionar a tiempo. Ambos planos de la subjetividad permite proponer el concepto de que *saber resolver* implica, paralelamente una posición de poder por parte de quien resuelve, estos es el *saber hacer como poder* (ver esquema 13).

Esquema 13



Fuente: Elaboración propia en base a las entrevistas

[113] Bruner, J. S. (1991). Actos de significado. Más allá de la revolución cognitiva. Madrid: Alianza. 2. Coll, C., y otros. (1999). El constructivismo en el aula, Barcelona, Editorial, Graó.

Para Vigotsky (1988 y 1995)[114] el aprendizaje es un proceso complejo de construcción social, en el cual convergen conocimientos, cultura de los individuos, intenciones, sentimientos, etc. Para Vigotsky, la unidad básica no es el individuo en singular, tampoco los procesos cognitivos o de aprendizaje del o los individuos fuera del contexto que da lugar a dichos procesos; por el contrario es la acción reciproca, es el conjunto de acciones-interacciones y dinámicas entre las actividades de los individuos que acontece en contextos y coyunturas determinadas (Vigotsky:1979; Wertsch :1988, 1993; Cole:1983); según dimos cuenta en el proceso de trabajo convergen una serie de *configuraciones subjetivas del aprendizaje*, mediado por conflictos y resistencias del saber hacer y el saber como poder. Es decir, el proceso de trabajo del software esta inmerso en una serie de configuraciones flexibles y rígidas, lo que nos llevaría a señalar una contexto subjetivo de aprendizaje como expresión de las relaciones sociales de producción, donde hasta el momento, se han identificado una serie de estructuras, acciones y subjetividades que moldean y constriñen las acciones concretas de los individuos, ya sea en forma de consensos y arreglos como de resistencias y conflictos, que se han abordado a través de conceptos como los siguientes: los conflictos de *resistencia a documentar el código*, entendida como una expresión de conflicto abierto; la *ocultación de código*, es una práctica social conocida también como "código ciego", inmersas ambas en un *conflicto no dirigido*. A continuación abordaremos otros tipo de conflicto que, consideramos son endógenos al proceso de trabajo, por ejemplo, el conflicto que se suscita al resolver parte del trabajo fuera del espacio laboral, es decir en los tiempos de recreación del programador; así como la dinámica laboral que denominamos *interacción virtual* pantalla a pantalla, dentro o fuera del proceso de trabajo; conflictos éstos que, consideramos como integrantes de un *conflicto institucionalizado*, en el entendido que éste tipo de conflicto hace referencia a aquellas acciones conflictivas que podrían estar o no reconocidas formalmente, se conducen como norma de costumbre o se reconocen como una práctica social embebida en la cultura pero que no haya alcanzado el estatus de norma.

Otro conflicto que se configura es la yuxtaposición de los espacios de vida y del trabajo mediados por el ámbito simbólico del programador. Ámbito que no es nuevo, ya Thompson señalaba que en el siglo XX los trabajadores de la Industria automotriz se reunían

[114] Vigotsky, L. (1988). El desarrollo de los procesos psicológicos superiores. México: Editorial Crítica, Grupo editorial Grijalbo.; Vigotsky, L. (1995). Pensamiento y lenguaje. Buenos Aires: Ediciones Fausto.

en los bares para intercambiar saberes relativos al oficio. Donde la "caja de herramientas" representaba un "indicador" de las habilidades y destrezas del artesano automotriz; utilizaremos esta metáfora para señalar que los programadores de software, acuden a *comunidades simbólicas virtualizadas* como foros, chat-room, blogs, wikis u otros espacios en Internet para *interaccionar virtualmente*, prácticas interactivas que no necesariamente es en tiempo real, sin embargo una forma de "evaluar" la "caja cognitiva" del programador, es el conjunto de destrezas y habilidades con respecto al conocimiento de lenguajes de programación, plataformas tecnológicas, numero de proyectos en los que colabora, participación activa en las *comunidades simbólicas virtualizadas*, "identidad" geek en la comunidad simbólica que pertenece, etc.

El conjunto de *dinámicas subjetivas* que intervienen en el proceso de trabajo como los arreglos, juegos, consensos y negociaciones, hemos obviado explicar la praxis de dichas subjetividades: ¿Que representaciones y significaciones posee los programadores frente a trabajo?; ¿Que sentidos e imágenes configuran las *dinámicas subjetivas* que intervienen en el proceso de trabajo?. Para abordar estas preguntas, retomaremos las propuestas teóricas De la Garza (1992, varios años) para explicar las relaciones entre estructuras, subjetividad y acción. A continuación examinaremos que está atrás de la *constelación de configuraciones subjetivas* que se traduce en acciones individuales y colectivas que involucra interacciones sociales y consentimientos del tipo *arreglárselas* y *arreglo social*; *lógica de programador* y *caja negra-cognitiva*. También se abordará que significados están embebidos en el *sentido subjetivo del trabajo como aprendizaje*.

Consideramos la existencia de *configuraciones subjetivas de poder*, que se expresa en heterogéneas figuraciones de resistencias, conflictos, boicot que están atrás del *sentido del trabajo como aprendizaje*. Los conflictos no están separados de los consensos, no emergen contrarios a la *sumisión voluntaria* de la *negociación* o el *arreglo social*, no están separadas del saber hacer y del saber como poder, no son contrarias al *saber hacer*, consideramos que son paralelas e integradas al proceso de trabajo, son provocadas mas no definidas por presiones estructurales de una serie de normas, sentimientos y valores que se rejerarquizan en el proceso de trabajo (De la Garza, varios años).

Las consideraciones anteriores de las relaciones entre estructura-subjetividad-acción. nos permitirá responder algunas preguntas. Los conceptos construidos a partir de la *sumisión*

voluntaria (Burawoy) por el individuo para constituirse como jugador individual, es parecido al término de Foucault, cuando dice que el poder se asume como un juego entre los individuos y una vez que culminan las relaciones de poder, culmina el juego. Ambos autores, Burawoy y Foucault, esquivan el hecho que en todo juego están en juego una serie de intereses, motivaciones y valores; que en todo juego el resultado no es "suma cero"; que en el juego se evaden algunas leyes reconfiguran procesos informales de conflicto y resistencias, que no está ausente en el saber hacer las colisiones y confrontaciones que forman parte del juego de *arreglárselas, arreglo social,* etc.

9.4.- Conflictos y resistencias en el proceso de trabajo del software a la medida

Las particularidades del proceso de trabajo con respecto al *sentido subjetivo del trabajo como aprendizaje*, van más allá de la perspectiva restringida de *sumisión voluntaria*, por el contrario consideramos que las acciones individuales y colectivas, como las de *arreglárselas* para *saber resolver a tiempo*, así como el *arreglo social* entre los agentes que intervienen en el proceso de trabajo, están implícitas y dan forma a las *dinámicas subjetivas* individuales como: *lógica de programador* y *caja negra cognitiva*, acciones que permiten dar cuenta de una serie de configuraciones subjetivas individuales y colectivas en el contexto del *saber resolver* y, del *saber como poder*. Configuraciones que a su vez comprenden una serie de conflictos abiertos y dirigidos: conflictos soterrados y ocultos; conflictos institucionalizados; conflictos discretos, pero continuos, constituidos como normas informales en la acción social del consenso y la negociación. Para construir dichos conflictos, partimos de las teorías de configuración de De la Garza (Varios años); las propuestas de Edwards y Scullion ([1982], 1987) y Baethge y Oberbeck ([1986], 1995); construimos los conceptos de conflicto que se configuran en el proceso de trabajo[115] (ver cuadro 25).

Consideramos que éstos conflictos y resistencias están contenidos y constreñidos por estructuras como la cultura, las políticas educativas y laborales, así como por el conjunto de *herramientas procedimentales* y *normas disciplinarias* que se erigen como respuestas y "únicos caminos" posibles para *saber resolver*; también por las *constelaciones de*

[115] Edwards P.K. y Hugo Scullion ([1982], 1987) La organización social del conflicto laboral. Control y resistencia en la fábrica.MTSS, Madrid España; y Baethge Martín y Herbert Oberbeck ([1986], 1995 El futuro de los empleado, Nuevas tecnologías y perspectivas profesionales en la gerencia empresarial. MITSS, Madrid, España.

configuraciones subjetivas que se conforman en las *dinámicas sociales* (arreglos social, arreglárselas, lógica de programador y caja negra cognitiva). Las *normas disciplinarias* junto con las *herramientas procedimentales* se constituyen como "el único mejor camino" para atenuar la "aflicción del software", imponiendo métricas de calidad, metodologías en el desarrollo, estándares en el proceso de trabajo, que terminan por gestionar una posible división entre diseño y desarrollo del código; las respuestas de los que intervienen en el proceso de trabajo cognitivo, es una serie de resistencias y conflictos, entendidos como expresión de réplica, de contra-respuesta, hacia los métodos de las ingenierías del software, contradicción que se erige como una reacción paralela, sincrónica y coexistente a las "sugerencias" de las ingenierías del software que se constituyen como estructuras y métodos verticales y burocráticos, promovidos por y para las grandes empresas de software, no para las microempresa que desarrollan software a la medida y que poseen equipos de programadores pequeños. La reacción las empresas pequeñas podemos nombrarlo como un *modelo de producción fluido.*

Cuadro 25

Constelación de configuraciones subjetivas (prácticas y dinámicas) como soporte virtual del sentido subjetivo del trabajo como aprendizaje en el proceso de trabajo del software a la medida

1.- *Conflicto abierto de la resistencia*: hacemos referencia al hecho de que el conflicto es conocido por los que intervienen en el proceso de trabajo y tiene lugar una acción que expresa el conflicto.
Documentación insuficiente del código: Esta acción de los programadores está suficientemente ilustrada por la Ingeniería del Software, en el proceso mismo del trabajo. Es una lucha presente entre diseñadores y desarrolladores. Por el saber hacer. La administración genera e implementa una serie de métodos para verificar la documentación (Programación orientada a objetos, lenguajes orientados como Java, herramientas como UML, RUP, etc. etc. Los programadores documentan lo menos posible, utilizan nombres de variables que sólo ellos conocen el significado, etc. Mientras que la administración busca un "código limpio" los programadores responden con "código sucio". Mientras que la administración aconseja un código documentado, el programador responde con un código insuficientemente documentado.
2.- *Conflicto no dirigido de la resistencia*: Aquí consideramos que coexiste un comportamiento persistente, pero no del todo abierto, las acciones no son abiertamente conflictivas.
Ocultación de código. Está acción de los programadores, es soterrada, implícita en el proceso de saber resolver; subyacente en la *lógica del programador* en esa *caja negra*-cognitiva que resuelve el problema, pero no "sabemos como opera". La administración implementa metodologías de calidad o revisión del proceso, sin embargo el programador responde con "código ciego", con un boicot no declarado, implícito en la "lógica" de los textos sígnicos. Se resiste a declarar el *saber-hacer*.
3.- *Conflicto institucionalizado*: Hacemos referencia a aquellas acciones de conflicto que podrían estar o no reconocidas formalmente, actua como norma de costumbre o reconocerse como una práctica aceptada tradicionalmente, pero no haber alcanzado el estatus de norma.

Interacción pantalla-pantalla en el lugar de trabajo. Para los programadores, acudir a los Chat-room, blogs o wikis entre otras *comunidades simbólicas de programadores* forma parte cotidiana del proceso de *saber resolver* un problema. También es resultado de arreglos sociales informales.

Trabajar fuera de la jornada de trabajo. Es una práctica común entre los programadores que parte del trabajo lo solucionen fuera del proceso de trabajo; para los administradores y programadores solucionar parte del trabajo cognitivo en otros espacios fuera del trabajo es común.

4.- *Conflicto implícito:* Hacemos referencia a aquel conflicto que no se expresa en una acción concreta, pero esta presente en la estructura de la situación del conflicto.

Yuxtaposición de espacio productivo y espacio de vida. Para los programadores, cuando no se resuelve parte del problema, cuando se "cicla" el conocimiento, cuando se dificulta la solución, se este agotando o no el tiempo concedido o negociado; o bien cuando el programador se empeña o se obstina (dependiendo del grado de internalización del problema, identidad con la empresa, ética de hacker, etc.) cobra relevancia las configuraciones subjetivas del consentimiento como el *arreglárselas* y *el arreglo social* para saber resolver parte del problema dentro o fuera de la jornada de trabajo.

5.- *Conflicto latente del poder*: Nos apoyamos aquí en parte en la definición de *conflicto latente* que hace Dahrendof (1959) citado por Edwards y Scullion (1982], 1987:29-30) quien señala que hay conflictos que pueden no mostrarse en acciones visibles cuando el *poder* de una parte es tan grande que suprime las formas concretas de protesta; o bien, si existen medios institucionales como normas, prácticas o procesos formales de blandir el conflicto de tal manera que no se traduzca en acciones perceptibles. *Primigenia división social del trabajo:* Es cada vez mas latente la presión estructural de las "nuevas" ingenierías en la transformación del proceso de trabajo en la industria del Software. Del proceso artesanal del software de la década de los sesenta a la intencionalidad de un proceso sistemático moderno del software, donde el mejor ejemplo son las fábricas de software. Entre estos dos etapas del software median una serie de transformaciones estructurales de conflicto por el poder:

Partición del proceso de trabajo: De un Diseño y desarrollo centrado y concentrado en el Programador en la etapa artesanal, se ha divido en el Diseño/requisitos por un lado y el desarrollo/calidad por el otro. Lo cual no se contradice con el hecho de que el programador esta presente en ambas, como hemos señalado, debido a que estamos hablando del software a la medida para las empresas estudiadas. Sin embargo esta transformación estructural (que se aplica en las fábricas de software) está presente subjetivamente en el proceso de trabajo.

Herramientas procedimentales, normas disciplinares y el texto sígnico. Hacemos referencia al hecho que las Ingenierías informáticas (Ing. del Software; Ing. de Requisitos; Ing. de la usabilidad ó Ergonomía del Software entre otras) se constituyen frente al proceso de trabajo como *normas disciplinares* que han formalizado un conjunto de *herramientas procedimentales* (ya descritas) para desarrollar código bajo métodos como el de "buenas prácticas", bajo metodologías comprensibles para la gerencia. Consideramos que a la usanza de ford-taylor en la industria del software está cuajándose una primigenia división social del trabajo y una lucha por el saber hacer, un conflicto y resistencias por el saber-poder; donde por un lado las *herramientas procedimentales* implican normas y por otro lado el texto sígnico representa un conjunto de simbolos abstractos-cognitivos que constituyen el saber-hacer del programador.

6.- *Conflicto estructural explicito del poder*: Consideramos que coexisten presiónes estructurales que constriñen y dan forma a la acción individual y colectiva de los sujetos. Los conflictos, resistencias y boicot se suceden tanto a nivel endógeno como exógeno al proceso de trabajo. Advertimos que se configuran cuando menos, dos acciones concretas de resistencia, la primera a nivel del proceso de trabajo y otra a nivel de estructura.

El trabajo cognitivo fluido y ágil. La propuesta de las *normas disciplinares* (Ingenierías) se tornan burocráticas y verticales frente a la *fluidez cognitiva* que demanda el desarrollo de software. En el proceso de trabajo se configuran procesos fluidos, ágiles donde el *saber hacer* no es burocrático, sino horizontal, no sigue métodos estáticos, sino resoluciones tácitas. Es fluida porque todos los agentes paricipan embebidos en una serie de consensos y conflictos, entre areglos sociales y boicot. Entre documentar el proceso y simplemente saber resolver a tiempo.
Modo libre del Saber–Hacer (Software libre). Se erige en un conflicto estructural entre el modo de software privado yel modo de software de código libre. En el software de código libre se da prioridad a compartir el código, antes que las herramientas procedimentales, se prioriza la lógica del programador y desarrollar código entendible, auto-documentado. Saber resolver no está en la documentación del código, sino en el acceso al código para que otro programador lo haga mas transparente, mas limpio, "menos ciego". El conjunto de *herramientas procedimentales* en el software libre estan orientadas a expandir el saber como poder entre los programadores de software libre.

Fuente: Elaborado en base a las entrevistas y redefiniendo conceptos de Edwards P.K. y Hugo Scullion ([1982], 1987) La organización social del conflicto laboral. Control y resistencia en la fábrica. MTSS, Madrid España; y Baethge Martin y Herbert Oberbeck ([1986], 1995 El futuro de los empleado, Nuevas tecnologías y perspectivas profesionales en la gerencia empresarial MITSS. Madrid, España

Edwards y Scullion, Baethge y Oberbeck así como Burawoy, son consistentes en sus argumentos que el conflicto, la resistencia, el consentimiento, la negociación y los juegos significan el reconocerse entre si y diferenciarse de los otros, el reconocerse como grupo, como ocupación, es un proceso fundamental para orientar el conflicto o la negociación. Como explicamos en el capitulo 7, las empresas de software en el Valle de México son recientes, el mercado de trabajo es de reciente conformación, ocupación de programador está en construcción social, la profesión está en un proceso de consolidación; subsiste un periodo de reconocimiento social de la ocupación que está generando "manifestaciones concretas del conflicto con su entorno social" (Edwards y Scullion ([1982], 1987:28). Apoyándonos en los argumentos conceptuales de Thompson E. (1989) en su libro "la formación de la clase obrera en Inglaterra", quien señala que atrás de los conflictos y resistencias está prorrumpiendo, constituyéndose una "conciencia de clase" entendiendo como clase a los individuos con conciencia de si mismos. Para el caso de los programadores de software a la medida, la idea de Thompson nos permite proponer que los programadores a través de las *configuraciones subjetivas del aprendizaje, dinámicas subjetivas, sentido subjetivo del trabajo como aprendizaje*; las *comunidades simbólicas de programadores*, entre otras interacciones y practicas sociales y virtuales dan forma a un conflicto no explícito del *saber hacer* y el *saber como poder*; conflicto que se expresa en una serie de dinámicas heterogéneas como actitudes, conductas y procesos objetivos y subjetivos, concretos y virtuales; unas veces como el

software libre, las comunidades *open source*, la disponibilidad de información en Internet para iniciar un boicot en la programación, técnicas de ocultamiento de información, la decisión del programador de "ensuciar" el código, "enredar" código (código espagueti), etc.

El conflicto es paralelo al proceso de negociación, consenso, arreglos sociales, etc. El proceso de trabajo de los programadores de software, se formaliza no entorno a estructuras rígidas como sindicatos, sino a través de comunidades simbólicas de trabajadores, como las conformadas por Debian, Ubuntu, Microsoft, Oracle, etc, que organizan, y difunden el *saber hacer* a través de *flujos cognitivos virtuales* conformados en *comunidades simbólicas de trabajadores*; por ejemplo los blogs, wikis, debian, ubuntu, etc., entre otros espacios virtuales. Por ejemplo la ocupación del trabajador cognitivo en el *modelo de producción de código libre*, las actividades sociales y culturales que los conforman son los textos escritos en creative commons, término que aplica para cine libre en Internet (youtube); clases académicas de acceso libre; música de formato libre; etc.

Consideramos que los conflictos señalados en el cuadro 31 representan la punta del iceberg de una serie de luchas, resistencias y confrontaciones que se contextualizan junto con una necesidad de explicar la *construcción social de la ocupación*. Esto es, explicar cuales son las variables estructurales o simbólicas que dan cuenta de un concepto ampliado de trabajadores simbólicos; concepto que podría estarse configurando como una nueva ocupación de trabajadores.. Necesitamos construir un concepto que vaya mas allá de concepto de "trabajadores del conocimiento" o "trabajadores de la sociedad del conocimiento" ya que ambos conceptos son ambiguos, carentes de teoría que no explican las luchas por el saber hacer y el saber como poder.

Para advertir las resistencias, conflictos y boicot como embebidos en el saber hacer, es importante comprender que en el consentimiento y la negociación es necesario recurrir a un contexto ampliado del significado simbólicos que de cuenta de un doble proceso: la capacidad de *saber resolver*, que hace alusión a la acción individual y colectiva de resolver a tiempo, arreglárselas, lógica de programador, y caja negra cognitiva; y, paralelamente, la posición del conflicto, la actitud del boicot, el sentido de ocultar información, de "ensuciar" el código.

9.4.1.- Conflicto abierto de la resistencia: entre el consentimiento y el boicot

Imitando la reflexión de Simón en referencia a los limites en el razonamiento, podríamos decir que "el individuo no resuelve óptimamente, sino que ofrece un programa informático aceptable, mientras no haya otro que le sustituya". La solución no es perfecta, está embebida en contingencias definidas como "aflicción del software". La no eficiencia, las incertidumbres en el software son tema común entre los entrevistados y las diferentes ingenierías del software que reconocen que no existe un software libre de fallos. Las diferentes ingenierías promueven una serie de *normas disciplinarías* compuestas por métodos cuantitativos, como son metodologías de programación y métricas de calidad; hasta procedimientos cualitativos implementados en las entrevistas con el cliente (Ingeniería de la usabilidad) y códigos morales entre los programadores que impulsan un "código de ética del profesional del software"[116] que pretende en términos generales una disposición moral de que implementen *buenas prácticas* en el proceso de trabajo.

> "...no hay un software que sea 100% perfecto y no hay un software que ya lo hiciste una vez y va a funcionar así para siempre... la tecnología va cambiando, entonces tu requieres que también el software se vaya ajustando a todos estos cambios. Entonces si requieres darle mantenimiento al software que ya existe, ... pero muchas veces la presión del tiempo, las malas practicas, hacen que el código, pues no se desarrolle completamente; digo los sistemas funcionan, pero funcionaran a un X porcentaje, unos mejor que otros, pero siempre vas a tener que darle mantenimiento al software, probablemente sea un mantenimiento mínimo o un mantenimiento mayor, depende de cómo se haya desarrollado, pero todo software requiere mantenimiento" (programador JG2 WALS)

El tercer código de ética indica que "los ingenieros de software deberán asegurar que sus productos y modificaciones relacionadas cumplen con los más altos estándares profesionales".

> "Cada programador tiene su estilo no creo que se dé mucho que uno tenga más conocimiento que otro, es más bien experiencia, por ejemplo si yo encontré (una solución) en cincuenta líneas es porque ya he trabajado antes con estos procesos o ya he hecho algo similar; entonces ya tengo una idea de cómo hacerlo y me sale más sencillo. Igual y él (el programador que resolvió en 100 LDC) tiene muy buenos conocimientos, pero no ha tenido experiencia en éstos procesos..." (Programador CB3, DASA)

---

[116] Código de ética y ejercicio profesional de Ingeniería de software (Versión 5.2), recomendado por el Grupo de Trabajo Conjunto del IEEE-CS/ACM en Ética y Ejercicio Profesional de Ingeniería de Software. Versión amplia en Ingles: http://www.ccsr.cse.dmu.ac.uk/resources/professionalism/codes/secode.html. Ver en español: http://chapters.computer.org/dominicana/contribuciones/Codigo_Etica_Ing_Software.pdf

En el señalamiento anterior es importante destacar la importancia de acumular el *saber-hacer,* entendido como experticia, se trata de acumulación como expresión de poder que posiciona al programador frente a los otros. Ahora bien, otro argumento, para no documentar eficientemente el proceso del código, es que se documenta el *saber hacer* en una serie de procesos que se compilan en un repositorio de rutinas de códigos documentados (librerías), dando lugar a un proceso de "almacenamiento de rutinas" (bibliotecas) que, con el tiempo se acumula una amplia biblioteca virtual de objetos o módulos comentados (Programación Orientada a Objetos), éstos repositorios podrían re-utilizarse en procesos similares, lo cual a su vez permitirá una fluidez de conocimiento acumulados en la empresa y no en el saber hacer del programador; sin embargo, si los textos sígnicos no están documentados eficientemente, si no se comprenden los comentarios porque son insuficientes, si sólo existe un "un mazacote de pensamientos" sin explicaciones, la fluidez del conocimiento se ve frenada.

> "De que java intenta robotizar al programador, no, lo único que hace es que canaliza adecuadamente la creatividad... Lo que robotiza al programador es la inclusión de reglas para hacer las cosas que son indispensables para entregar buenos productos como UML, RUP; si no lo haces, lo que entregas es un mazacote del pensamiento de diez programadores" (Líder DC2 CATI).

La resistencia a documentar esta embebido en las prácticas sociales entre programadores, quienes hacen como que comentan el código, ya que aún con metodologías orientadas a objetos (programación orientada a objetos) que obliga a los programadores a documentar los procesos en la media que codifican, eluden dicha obligación engañando al compilador. Aún con presencia de lenguajes de programación orientados a crear objetos y clases, como: C++, Object Pascal, Java, Eiffel, etc; y lenguajes basados en prototipos: Self, Kevo, Poet/Mica, Cecil, etc. (están en fase experimental, aún no se libera su uso masivo en la industria) las resistencias, boicot y engaños en la documentación persisten y sólo documenta una serie de datos para "cumplir" con la descripción, pero dista de ser eficiente o completa; es decir, el programador "engaña" al sistema, suscribiendo caracteres que validen "la documentación" sin haber documentado eficientemente.

> "....hay de todo. Hay personas que no documentan y que no siguen estándares ni nada. Había un compañero que a sus variables a todas les ponía el nombre de *chico*. Usaba muchos ese nombre de *chico*...., para que fuera más difícil para los siguientes..." (programador CB3 DASA).

Para el programador no documentar eficientemente el algoritmo lo justifica señalando que describir el *como se hizo* es complejo, ya que además el compromiso del programador frente al trabajo es *saber resolver* no describir el *como* resolvió el problema; también se alude a que la *lógica de programador* es un conocimiento abstracto difícil de redactar, se transforma en una "*caja negra*". La *lógica de programador* para *saber resolver* un problema es lo importante en el trabajo, donde la experticia acumulada es la que permite resolver de la mejor manera y no la existencia de docuementos. El programador JO3, de la empresa MIXE señala que aquel programador que resuelve en 50 LDC un problema X en lugar de resolver en 100 LDC, es porque posee habilidades cognitivas complejas, pero intentar suministrarle mantenimiento o actualización en el futuro será complicado y oscuro, ya que comprender la "lógica" de quien resolvió en 50 LDC es como una "caja negra" (programador JO3 MIXE).

JO3 señala que si el programador de 50 LDC indexa objetos ya desarrollados con anterioridad (programación orientada a objetos o reutiliza código) puede ser fácil de comprender el código añadido; en cambio el que resolvió en 100 LDC genera sus propias rutinas y es probable que el de 50 LDC sea mas eficiente al optimizar recursos en el procesador y reutilizar código ya elaborado, que el hecho de volver a desarrollar código (como el de 100 LDC) ya que incidiría nuevamente en la incertidumbre latente en el desarrollo de nuevo código (programador JO3 MIXE). La cita permite resumir el conflicto y boicot embebido en la *resistencia abierta* de no documentar y la *resistencia no dirigida* pero presente en la práctica de ocultar código (código ciego). Por ejemplo, si existe código seguro, es aquel que está suficientemente documentado y consistente en las rutinas y puede reutilizarse o bien en caso de proveérsele mantenimiento o actualización es fácil de comprender; éste tipo ideal de *texto sígnico* es escaso, lo contrario es el código sucio, el código oculto.

> "....suponiendo si yo quiero un reporte de ventas, tenía 8 columnas y ahora quiero que tenga 9 columnas... saber donde se debe hacer ese cambio, no tanto que se debe de hacer o como lo hizo el programador, sino donde me tengo que dirigir para sumergirme en ese mar de código, para poder hacer ese cambio, y tiene que ver con lo que se implementa con estas metodologías de trabajo (UML y RUP) y lo que es la documentación... hay mucha gente que así lo hace, "hago mal las cosas o declaro mal o no hago esto", ¿para qué?, para que al que llegue le cueste trabajo... hay muchas mañas que ya se tienen de ocultar códigos o de hacer varias cosas que dices ¿ohh y esto?, ¿de dónde sale?, ¿cómo sale?.... Si yo no documento y nadie me ve, el día en qué vayan a revisar mi programa no van saber qué hice y porque declare una variable de una forma y porque declare una variable de otra forma.... (Programador JO3 MIXE).

9.4.2.- Conflicto latente: ocultar código, código sucio, código encriptado, etc.

Fuenlabrada (2007)[117] investigador de la Ingeniería del Software del Instituto Politécnico Nacional (IPN) comenta que los problemas comunes en el software para facilitarle mantenimiento y actualización se reducen a la falta de documentación y definición del proceso del *como* y *porque* se construyó determinado orden lógica de algoritmos (texto sígnico) por parte de los que desarrollaron el software, ello se debe entre otras razones a las siguientes tres leyes que ha propuesto: i) ley de la opacidad. La brillante idea que se tiene en la cabeza, cuando se quiere escribir pierde su brillantez: ii) ley del lo obvio que consiste en omitir aquellas cosas que para el escritor son obvias, pero para el lector no lo son; iii) ley de la tortura, cuando para el escritor la idea escrita esta clarísima, pero el lector interpreta otra cosa. Dostoievski llamaba a esto "la tortura de que la palabra no siga al pensamiento".

Las prácticas de ocultar código y el código sucio exteriorizan un conflicto latente (Dahrendof:1959, citado por Edwards y Scullion (1982], 1987:29-30) que se refiere al conflicto que puede no reflejarse en acciones visibles, cuando el poder de una parte es tan grande que suprime las formas concretas de protesta; o bien si existen medios institucionales que presionan, como son las normas, prácticas institucionales o procesos formales que limitan el conflicto, de tal manera que no se observa la resistencia en acciones. Este *conflicto latente*, se circunscribe a un conflicto estructural del poder, entendiendo por estructural porque se impone al proceso de trabajo, que para el caso de la programación, coexisten una serie de *normas discplinarias* explicadas por las diferentes ingenierías del Software y un conjunto de *herramientas procedimentales* como son lenguajes de programación, las metodologías de programación (en cascada, espiral, por capas, etc.); las plataformas de desarrollo (punto net de Microsoft, Base de datos de Oracle, etc.); las metodologías de desarrollo (POO, POA, etc.). Suponemos que *el conflicto latente* es constreñido por las *normas disciplinares* y por las *herramientas procedimentales*. Por ejemplo el programador CM3 de la empresa MEKI, comenta que el programador que resolvió de la mejor manera el problema X (si el que utilizo 50 LDC o el de 100 LDC) es aquel que haya utilizado un lenguaje de programación óptimo al problema planteado, y que no influye la cantidad de código utilizado, por ejemplo si ambos programadores conocen los lenguajes C y C-Sharp (ambos desarrollados por Microsoft) y, uno de ellos lo resuelve en C y tarda una semana, porque resolvió en 100 LDC y el otro en C-

[117] http://www.cidetec.ipn.mx/congreso2007/articulos/30.DOC Acceso 21 08 07

Scharp y resolvió en 50 LDC éste es mas eficiente, ya que 100 LDC en C implica un doble esfuerzo y mas recursos técnicos; si suponemos que ambos lo resuelven en el mismo lenguaje de programación, entonces el que resuelve en 50 LDC es mas eficiente ya que consumirá menos recursos del sistema, y el programador demuestra habilidades y destrezas en el lenguaje y conocimientos de metodología orientada a objetos, que le permiten reutilizar código (programación orientada a objetos POO). Aquel que resuelve en 50 LDC es mas fácil de leer y comprender por otro programador que el de 100 LDC. Este último argumento contradice lo señalado por JO3 de la empresa MIXE, quien señala que la solución de 50 LDC podría ser oscura, compleja y difícil de leer por otros programadores.

El programador GG3 de la empresa MIXE-ET1, añade otra herramienta, la del tipo de metodología de desarrollo que se utiliza, por ejemplo si el programador que resolvió el problema en 100 LDC es mas eficiente, porque quizá utilizó el método de "desarrollo en cascada" utilizando tres capas, donde cada capa contiene determinado numero de módulos, lo que significa flexibilidad en quitar un módulo, reconfigurarlo, reutilizar un modulo en otro proyecto, sin que ello implique errores o fallas en otros dos módulos. A diferencia del programador que resolvió en 50 LDC en una sola capa, lo cual lo hace monolítico, complejo de reconfigurar, no imposible, pero sí con mayores márgenes de incertidumbre (programador GG3 MIXE). La solución del problema dependerá de la "lógica de programador" empleada en *saber resolver*. Ambas respuestas la de 50 LDC o la de 100 LDC son válidas, ambas ofrecen una solución, pero contienen una incertidumbre interna que hemos denominado "*caja negra-cognitiva*" que está embebida, implícita en la configuración de las líneas de código que componen los algoritmos del sistema informático. El programador que resolvió en 100 LDC pueden ser más abstractas, más complejas, y el programador que elaboro 50 LDC encontró un camino sencillo, sin complicaciones. Lo cual no significa que uno sea mejor que el otro (programador GG3 MIXE-ET1).

Ambos programadores coinciden en que las *herramientas* empleadas influyen en la complejidad o sencillez del software desarrollado, por ejemplo el programador GG3, coincide con el programador JO3, cuando advierte que también se puede señalar que el programador que utilizo 100 LDC esta mejor documentado, mejor explicado el proceso de cómo resolvió el problema, es decir no oculto código, comento lo suficiente las rutinas de código, depuro el código (no hay código sucio), procedimientos que si bien representan más LDC (código

seguro), significa facilidades para proveerle mantenimiento y actualización futura al código desarrollado. Por otro lado, la solución en 50 LDC parece sencillo, pero implica complicaciones al momento de proveerle mantenimiento, ya que puede contener código sucio y código oculto (algoritmos para que funcione, no para que sean leídos por otros programadores) de tal forma que termina desechándose éste código (programador GG3 MIXE). En este sentido, GG3 señala que el proceso de comprender la *lógica del programador* y la noción embebida de *caja negra-cognitiva* en el desarrollo del texto sígnico, se relativiza frente a la necesidad de que el software desarrollados, se le debe suministrar mantenimiento y actualización, ello sólo será posible en la medida que se comprenda el código desarrollado, es decir, que sea lo suficientemente documentado, sin importar la extensión de las líneas de código –siempre y cuando sean lógicamente estructuradas- ; para que el software no "envejezca" ante el incremento de las capacidades del hardware, incorporación de nuevo software (interoperabilidad), etc. Conjunto de promesas de la Ingeniería del Software y la Arquitectura del Software que no han resuelto la crisis del software.

Estos ejemplos, nos permiten evidenciar una serie de conflictos latentes, entendidos como sabotaje: *ocultar código, código ciego, código sucio, código duplicado, código encriptado*; entre otros tipos de conflictos intrínseco a la creación de código, análogo a la generación del *texto sígnico* (rutinas algorítmicas), este tipo de conflictos latentes están penetrados en la *práctica cognitiva* frente al proceso de trabajo. Estos conflictos pueden comprenderse como un *sabotaje intencionado,* que limita el propio desarrollo del programa. Al respecto el líder/gerente RS0 de la empresa TADI comenta, que aquel software que no esté lo suficientemente documentado, es complicado ofrecerle mantenimiento o actualización, "Esto es un grave problema...la idea es que exista la documentación... no toda la gente lo hace y...ahí... analizar la lógica de otras personas es muy difícil, muy difícil. Ese es un grave problema"; el gerente RS0 -que también hace software experimental[118]- señala que mucho del software que ha desarrollado como microempresa, no cuenta con la documentación (Gerente RS0 TADI) pero como es software para una cartera de cliente muy específico, porque no lo

[118] El gerente RS0, desarrolla software experimental hacia nichos de mercado específicos que en caso de ser viables significarán una oportunidad de negocio para la empresa. Por ejemplo a lo largo de 5 años desarrollo un proyecto de "laminillas" codificadoras para precios de productos. En el momento de la entrevista estaba haciendo prototipos para la empresa Wal-Mart para implementarlos en los productos, y sin necesidad de sacar los productos del carrito de compras, un lector "leera" las laminillas de precios. Así como otros proyectos que para Diciembre de 2007, ya se transformaron en nichos de mercado.

vende en forma masiva, considera que no representa un problema, ya que cuando hace una actualización del sistema, lo ofrece gratis a la cartera de clientes, como un "plus" por fidelidad al producto. Sin embargo esta respuesta obvia un problema fundamental de la "aflicción del software", ¿Qué sucederá cuando la empresa cierre operaciones?, ¿Acaso los clientes podrán solucionar los problemas de actualización o mantenimiento en otras empresas de software?, supongamos que aún continúa operando la empresa, ¿Sigue laborando el programador que conoce el "el código espagueti" del programa?.

El programador senior MZ3 de la empresa TADI comenta que en el caso de darle mantenimiento o actualización a un sistema informático, sea de 50 ó 100 LDC si no está documentado el proceso, si no esta claro el procedimiento del código, será complejo proporcionarle mantenimiento "Lo mas difícil de (para) un programador es tratar de entender a los programadores y si no esta documentado va a ser difícil. Uno como programador va a preferir siempre hacer algo nuevo que tratar de corregir problemas de otros programadores" (programador MZ3 TADI). El programador MZ3, concluye que es una combinación, una amalgama de factores como la experiencia, conocimientos teóricos, habilidades de abstracción, entre otros lo que permitirá que se desarrollen LDC eficientes (programador MZ3 TADI).

9.4.3.- Conflicto institucionalizado: Trabajar fuera del espacio laboral e interacción pantalla-pantalla.

Dimos cuenta de la construcción social de los juegos y consensos, así como las *operaciones subjetivas*, sin embargo el hecho de *saber resolver* el problema, *el arreglárselas, el arreglo social* están circunscritos dichos juegos a un proceso del *sentido del trabajo como aprendizaje* e intrínsecamente una internalización del problema (aprendizaje situado). Ahora bien, también dimos cuenta que esta industria está ceñida a un proceso de "aflicción crónica", para ilustrar esta crisis, citamos dos ejemplo contemporáneos que estimularon la discusión en torno a la "crisis del software", los fallos del Windows Vista de Microsoft, que fue puesto en venta en el año 2006 con más de 800 errores en el sistema (http://barrapunto.com acceso 6.10.07);o bien la competencia de Microsoft en navegadores de Internet, que es la fundación *open source* Mozilla.org, que en Noviembre de 2007, "subió" en Internet el navegador Firefox versión 3 Gran Paradiso, con mas de 500 errores: Mozilla considero que no afectan sustancialmente al

sistema, es decir "no son de alta vulnerabilidad" ya que se corrigieron los errores mas graves (http://www.hispamp3.com acceso 15.11.07). Ambos ejemplos nos permiten ilustrar que aún con metodologías modernas, sea en el modo de desarrollo privado u *open source* la aflicción del software es latente al proceso de trabajo, donde no influye la alta tecnología o métodos de calidad implementados por Microsoft en el Windows Vista, o bien Mozilla con mas de 3,000 programadores distribuidos en el mundo, ambos programas contenían fallas, aún después de su liberación en el mercado; este ejemplo permite citar los señalamientos de lideres y programadores "no hay software libre de fallos", enunciado que sintetiza las contingencias que envuelven el proceso de trabajo. Señalamiento que el experto en programación DC2 y líder de proyectos de la empresa CATI confirma:

> "Ningún software sale limpio y aun cuando uses los sistemas de calidad que tú quieras, ningún software sale limpio. El proceso de desarrollo del software, incluye el ponerlo en venta en el mercado para que surjan la mayor cantidad de errores posibles que se puedan detectar, les llamamos bugs (errores/fallas) eso es una técnica indispensable, el sacarlo como beta...esos bugs se van detectando con el tiempo y se van corrigiendo con el tiempo" (Líder DC2 CATI)

El párrafo anterior ratifica que en el proceso de trabajo convergen una serie de configuraciones subjetivas, intereses e intencionalidades que opacan y ensombrecen un conflicto de fondo, latente e institucionalizado en el *saber-hacer*, y el saber como poder; el saber hacer está mediado por la internalización del problema, como una interiorización del problema para si mismo en el programador, de aprehender y resolver para si mismo el problema. Es decir, en *resolver* reside la generación de aprendizaje y paralelamente se aprende y acumulan experiencias. Cuando señalamos que hay un conflicto que no está reconocido formalmente, sino que se expresa como una *norma de costumbre* o incluso como una *práctica socialmente aceptada*. Proponemos que en el proceso de trabajo del programador coexisten dos prácticas comunes endógenas al trabajo cognitivo: a) *Interacción pantalla-pantalla* fuera o dentro de la jornada de trabajo. Este proceso hace referencia a la práctica social de interaccionar virtualmente, pantalla-pantalla en el lugar de trabajo o fuera de él, interacción que se sucede como una actividad típica entre los programadores, sin embargo, ¿Por qué acuden los programadores a los foros -Chat-room, blogs o wikis-?; ¿Qué les motiva?; ¿Cual es la intencionalidad al acudir a las comunidades simbólicas?; ¿Qué trascendencias tiene en el *saber hacer* y en el *saber resolver*?, y b) Trabajar fuera de la jornada de trabajo que también

se presenta como una *práctica social* propia del programador. Sin embargo ¿Qué le induce a trabajar en horarios extra laborales?; ¿Qué parte del texto sígnico lo resuelve fuera de la jornada de trabajo?

a).- Interacción pantalla-pantalla

La capacidad de abstracción, motivación, sensibilidad, habilidad y sentido de apropiarse e interiorizas el problema constituyen parte fundamental del *juego de resolver a tiempo*. La posición del programador frente al aprendizaje motivará el grado de participación o resistencia a formar parte de foros, de comunidades simbólicas donde amplíe sus habilidades, acumule saberes y/o se auto-capacite. La participación del programador en actividades virtuales (pantalla – pantalla) en tiempo real o virtual, colaborando o sólo "observando" dependerá de una serie de subjetividades como necesidades de aprendizaje, identidad frente a la empresa, etc., la resistencia o participación se sujetará a una serie de prácticas subjetivas individuales del programador frente al proceso de trabajo y frente al sentido del trabajo como aprendizaje. El trabajar fuera del espacio de la empresa e interaccionar virtualmente se constituyen en conflicto cuando el programador se resiste a utilizar estos medios o bien cuando se internalizan en el programador como parte activa del trabajo, también cuando forma parte de la competencia entre trabajadores por acumular saberes para hacer frente a otros programadores, o bien si se auto-limita o realiza algún tipo de boicot para no participar en los espacios de aprendizaje.

b).- Trabajar fuera del espacio laboral

Como ya señalamos en el capitulo 8, *saber resolver* y el *saber como poder*, implica una serie de juegos y consentimientos como *arréglaselas* donde convergen subjetividades, intereses, motivaciones, intenciones, etc. Para *saber resolver* a tiempo el programador además de interaccionar frente a frente en el proceso de trabajo, también interactúa virtualmente pantalla a pantalla a través de portales electrónicos; interacción que puede ser del tipo sincrónica a través del Chat, pizarras electrónicas o en foros virtuales en tiempo real; también puede ser asincrónicamente, indagando en foros, examinando otros programas; consultando con colegas en diferentes espacios con el fin de *saber resolver* y /o acumular saber como poder. Éstas interacciones y acceso a la información puede ser dentro o fuera de la jornada de trabajo; sin

embargo, el programador puede desistir en cualquier momento a continuar analizando información, limitar su participación en los portales, restringir el auto-aprendizaje, etc.; acciones que dependerán del sentido y representaciones del programador frente al trabajo; del grado de interiorización del problema y la posición del programador frente a la empresa (identidad, compromiso, etc.).

Trabajar fuera del espacio del trabajo e interaccionar pantalla a pantalla en el espacio del trabajo o fuera de el es el conflicto que definimos como de *yuxtaposición del espacio productivo y el espacio de vida,* intentado explicar que se torna en *conflicto implícito* porque los flujos de aprendizaje cognitivo rompen con los espacios tradicionales de la oficina y el espacio de vida personal del programador. El proceso de *saber resolver* no se circunscribe al proceso de trabajo, sino que éste se amplia fuera de las muros de la empresa. Por ejemplo cuando el programador se "cicla", es decir cuando se le dificulta un solución, cuando está agotado y acabándose el tiempo consensado, o bien cuando el programador se empeña, se obstina (dependiendo del grado de interiorización y temperamento del programador; identidad con la empresa, ética de hacer, etc.) en solucionar un problema, el programador subjetivamente reacomoda sus intereses y representaciones frente al trabajo y analiza como "arreglárselas" (individual) o mediante un "arreglo social (colectivo) y *resuelve* parte del problema dentro o fuera del proceso de trabajo.

Consideramos que este *conflicto implícito* no se expresa en una acción concreta de conflicto o resistencia, pero está presente en la actitud y posición de resolver un problema. En las entrevistas se coincidió que las empresas no cubren horas extras, porque se trabaja por metas, por problemas a resolver. Si se desvelan o laboran en su tiempo libre, la empresa no les paga ese tiempo (programador GG3 MIXE)

> "no hay horas extras, por ejemplo si tu te comprometes a sacar la tarea X para pasado mañana, tu dices «Esta sencillo, si lo saco» pero resulta que no, entonces tuviste que aventarte más de ocho horas diarias, te toco desvelarte, etc. y lo sacas pues hasta ahí, pero no hay horas extras." (Programador CM3 MEKI).

Consideramos que las acciones individuales de conflicto o resistencia están guiadas por una serie de subjetividades y presiones estructurales, en este conflicto que denominamos *yuxtaposición de los espacios de trabajo y de vida* tiene que ver con la participación activa o pasiva de los programadores en espacios de aprendizaje, auto-capacitación del programador en foros especializados, por ejemplo http://java.codeproject.com/ (acceso 15 de Abril de 2008)

cuenta con poco mas de 5 millones de programadores registrados. En esta industria estar "en la frontera técnica" es fundamental, ya que la tasa de cambio técnico es más veloz que los cursos que podría ofrecer las empresas como estrategia de capacitación y el alto grado de especialización que exige esta industria, por tanto, es una ventaja que el programador acuda al Internet para fortalecer sus habilidades y destrezas (programador GG3 MIXE)

La auto-capacitación esta presente en el discurso de los programadores quienes tienen una fuerte interacción virtual de auto aprendizaje. Al respecto el programador DINS señala que esta "unas, cuatro horas diarias...en los foros, en sitios de proyectos nuevos, en listas de anuncios de proyectos nuevos como *freenet*[119], en listas de tema de moda como *transport* y mas allá de eso, es lo que está de moda, lo que esta calientito, bajar la aplicación, instalarla, ver como funciona, leer el código, incluso ver como la puedo adaptar a mis necesidades" (programador experto SA3 DINS). El líder de proyecto RS0 gerente y dueño de una empresa con presencia en Latinoamérica, señala que las habilidades son destrezas congénitas.

> "... la habilidad debe de ser innata... difícilmente se te da, o sea si te va a ayudar mucho ,el hecho que estudies y leas y todo..., pero si no tienes la lógica, y la lógica yo creo que se desarrolla a través del tiempo y ya la traes. En la escuela probablemente también la desarrollen o sea que la hayan desarrollado, pero si no tienes esa parte..., no vas a ser nunca un buen programador" (Líder RS0 TADI)

Consideramos que la acumulación de saber hacer y el saber como poder es una acumulación de habilidades y destrezas que se construyen en la interacción situada, interactiva y frente a otros. Sin embargo, en el discurso de los programadores, cuando aluden al conjunto de reglas informales como la *lógica de programador*, la *complejidad de documentar*, por tanto que el texto sígnico es una *caja negra-cognitiva* entre otras reglas, éstos discursos ocultan y obscurecen un conflicto que cubre el *saber-hacer*, en la medida que el *saber hacer* se construye como *poder* se acumulan conocimientos, así que el laborar fuera de la jornada de trabajo, horas extras o con mas intensidad significa una mejor posición en el trabajo y una categorización para negociar frente al otro, el sentido de aprendizaje frente al trabajo, incluso va mas allá del *arreglo social*, *saber hacer* y *arreglárselas*, supone un acentuación del poder. Superando el concepto de "sumisión voluntaria" de Burawoy, o la sumisión colaborativa de

[119] http://freenetproject.org/ Freenet es una red de comunicaciones entre pares descentralizada diseñada para resistir la censura, la cual utiliza el ancho de banda y espacio de almacenamiento de las computadoras de sus miembros para permitir publicar u obtener información de todo tipo en completo anonimato. http://es.wikipedia.org/wiki/Freenet Acceso 18 de Abril de 2008.

Tripier; consideramos que está en juego no el aprendizaje neutral, sino el detentar el *saber como poder* para si mismo, atrás del concepto de *saber resolver* está intrínseco la acumulación de saber y poder.

> "Yo sólo lo sé y no quiero que lo sepas tú, por que si tú ya lo sabes entonces ya no me van a querer, o yo para que voy a servir....digamos que desde un punto de vista egoísta pues si, si tu haces (un algoritmo) y no le dices a nadie y no documentas, si te haces un poco más valioso para la empresa. Por que la empresa dice "Pues es él, el que sabe hacerlo, o sea si lo corremos, perdemos" (programador CM3 MEKI).

En estas entrevistas se sintetiza en parte este conflicto implícito, endógeno al proceso de trabajo, tiene que ver con el *saber-hacer* y el *saber como poder* donde el programador concentra en sí y para sí el *saber hacer* del módulo o rutina que se le haya asignado

> "el conocimiento del proceso (documentación eficiente del texto sígnico) comienza a entrar en juego cuando... a mi me atropellan o cuando a mi me operan de la apéndice y se necesita que alguien entre por mi lugar y continué a partir de ese momento, sino la compañía que me contrato tiene serios problemas.....la mayoría de los proyectos que hago no existe (la documentación) y si quieren reemplazarme necesitan contratar a alguien con la misma competencia que yo y, a parte pagarle horas extras para que se entere de que es lo que yo estaba haciendo, se meta a mi código... a parte cuando se meta a mi código va a encontrar cientos de fallas que yo no había contemplado" (Programador SA3 DINS).

El conflicto institucionalizado, parte del criterio de Burawoy, con respecto al sentimiento de autonomía del trabajador frente a la maquina, ya que aquí el programador no sólo posee autonomía frente al código desarrollado, sino que concentra el *saber hacer*, y el saber como poder frente al proceso de trabajo. Al respecto el gerente DC2, se expresaba que el programador concentraba para si el conocimiento del saber hacer del programa:

> "...le daba poder...por que era el único que podía modificarlo. ...era el único que conocía, y eso tienes que romperlo, no puede ser que alguien que te esta haciendo el trabajo (el programador) se convierta en el dueño de lo que te esta haciendo. Y eso se tiene que romper" (Gerente DC2 CATI)

Hasta este punto hemos descrito los conflictos endógenos al proceso de trabajo que consideramos están presentes ya sea explícitamente o embebidos en las operaciones subjetivas de los programadores. Sin embargo, consideramos que coexisten problemas ligados al proceso de trabajo, pero que son de carácter endógeno al proceso. Es decir son conflictos de carácter estructural, invisibles, pero que ejercen una presión estructural en la acción individual.

9.4.4.- Conflicto estructural del poder: normas disciplinares y herramientas procedimentales

A continuación abordaremos un tipo de conflicto que denominamos como *conflicto estructural del poder*. Partiendo de la propuesta de la definición de *conflicto latente* que hace Dahrendof (1959) citado por Edwards y Scullion (1982], 1987:29-30) quien señala que hay conflictos que pueden no mostrarse en acciones visibles cuando el *poder* de una parte es tan grande que suprime las formas concretas de protesta; o bien, si existen medios institucionales como normas, prácticas o procesos formales que eclipsan el conflicto. En este sentido, las ingenierías del software constituidas como *normas disciplinares* y las diferentes *herramientas procedimentales* reprimen y generan conflictos. Por ejemplo, la separación del software con respecto al hardware en los ochenta; la distancia entre el proceso de trabajo artesanal y el proceso moderno de programación; la existencia de cientos de lenguajes de programación unos orientados a objetos, otros a aspectos, otros a la Web; las distintas plataformas tecnológicas, unas para software privado otras para *open source*; cientos de metodologías para desarrollar software, unas burocráticas como CMMI, MOPROSOFT, IEEE; etc., otras con metodologías ágiles y flexibles; dos modos de desarrollo de software, uno privado otro libre; etc. éstas configuraciones constriñen y configuran heterogéneas dinámicas en la industria del software.

Para Vigotsky (1988, 1995) la internalización del aprendizaje no es ajeno al contexto cultural donde se genera e influye predominantemente el medio cultural, como el lenguaje, el juego simbólico de la escritura, la memoria, la atención, etc. en este contexto las *herramientas procedimentales*, las *normas disciplinares* y los símbolos son instrumentos mediadores para el aprendizaje situado. Por *herramientas procedimentales* entendemos un conjunto de metodologías, métricas de calidad, métodos y procesos utilizados para crear los algoritmos que darán lugar a los programas de software. En el cuadro 26 se enlistan las diferentes *herramientas procedimentales*, sin embargo esta lista es mínima, frente a las tasa de crecimiento de estas herramientas; las *normas disciplinares* hace referencia al conjunto de supuestos e hipótesis de las diferentes Ingenierías (del Software, de Requisitos, de usabilidad, etc.) se constituyen como un conjunto de estructuras que presionan el proceso de trabajo.

Cuadro 26
Conjunto estructural de herramientas procedimentales[120]

o	+ de 2000 lenguajes de programación
o	Cientos de bases de datos
o	Cientos de plataformas de desarrollo
o	Variadas metodologías de desarrollo, etc.
o	Variadas métricas o procesos de calidad
o	Dos modos de desarrollar software: Privado y de acceso libre al código.

Fuente: elaboración propia en base a las entrevistas.

Estas condiciones representan el contexto en el cual se localiza el *conflicto estructural de poder*, consideramos que coexiste un conjunto de jerarquías no reconocidas al interior del proceso de trabajo, configuradas en la atmósfera de las *normas disciplinares y las herramientas procedimientales*, y su influencia en el proceso de trabajo. Ya en el capitulo 8, explicamos éstas estructuras que se constituyen como una serie de métodos, métricas, y proceso que se erigen frente al proceso de trabajo como "las únicas vías para solucionar la crisis del software" que, como señala Foucault, las normas se erigen frente a los individuos restándoles poder y constituyéndose con leyes y reglas que imponen poder. Suponemos que las *normas y las herramientas* se erigen con poder e pretendiendo arrebatar el poder a los programadores, proponiendo para ello, una serie de métodos y métricas como "la buena nueva" para resolver de fondo el conflicto en el proceso de trabajo. Es decir las *herramientas procedimentales*, se constituyen como procesos que establecen *rutinas técnicas en el saber resolver*, construyen una serie de *tecnicismos burocráticos* que promueven "buenas prácticas" en el proceso de trabajo. Sin embargo, tanto las *normas* como las *herramientas* se enfrentan a la crisis del software, crisis que continúa presente en la programación moderna y que podemos señalar que aún con normas y herramientas de cuarta generación: no hay software libre de fallos, no hay software con cero errores, no hay software justo a tiempo, no hay software cero defectos.

[120] Una base de datos o banco de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. En la actualidad, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos tienen formato electrónico, que ofrece un amplio rango de soluciones al problema de almacenar datos http://es.wikipedia.org/wiki

Una plataforma de desarrollo es el entorno común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo, sin embargo, también es posible encontrarlas ligadas a una familia de lenguajes de programación o a una Interfaz de programación de aplicaciones o API por sus siglas en inglés. http://es.wikipedia.org/wiki

Estas particularidades, nos llevan a considerar que en el software aún prevalece un alto contenido "creado a mano", creación que configura una serie de resistencias que entran en conflicto la programación moderna. La ausencia de "un solo mejor camino", conduce a los programadores a una serie de dinámicas creativas por acumular el *saber hacer*. Por ejemplo, *fluidez cognitiva* en el proceso de trabajo, garantiza que el *saber hacer* fluya, ésta fluidez legítima los procesos informales del juego, el consenso, el consentimiento y los arreglos sociales. Por ejemplo, al aprendizaje situado entre el programador senior y el programador junior, el primero es aquel programador con más habilidades, destrezas y capacidades de experticia frente al programador junior, que es el programador novato, con menores cualidades idóneas. El programador JO3, de la empresa MIXE comenta que hay dos categorías de programador, el programador senior, que lo caracteriza el grado de experticia, y el programador "shadow" (sombra) o Junior quien está bajo la responsabilidad del programador senior, "el programador junior hace ciertas actividades que le pagan al programador senior" (Programador JO3 MIXE). Ahora bien, éstas categorías no son formales en la empresa, es decir aún que en los hechos se utilicen dichas categorías, ello no significa que les asignen un salario mayor, o que les asignen un tabulador transparente entre ambas categorías. Al respecto el programador JO3 señala que aún después de que evaluaron sus habilidades, no le asignaron una categoría, si serán catalogados como progress-advisor senior (especialista en base de datos Progress), o se quedarían en el puesto de "programador" "...iban a observar todavía un mes, haber como resultaba el aumento de responsabilidades, en base a eso...será definitivo si van a aumentar o no el sueldo" (Programador JO3 MIXE).

Consideramos que estas categorías se utilizan mas para diferenciar grados de expertiz en determinadas plataformas tecnológicas, lenguajes de programación ó conocimientos de herramientas, que le son útiles al empleador que convoca la plaza de trabajo o evalúa los conocimientos de los recursos humanos; empero hacia el interior del proceso de trabajo estas jerarquías se desdibujan, se traslapan, cobrando relevancia las capacidades del *saber hacer* más allá de los títulos de programador senior, junior, shadow, etc. El nombre de la categoría laboral es relativa, porque predomina mas las habilidades y destrezas del programador y las capacidades que requiere el proyecto que se esta desarrollando, es la complejidad del proyecto el que exige determinado conjunto de habilidades y destrezas, ya que éstas varían de un proyecto a otro; por tanto las jerarquías laborales se desdibujan frente a ceda proyecto, ello no

significa que la empresa y la industria en su conjunto no premien a quienes poseen habilidades y destrezas. La industria en su conjunto premia a los programadores más cualificados, donde no median los conocimientos formales, sino las destrezas, habilidades y cualidades.

Consideramos este supuesto como válido, en el sentido que al hacer una búsqueda en google con respecto a "programador senior" se despliegan mas de 117 mil páginas con este título, que en una rápida lectura a unos cuantos, observamos una tendencia a etiquetar como "programador senior" al programador que posee cierto grado de experiencia, pero en aquellos conocimientos que la empresa demandante considera pertinentes, como son conocimientos en determinadas bases de datos, diversas metodologías o procesos que requiere la empresa; etiquetado el cuadro de necesidades de la empresa bajo el concepto de programador senior, es decir no rigen los años o numero de proyectos en los que se haya participados, o en los miles de códigos que se hayan escrito; incluso no prevalece la acumulación de herramientas tecnológica, lenguaje de programación o método; sobresale las necesidades de la empresa, como es el criterio de especialización, el conocimiento de programación moderna, en herramientas y normas recientes Por ejemplo, en el siguiente anuncio de contratación para programador senior, observamos que la categoría de programador senior no es relativa al grado de experticia del programador, sino a la especialización que, en este caso, es en el uso de las herramientas de desarrollo de herramientas y normas Oracle:

> "México-DF - Solicitamos programador Senior Oracle Applications con experiencia en: *Oracle ERP; *Excelente nivel de programación con Developer 6i; *Excelente nivel de programación con PL/SQL, SQL Plus y solución de querys complejos.; *Conocimiento de UNIX, manejo de shell. ; *Conocimiento de base de datos Oracle. ;*Conocimientos de manejo de API's e Interfaces de Oracle Applications en sus diferentes módulos. *Configuración de nuevas customizaciones y navegación en Oracle Applications *Conocimiento técnico estructura de base de datos" [121]

En este publicación se le denomina programador senior a quien esta especializado en el uso de las *herramientas y normas* de Oracle. En el siguiente párrafo, la solicitud de programador junior hace referencia más a las "buenas prácticas" como la confianza y las habilidades para resolver problemas, sin considerar del todo el conjunto de herramientas, es decir se hace referencia a desarrollar destrezas dentro de ciertas herramientas tecnológicas:

---

[121] Datos de la Empresa: Lic. Julia López; laboramexico@gmail.com; Labora México; Mina #101 Col Guerrero; México, DF 06300." http://jobsearch.occ.com.mx/getjob_esmas2003.asp 23.11.07 21.10.07

> Knowledge on Demand S.A. de C.V. empresa mexicana líder en el ramo de TI solicita programador .net junior "Programadores C# .NET" que cubran el siguiente perfil. Confiamos en que tu conocimiento y habilidades formen parte importante de nuestra organización. Habilidades: Proactivo; Capacidad para trabajar en equipo; Facilidad para detectar problemas y solucionarlos. Conocimientos técnicos requeridos: Comprensión avanzada de programación orientada a objetos; Conocimientos sólidos de C# enfocado a .net; Conocimientos de SQL; Conocimientos de XML. Requisitos: Disponibilidad de tiempo completo; Excelente presentación; Puntual; Buena disposición; Experiencia mínima de un año usando las tecnologías requeridas.[122]

Pensamos que la *resistencia estructural del poder* se constituye y enfrenta en la atmósfera de necesidades del empleador, de las necesidades del mercado profesional, de los conflictos entre el uso de una herramienta u otra, entre un lenguaje de programación y otros, entre programar con herramientas del software privado o herramientas del open source, entre considerarse programador senior o junior. Entre estas dos categorías laborales coexiste un abanico de "estratos laborales" como son: programador semi-junior; programador semi-senior, programador master, programador senior, etc., sin embargo, lo que media en estos es una combinación de especialización y habilidades cognitivas, que consideramos están inmersas en una paradoja del aprendizaje: especializarse en *herramientas y normas* recientes, la incertidumbre es ¿Cuáles garantizan mejor desempeño?, si el mercado profesional es una turbulencia en nuevas herramientas y normas ¿Cuáles garantizan que no sean sustituidas, en el periodo de aprendizaje del individuo? Por ejemplo, un programador experto de Teléfonos de México comentaba que la crisis del software que sobreviene es por el déficit de programadores en Cobol y Fortran, lenguajes de programación de los setenta, y sin embargo, las supercomputadoras IBM de los setenta y ochenta que operan en grandes empresas corren sus programas a la medida en éste lenguaje de programación, y hacia fines de la primer década del siglo XXI, son escasos los que utilizan éstos lenguajes. Otra paradoja del aprendizaje, es la persistencia de la crisis del software, aún con la programación moderna.

Por el momento esta paradoja le confiere al programador un control sobre el proceso de trabajo, medido por la acumulación de habilidades y destrezas en *saber resolver*, que le dan al programador el reconocimiento de senior o junior, de experto o de aprendiz. Por ejemplo el caso citado de la empresa que contrata un programador senior que posee destrezas y habilidad en el uso de las herramientas de Oracle; herramientas que no necesariamente son útiles para

[122] http://jobsearch.occ.com.mx/getjob_esmas2003.asp?ss=&jobid=1836647 21.10.07

otra empresa que desarrolla en base a herramientas de punto.net o para las empresas que desarrollan en *open source*, como la organización Mozilla.org; sin embargo, lo que da soporte a las habilidades y destrezas en el *saber hacer* y el *saber como poder* son las dinámicas cognitivas y la inserción del programador en flujos cognitivos de aprendizaje.

> "Yo personalmente, estoy certificado en Linux por el LPI en el nivel de Administrador Júnior de los módulos 101 y 102 y también soy Ingeniero Senior Certificado Novel en particular sobre lo que es SUXES, también hicimos en algún momento una certificación con C-Linux y también ahí logre la certificación de Professional senior Linux Ant faller, entonces eso mas la experiencia que ya tenemos de 6 años trabajando pues nos dan las habilidades y los conocimientos necesarios para implementar soluciones de todo tipo basados en OPEN SOURCE" (Líder WALS JG).

9.4.5.- Conflicto estructural- explícito: fluidez cognitiva en el aprendizaje

Para las empresas de software a la medida cobra relevancia un modo de resistencia que es dinámico, flexible, ágil y que posee como la fluidez en *saber hacer* y el *saber como poder* entre los programadores, donde la intención es *saber resolver* no explicar el *saber hacer*. Aludimos al proceso de *fluidez cognitiva en el aprendizaje* que se erige como resistencia explicita a las normas y herramientas que promueven la documentación del *saber hacer*. En este proceso no prevalece la división del trabajo Diseño-Desarrollo-Implementación que promueve las normas disciplinares; no se persigue una metodología o métrica en específico, sino que el *saber hacer* se realiza conforme se va planteando el problema. Es ágil, no burocrático, es fluido no jerárquico, es horizontal no modularizado entre los agentes. Es ágil porque no se conforman equipos de trabajo burocráticos o rígidos, sino que el *saber hacer* es entre pares para evitar la perdida de la información, es decir hay dos programadores y fluye entre ambos el *saber hacer*, se intercambian de puestos de trabajo sin previo aviso, la documentación es mínima, el *saber resolver* no esta en la documentación del código, sino en la claridad del texto sígnico (código auto-documentado). Las *normas disciplinarias y las herramientas procedimentales* se orientan mas a estructuras que dividen el *saber hacer*, separan el diseño y el desarrollo, por el contrario en la *fluidez cognitiva en el aprendizaje* se da prioridad a compartir el código, antes que la documentación se prioriza la *lógica del programador* y desarrollar código entendible (auto-documentado); la solución del problema no está en la documentación del saber hacer sino en el acceso al código para que otro programador reelabore el código mediante un proceso transparente "menos ciego".

Conclusiones generales

Presuponemos que hacia fines del siglo XX la producción capitalista clásica fundada en las relaciones diádicas entre producción y circulación; entre empresario y trabajado, se desdibujan frente a la mayor presencia del cliente o usuario final en la producción y la cada vez mayor creación de bines inmateriales (De la Garza, 2007). Todo sistema de producción posee dos dimensiones: la material y la simbólica, si predomina la dimensión material es producción física, tangible; si predomina la dimensión simbólica, como la información y el conocimiento, la estética, lo armonioso, lo sentimental, hablaremos de una producción simbólica.

La producción simbólica en las formas de producción capitalistas cobra relevancia hacia fines del siglo pasado como producto de consumo final, donde la información y el conocimiento son materia prima. Coexisten diferentes tipos de trabajos simbólicos: *Trabajo cognitivo*, se entendería por aquellas actividades en las cuales predomina el aspecto intelectual, reflexivo, por ejemplo, la actividad académica, el periodismo, la abogacía, el diseño gráfico, el desarrollo de software, etc. *Trabajo emotivo,* estaría compuesta por diversas actividades en las cuales los sentimientos, las emociones, las pasiones, etc. están presentes en mayor medida, (Bulton:2006, Vigotsky;2004 [1934], Girard:1975), por ejemplo las empleadas de mostrador, los sobrecargos en aviación; empleados de casas de la tercera edad, guarderías, etc. *Trabajo estético*, formarían parte aquel conjunto de actividades que se transmiten a través de signos abstractos como el sentimiento, las formas, expresiones o composiciones de notas musicales, libretos, literatura, etc., por ejemplo, los compositores e interpretes, los dramaturgos, escritores y literatos, en el arte fotográfico, pintura, escultura, danza, etc.

Es importante distinguir dos espacios generales del trabajo simbólico: *Trabajo simbólico subjetivo* y *trabajo simbólico que se objetiva.* Los productos o bienes subjetivos en la esfera del consumo, son aquellos que se producen y consumen en un mismo acto, producción-consumo se yuxtaponen, por ejemplo, aquellos que se generan en restaurantes con venta a la carta, un concierto musical en vivo, el servicio médico, el asistir a cursos, seminarios o pláticas de académicos, obras de teatro, cinematografía, turismo, el servicio del abogado, atención de la tercera edad, guarderías, servicio aéreo, terrestre, etc. El producto y el trabajo subjetivo se realiza en el acto mismo que se consume. El *trabajo simbólico que se objetiva* es aquel producto que se objetiva fuera de la esfera del consumo, pero continua supeditado al consumo, por ejemplo el desarrollo de páginas Web, el software a la medida,

cine animado por computadora, entre otros productos simbólicos que, si bien es cierto la producción y consumo están yuxtapuestas, ya que es en el consumo donde crean su potencial, pero no es menos cierto que éstos productos pueden almacenarse, copiarse, clonarse y revenderse fuera de la esfera de la producción-consumo. En la producción simbólica, la materia prima es la información y el conocimiento, que se propaga en un contexto de fluidez, posee la particularidad de continuar reproduciéndose, aún fuera de la esfera de la producción (piratería, clonación, reutilización), por ejemplo la industria del software, el diseño virtual, operaciones financieras (se objetivan en perdidas o ganancias), cine animado por computadora, entretenimiento *on line*, etc. Al respecto Marx ya había señalado que la separación entre lo material y lo simbólico no es del todo claro, argumentándolo en su clásico ejemplo de las abejas constructoras o la araña que teje complejos sistemas de redes, ambos procesos son superados en mucho por el albañil menos diestro, por el hecho que éste concibe de manera abstracta (pensamientos) la tarea que ejecutará posteriormente (Marx:1973, 130-131); esta cita de Marx la podemos ampliar, señalando que el albañil concibe el trabajo que realizara pero dicha abstracción esta sujeta a una *constelación de configuraciones subjetivas*: como la cultura, experiencia, habilidades, destrezas, intereses, sentido del trabajo, etc.

Proponemos que el trabajo cognitivo en el desarrollo de software a la medida es un proceso de simbolización y negociación de significados que se objetivan, donde la objetivación implica varias fases o etapas, sin que ello signifique una estricta división del trabajo. Una primer fase es la resolución de problemas planteadas por el cliente, quien colabora junto con el programador o analista el diseño del problema a resolver; una segunda fase es cuando el programador transforma los requerimientos del cliente en algoritmos lógicos que resuelven el problema o parte del problema planteado por el cliente; en una tercera fase, el cliente implementa el producto simbólico (software). El trabajo cognitivo del software, rompe con las rigidez de la manufactura, ya que no sólo se almacena, reproduce y reutiliza con un costo que tiende a cero (rompe con las economías a escala), sino que la multiplicación se sucede fuera del proceso de trabajo (piratería y clonación del software) y, prácticamente no se requiere de la intervención del cliente y de los programadores que lo desarrollaron (reproducción fuera del espacio laboral); sin embargo ello no implica que los programadores se "independizaron" de la información y conocimiento que dio origen al software, permanece embebido no en la empresa, sino en el saber hacer del programador. De tal forma que

podemos establecer algunas características del trabajo cognitivo en la producción de software a la medida: en su desarrollo existe un relación triádica entre clientes, productores y trabajadores; el producto cognitivo se objetiva fuera del proceso de trabajo; está compuestos por signos (algoritmos) y símbolos (interfaz gráfica), existe una presión cognitiva en el saber resolver de la mejor manera posible; coexiste una orientación subjetiva con respecto a mayor creatividad; coexiste una acumulación de habilidades y destrezas cognitivas; la colaboración y el juego están mediados por conflictos y resistencias por el saber hacer y el saber como poder; el saber como poder se distingue en creación y decisión por una "ruta cognitiva" en el desarrollo de los signos, elección que está embebida por incertidumbre y contingencias.

La característica principal del trabajo cognitivo del software supone una redefinición de la actividad laboral a nivel de interacción sociales entre agentes que participan, como el programador, el cliente y el gerente; interacción que no necesariamente sucede en un contexto fordista, supeditada a "tiempos y movimientos" ordenados por un capataz o gerente; consideramos que las interacciones se producen en un contexto colectivo que genera flujos de aprendizaje a través de interacciones de tiempos-pensamientos y movimientos reflexivos; por tanto cabría señalar una *construcción social de trabajo ampliado* que se sucede al interior de comunidades simbólicas de trabajo (De la Garza, 2006a:16). En *las comunidades simbólicas del trabajo de Software* implican una forma de organización laboral no Taylorista, donde se producen símbolos. las relaciones laborales se configuran en un contexto de contingencias, participación activa y creativa del cliente, yuxtaposición de las fases del proceso de producción-circulación-consumo, acumulación del saber hacer (información y conocimiento).

Las comunidades simbólicas del trabajadores de software a la medida es un concepto amplio que busca explicar la *constelación de configuraciones subjetivas* que se suceden en la interacción dentro y fuera del proceso de trabajo y están comprendidas por un conjunto de trabajadores que establecen relaciones sociales amplias, extendidas hacia el interior como al exterior del proceso de trabajo en tiempo real y virtual, sincrónico y diacrónico; relaciones sociales que no son estructurales, verticales o rígidas, sino que pueden ser cara a cara, pantalla a pantalla, en tiempo real o virtual, etc., no están limitadas o restringidas al ámbito laboral; es una comunidad con estructuras que le constriñen y con grados de libertad en la toma de decisiones que implica un sistema de prácticas subjetivas, individuales o colectivas; como dinámicas relacionales. Las *comunidades simbólicas del trabajo* de software estarían entonces

constreñidas por una serie de estructuras que forman parte del proceso de trabajo como: un conjunto de *normas disciplinares* que son constituidas por el conjunto de Ingenierías del Software (Ingeniería de la Usabilidad, Ingeniería de Requisitos, Arquitectura del Software, entre otras) y, las *herramientas procedimentales* (métricas de calidad, metodologías, ciclos de vida, etc.), éstas estructuras presionan y constriñen la acción individual de quienes sustentan el *saber resolver y como resolver* . Las normas como las herramientas se erigen como un conjunto de leyes y procesos (gobernanza) que señalan el *como hacer* desde la gerencia, sin embargo la *aflicción del software moderno*, continúa presente en la creación de algoritmos, es intrínseca a la transformación de símbolos (requerimientos en algoritmos) y está presente en la redacción de textos sígnicos (que es la transformación de los signos presentes en los lenguajes de programación y la combinación de éstos en líneas de código), presencia que se hace objetiva en los problemas de operabilidad, falta de calidad, exceso del presupuesto y tiempos estimados, etc.

El proceso de trabajo en el desarrollo de software a la medida no culmina en la elaboración del último código, se extiende fuera del proceso de trabajo y abarca el proceso de producción del cliente y los tiempos de vida del programador. El software, debe valorarse (testarse) y comprobarse que no haya problemas de operabilidad interna e interoperabilidad con otros programas. Al implementarse en el ambiente (hardware y software) del cliente y ser compatible y claro para los usuarios finales (interfaz gráfica) se acepta socialmente, en caso que no haya incertidumbre, ya sea interna o externa. La primera hace referencia a problemas de compatibilidad con el ambiente informático del cliente o errores internos; la externa hace referencia al rechazo social, mediante *prácticas subjetivas* de resistencia, boicot, mal uso, etc. por parte del usuarios final, ya que se rompen arreglos tácitos en el proceso de trabajo del usuario final; es decir, el usuario final forma parte del proceso de trabajo, ya que el software desarrollado potencia y reestructura la organización del cliente y, dependiendo del tipo y robustez (cantidad de funciones) del software, se modificará la organización interna de la empresa contratante alterándose las relaciones de poder del usuario final. El proceso de trabajo del software no culmina con "empaquetarlo" en el CD de instalación, ya que éste debe ser implementado, probado y aceptado por el usuario final; no debe haber problemas de operabilidad y que este libre de conflictos de inter-operabilidad con los distintos programas del cliente, que haya compatibilidad entre el software instalado y el hardware del cliente,

proceso que según los gerentes entrevistados puede llevar de semanas hasta meses, dependiendo de la complejidad del producto. También depende, sí el cliente es simultáneamente usuario final del software o bien se trata de un cliente intermediario (consultorías de nuevas tecnologías).

Los usuarios finales son aquellos que utilizarán cotidianamente el software desarrollado, convienen un periodo de tiempo para estabilizar el sistema, se ajustará a las necesidades del cliente, quien señalará si se cumplieron o no los requisitos previamente acordados, aceptando el software desarrollado cuando se confirme que no existen problemas de compatibilidad (algunos clientes optan por solicitar una póliza de garantía y mantenimiento del software, lo cual implica que el desarrollo del software se extiende mas allá del proceso de gestión). En caso de que se rechace el programa o haya problemas de compatibilidad, se revisan los acuerdos de requisitos o se rediseñan aquellos errores de configuración. Al respecto la ingeniería de la usabilidad, señala que 20% de los errores del software devienen de un mal uso del mismo por parte de los usuarios finales, quienes no se familiarizan con la nueva interfaz grafica. Sin embargo, 80% de los errores restantes es un conjunto de contingencias inherentes al proceso de desarrollo del software a la medida.

En el proceso de producción del software a la medida están presentes una serie de contingencias, eventualidades y procedimientos no clásicos en el trabajo en la manufactura. El contexto de imprevistos y circunstancias en el proceso de trabajo del software se le conoce como "aflicción del software", alude a una serie de contingencias no previstas, no predecibles, que están ocultas en el proceso mismo. Están asociadas al *como* se construyo el programa (diseño) y el *porqué* se creó determinado proceso algorítmico ó rutinas de algoritmos (texto sígnico), la ausencia de información y conocimiento del *como* se hizo (documentación relativa al proceso) y la nula existencia de información disponible del *porqué* se configuró el software en determinadas rutinas o textos sígnicos (comentarios al código) implicó limites a la modificación, mantenimiento y actualización del software; complicando la migración hacia nuevas plataformas. La estrategia que se prosiguió fue reescribir el código, iniciar de cero todo el programa o bien reescribir "pedazos de código" (incrustaciones) que hacían las veces de dispositivos "puente" entre el software del tipo artesanal y software moderno.

Sin embargo, es ilusorio señalar que la "aflicción del software" está ceñida al contexto de la etapa artesanal del software, ya que actualmente las contingencias e incertidumbres

persisten en software moderno, sea éste desarrollado por fábricas especializadas de software (Windows Vista de Microsoft) o por microempresas que construyen software a la medida. La Ingeniería del Software reconoce que no hay software 100% seguro, confiable y libre de errores (bugs), lo cual significa presencia de la "aflicción del software" en los terrenos de la programación moderna[123]; basta mencionar el actual reto que significa la migración del sistema de 32 bits a 64 bits, dificultad que deviene del hecho que el software está hecho a la medida, no sólo de las necesidades del cliente, sino que se ajusta al contexto tecnológico del hardware y software que posee el usuario final.

El desarrollo del software no está restringido a las "leyes" de la manufactura tradicional o por los límites físicos de la fábrica propios de la producción en serie del fordismo y el toyotismo. El proceso de trabajo del software no es material, es cognitivo, el trabajo no es esfuerzo físico, es reflexivo; la eficiencia y la productividad del trabajo no se ajusta a las economías de escala, no obstante que las Ingenierías del Software proponen medir la productividad y eficiencia con métricas de producción por hora-hombre o errores por cada mil líneas de código. En el proceso de trabajo del software las economías a escala no operan como en la manufactura; en el proceso de trabajo del software el primer programa concentra el costo total y en el segundo software el costo tiende a ser equivalente al costo del dispositivo donde se genere la copia. Es decir, en el trabajo clásico, los costos son inversamente proporcionales al número de piezas, en cambio en el trabajo del software el costo se concentra en el primer producto, el segundo software generado su costo tiende a cero; el software no se deteriora con el tiempo (suponiendo que no haya cambios importantes en el ambiente tecnológico donde se implementó), pero en caso de presentarse errores y fallas en el software postproducción, no significa que sean errores generados en el uso, sino que ya estaban ocultos en el sistema El software contiene fallas implícitas al *ciclo de desarrollo* (periodo de tiempo del proceso de trabajo que tarda en desarrollar el software) que se conforman en el diseño (conceptualización y formalización de requerimientos) o bien en el procesamiento de los datos (generación de rutinas de algoritmos o textos sígnicos) y se manifiestan en las pruebas de calidad o en la implementación (con el cliente o el usuario final), o bien permanecen ocultas, latentes, es decir *no hay software con cero defectos*. Es importante señalar que el software con errores y

[123] Véase ejemplos reciente de la "aflicción del software" en la programación moderna, en www.oonumerics.org/. Acceso Junio de 2007

defectos no se rechaza, no se destruye (dependerá de la magnitud), se "parcha" y se asume como inevitable que se incrusten nuevos dispositivos de "actualización"; sin embargo el error y defecto no son nuevos, éstos ya estaban presentes en el proceso de trabajo y afectan a todas las copias del mismo. *No hay software justo a tiempo,* no sólo debido a la falta de métricas y metodologías de calidad eficientes, sino que tampoco hay una forma de predecir los errores y defectos. Aún con *herramientas procidimentales* de cuarta o quinta generación (programación orientada a objetos y aspectos, métricas de calidad tipo CMM, estándares IEEE, etc.; el proceso de trabajo contiene una alta dosis de *"creado a mano"*, donde los fallos y errores están contextualizados por una serie de contingencias: persistencia de una continuada ausencia de especificaciones coherentes, lógicas y argumentadas (documentación del diseño, y especificaciones de los módulos) tanto a nivel del desarrollo del proyecto, como para el cliente; informalidad en los comentarios del *porqué* se configuraron los textos sígnicos; ausencia de planificación en el desarrollo del proceso de trabajo; escaso empleo de normas de calidad; insuficiencia o limitación en el seguimiento y gestión de los módulos desarrollados como en el procesamiento de datos; necesidad de personal cualificado con habilidades, destrezas, experiencias en nuevos métodos y plataformas tecnológicas. Así como un entorno polarizado en las calificaciones impartidas por el Estado (heterogeneidad en las profesiones).

Aunado a lo anterior, el proceso de trabajo del software es atravesado por una ausencia de recetas únicas, además de no existir un control total de calidad, cero errores, cero defectos e incumplimiento en entregas justo a tiempo, se suma lo imprevisible de explorar *"un solo mejor camino"* para desarrollar los proyectos de software que se soliciten. En las entrevistas los líderes y gerentes sostenían que el proceso de trabajo en el software debería ser como la manufactura de automóviles, donde existan tolerancias al error, rangos para las fallas por cada número de unidad producidas, etc., igual debería corresponder al software; sin embargo, en las mismas entrevistas, señalaban que cada proyecto es *sui generis*, con contextos determinados, únicos e impredecibles; cada proceso para desarrollar un software posee lógicas propias, internas, que varían entre proyectos. Cada software se enfrenta a contenidos diversos, (requerimientos a la medida), necesidades de actualización y mantenimientos específicos, es decir, que los programas se modifican a lo largo del ciclo de vida (periodo de tiempo en el cual un programa deja de ser útil y es remplazado por otra versión u otro programa).

Frente a este conjunto de contingencias, en los últimos quince años se ha estructurado una serie de *normas disciplinares* que no sólo señalan una recomposición de metodologías, métodos, procesos y disposiciones al interior de las Ingenierías del Software, como son la Ingeniería de Requisitos, Arquitectura del Software, Ingeniería de la Usabilidad, entre otras, que intentan establecerse en segmentos del ciclo de desarrollo (periodo de tiempo en el cual se desarrolla el proyecto) y los ciclos de vida del software (periodo de tiempo que el software es utilizado por el cliente). Esto es, que la Ingeniería de Requisitos -por ejemplo- instituye una serie de procedimientos para conceptualizar el conjunto de necesidades del cliente, una vez que los requerimientos son especificados; otro ejemplo es la Arquitectura del Software que formaliza los requerimientos en un diseño que contiene módulos y tareas a desarrollar, vigilando la calidad, eficiencia y procedimientos en la implementación del código; otro ejemplo es la Ingeniería de la usabilidad, que intenta normar el desarrollo de la interfaz grafica del software, que el diseño visual para el usuario final sea accesible y comprensible (implementación del software desarrollado). Este conjunto de *normas disciplinarias* propuestas por las Ingenierías del Software proponen un conjunto de *herramientas procedimentales* que presionan al proceso de trabajo; *normas* que pretenden *"vigilar y castigar"* a quienes no cumplen este conjunto de normatividades que tienen como criterio general estandarizar y dividir los *flujos cognitivos de aprendizaje* en los cuales fluye la información y conocimiento como aprendizajes entre quienes participan en el proceso de trabajo; se pretende estandarizar el proceso de trabajo mediante metodologías de desarrollo, proporcionando a los actores una especie de guías (configuración del software y métricas de calidad) que especifican los pasos a seguir del *como* deben planificarse las tareas, documentación de los procesos, etc.; es decir, se pretende codificar *el saber hacer y el saber resolver* en la creación de textos sígnicos (explicitar el proceso). También coexisten estándares del *como* especificar los procedimientos del código, con la intención de minimizar la presencia de conflictos en el desarrollo de los textos sígnicos como: código oculto, código sucio o los engaños para *saber resolver* problemas. Es decir, el conjunto de Ingenierías del Software confeccionan un conjunto de *herramientas* que intenta hacer visible la *fluidez cognitiva en el saber hacer* que se soporta en comunicación e información que atraviesa formal e informalmente el proceso de trabajo, que no es otra cosa que el intento de transformar

el conocimiento del saber-hacer (conocimiento tácito) en una serie de referencias explicitas (documentar el proceso).

Las *herramientas* y las *normas disciplinarias* -que le dan cuerpo las Ingenierías del Software- que se constituyen como relaciones de poder verticales. Las Ingenierías del Software se instituyen como un conjunto de *normas disciplinarias* instauradas por diversos organismos, ya sea públicos, privados, institucionales o descentralizados que gestionan una serie de reglas normativas que coaccionan el proceso de trabajo[124] y presionan a los actores que intervienen. Sin embargo, frente a éstas *normas* se constituyen y coexisten formas de conflicto, boicot y resistencias en las relaciones del *saber hacer* y el *saber como poder*, que se traducen en acciones individuales, estructurales y colectivas de resistencia y lucha. Conjunto de resistencias como conflicto que están embebidas en un doble juego del *saber hacer* y el *saber como poder*, por un lado es un proceso de creación y transformación permanente a través de juegos de poder, del consentimiento como el *arreglárselas* para *saber resolver*, otras veces es de consenso como el *arreglo social* mediante la comunicación entre compañeros, ya sea al interior como hacia al exterior del proceso de trabajo, donde parecería ser que la resistencia no es anterior al poder y que es inherente a las relaciones del saber (Foucault:1991); sin embargo, la resistencia posee matices intrínsecas como el conflicto, el enfrentamiento, la lucha, el boicot, oposición, ya sea como un acto individual, grupal o social. Podemos entender la resistencia como una respuesta que configura un conjunto de significaciones y representaciones implícitas del saber como poder, se configuran percepciones subjetivas del sujeto y su ideología como son los significados, los sentidos, las representaciones e intencionalidades como parte de la subjetividad que dan forma a las acciones de resistencia que asumen los individuos en acciones formales e informales frente a la pantalla. Por ejemplo, el ocultar código, el código sucio, o bien la resistencia a documentar y, si documentan lo hacen insuficientemente, no resuelven a tiempo, y si resuelven es con deficiencias, entre otras resistencias que mas adelante abordamos como conflictos del saber hacer entendido como poder.

[124] por ejemplo las normas de calidad internacionales como las del American Institute of Aeronautics and Astronautics (AIAA), American Society of Quality Control (ASQC), Electronics Industries Association (EIA), para el caso especifico del proceso del software son el Instituto de Ingenieros Eléctricos y Electrónicos (IIEE) y el Departamento de Defensa de Estados Unidos, para México son las normas de calidad del software como MOPROSOFT que es una derivación mejorada de los estándares IEEE y NBS de Estados Unidos

Las resistencia vista como respuesta de la acción individual o grupal están constreñidas estructuralmente por el conjunto de *herramientas* y *normas* que significan presión, coacción e imposiciones de *poder-saber* en el proceso de trabajo, la *resistencia como respuesta* se erige frente a estas *normas y herramientas* que comprenden una compleja red de metodologías, métricas y procesos que construyen una serie de rituales implícitos en las normatividades propuestas que unifican ciertas prácticas de poder, axiomas y reglas que moderan formas del *saber-hacer*. Si bien es cierto las *normas disciplinarias* representan una configuración de métodos, métricas y procedimientos técnicos que presionan como "única alternativa" del *saber-hacer* legitimado, que en lugar de imponerse a sus sometidos mantiene a éstos en un movimiento en torno del *saber-hacer*. Las *disciplinas normativas* en cumplimiento o validación de sus preceptos gestionan sus propios rituales y protocolos que se erigen como la "única vía creativa", diferencian, homogenizan, en suma normalizan a través de la disciplina y las normas (Foucault:1984:197). La disciplina (metodologías, métricas, etc.) se constituye como un tipo de poder que erige un conjunto de instrumentos, técnicas y procedimientos, mediante los cuales impone un poder disciplinario que *fabrica saberes* y les otorga una nueva modalidad de poder a los individuos, en la que cada cual recibe su propia individualidad vinculada a las cualidades y particularidades que le definen, encauzando sus conductas (Foucault:1984:197-198). El ejercicio de la disciplina supone un dispositivo que constriñe y oprime por el juego de la normas, por la designación de roles con poder (el líder frente al programador, éste frente al tester, éste frente al documentador, etc.). La disciplina es un poder que se ejerce haciéndose invisible, movilizando fuerzas ocultas, por ejemplo las jerarquías laborales que generan puntos de dominación que llegan a ser infinitesimales, induciéndose un efecto que Foucault denomina *observatorio*, y que nosotros definimos como "*fábrica de saberes*". El modelo de observatorio de Foucault lo retomamos bajo el entendido que las universidades generan un conjunto de *normas disciplinarias* a los programadores, *normas* que son a su vez difundidas y promovidas por otros agentes estructurales como los organismos civiles: la asociación mexicana de tecnologías de la información (AMITI), la cámara nacional de la industria de las tecnologías de la información (CANIETI). Los colectivos de la especialidad como software.net; amesol, prosfot, etc. entre otros que proponen el cumplimiento de las normas. Para Foucault (1994) las instituciones científicas constituyen un observatorio del saber que vincula la acción científica y técnica con la intervención sobre el

cuerpo social, se constituyen como un poder-saber que coacciona. Para Foucault (1994) la otra cara del poder es el *saber*, que actúa sobre el cuerpo social y el individual y, posee efectos disciplinarios sobre cada cuerpo individual como efectos de regulación.

Para el caso del proceso de trabajo del software a la medida, las *normas disciplinarias* pretenden regular e instituirse como dispositivos "científicos" creados por la ingeniería del software, sin embargo este cuerpo disciplinar de normas estructurales están atravesadas por una serie de dinámicas relacionales y prácticas cotidianas, no hay una apropiación de éstos protocolos normativos en la pantalla del programador, por el contrario en las rutinas habituales están presentes las resistencias y conflictos abiertos o soterrados, frente a las *normas y herramientas* que son atenuadas por el complejo de configuraciones subjetivas que intervienen en el proceso de trabajo, y se traducen en una *resistencia como respuesta* al poder que intenta imponerse. Para Foucault, el ejercicio del poder es una lucha política que se procura para perpetuar las relaciones de fuerza inscribiéndola en las instituciones, en los organismos, en el discurso, en el cuerpo teórico, científico y académico, en leyes, normas y procedimientos. La *resistencia como respuesta* de los programadores es una oposición a las estructuras de las *normas y herramientas* que se imponen como reglas científico-académicas que "observan" los contenidos que se transmiten a través de las voces oficiales (Universidades), cursos y discursos de la mejora de la calidad (MOPROSOFT) que se enfrentan a otras voces, de resistencia, de luchas contra las relaciones de poder. El caso mas evidente de *resistencia como respuesta* a la apropiación del *saber-hacer* es el modo de desarrollo de software libre, que se opone a los privilegios de patentar el conocimiento, se opone a coartar el flujo de información y del conocimiento que circula y funciona en el software libre, donde dichas resistencias son formas de *saber hacer* y *saber como poder*.

Las *normas y herramientas* forman parte de las estrategias actuales de la programación moderna y como resolución de un tipo-ideal a la aflicción del software producida en el contexto de la programación artesanal, caracterizada por un entorno textual, donde el programa en ejecución controlaba la información de la pantalla (no hay ventanas), y los únicos atributos disponibles eran la tinta y los trazos, no se compartían los recursos (monoprograma), se utilizaban rutinas de bajo nivel, sin problemas de operabilidad, el programador concentraba el saber hacer en si y para si, decidía que ocurre en cada momento de la ejecución del programa, no estaba socializado el conocimiento del *saber hacer*. La programación moderna

se constituye a medida que el hardware es mas potente y demanda software complejo, multitareas, que permita la multiprogramación, conformándose nuevas metodologías para desarrollar software, como es la programación orientada a objetos (POO), programación orientada a aspectos (POA), programación orientada a servicios (POS), entre otras metodologías que dan cuerpo a la programación moderna, que en síntesis dispone de una interfaz grafica dinámica y flexible. Se dispone de un conjunto de herramientas y parámetros de diseño amplios, no solo de distintos caracteres (letras, dibujos y figuras) sino también de diversos atributos (tipos, tamaños y colores), así como un conjunto de librerías que contienen iconos e imágenes prediseñados (liberarías, bibliotecas y repositorios). Posibilidades de multiprogramación y se dispone de nuevas posibilidades graficas (distintas ventanas). La pantalla se convierte en otro dispositivo de entrada de datos a través del mouse (además del teclado). El programador dispone de un conjunto de soluciones preconstruidas (clases y objetos), así como de bibliotecas con rutinas de código disponible, donde este código no necesariamente es construido por el programador ya que éste puede "heredar" código almacenado. Es decir, el programador cuenta con una serie de objetos, clases, librerías y herencias de código, reutilizables. El programador cuenta con mecanismos que le permiten re-factorizar las rutinas de código, es posible revisar y documentar directamente el algoritmo en otros repositorios y en tiempo real sin que ello afecte el proceso de trabajo. Sin embargo, la aflicción del software continúa presente.

La programación moderna supone mayor grado de interacción en el proceso de trabajo, es dinámica y re-configurable, lo cual implica que los canales de comunicación e información entre los actores es flexible, cooperativo y colaborativo, sin embargo, aún con estas metodologías de quinta generación el costo total del proyecto es variable, la planificación es larga y tediosa, el cumplimiento de tiempos es difícil de considerar. Se requiere de un conjunto armónico de programadores, una acumulación de experiencias, habilidades y destrezas técnicas y de comunicación (oral y escrita). Éstas metodologías son criticadas porque continúa presente la incertidumbre y la contingencia en la creación lógica de las rutinas o bucles de algoritmos encapsulados en clases, esto es así porque las cualidades de unos algoritmos pueden no ser necesariamente útiles en otros proyectos, lo cual implica que el programador conozca los fines y la lógica del código contenido y acumulado en las clases encapsuladas o en las bibliotecas de la programación orientada a objetos y/o aspectos; en otras

palabras, el software no es estático, no es que se construyan "cápsulas de código" y estos se hereden tal cual, porque están especializados para determinadas tareas o rutinas. Aún con este conjunto de objetos que encapsulan la información en atributos y métodos (configuración del código) la programación orientada a objetos no resuelve del todo los problemas de calidad, eficiencia, entrega a tiempo, contingencias e incertidumbres del proceso de trabajo. Es decir, aún en el contexto de la programación moderna, persiste la imposibilidad de software libre de fallos y cero errores, así como la nula posibilidad de garantizar un software eficiente.

Hacia finales de los noventa las *normas disciplinarias* señaladas (programación orientada a objetos, aspectos y servicios) cobran relevancia con relativo éxito entre las fábricas de software[125], donde la división del trabajo está caracterizada por el hecho que los clientes son empresas diseñadoras de los objetos a desarrollar (los objetos sustituyen a los módulos), las fabricas de software solo maquilan el código, sin embargo dicha metodología (POO) es criticada fuertemente (no sólo por sus adversarios promotores de la metodología orientada a aspectos) por los pequeños y medianos empresarios quienes señalan que se imponen *normas* burocráticas para definir y medir el proceso de desarrollo, incorporando *herramientas* de medición y control de calidad (CMM, IEEE, ISO, etc.) costosas y que poco hacen por la eficiencia y la calidad del software desarrollado a la medida.

Otra crítica es que la POO está orientada a dividir el proceso de trabajo mediante el encapsulamiento de información (código almacenado) y métodos del saber-hacer en objetos que expliquen y detallen los atributos denominados clase (código despejado), donde cada clase describe las propiedades de los algoritmos desarrollados, transformando estas propiedad en familias que a su vez son un conjunto ordenado de atributos y métodos de rutinas o bucles de código (código encapsulado) que se pueden heredar (conocimiento tácito). En las entrevistas con lideres y gerentes explicaban que la POO "es el mejor camino", sin embargo también declaraban que es burocrática, rígida y que demanda elevados recursos económicos, de tiempo, y equipos de trabajo, jerarquizados y rígidos; pero en un contexto de las empresas mexicanas de software con escaso apoyo financiero y donde es costoso implementar dichos procesos, ya que 83% de las empresas desarrolladoras de software son pymes, 87% tiene

[125] Para el caso de Japón, donde las fabricas de software datan desde los ochentas véase Reporte sobre el estado del mercado de software en Japón, 1994, japan economic intiute report disponible en www.gwjapan.com/ftp/pub/policy/jei/1994/a-series/040194.txt, Otros ejemplos en Bracho F. Díaz, Arnoldo 1995, La cadena virtual de producción: hacia una fundamentación de la ingeniería. Soluciones Avanzadas.

menos de diez empleados, 86% de los trabajadores poseen menos de 15 años de haberse graduado de la universidad;

Las metodologías orientadas a objetos (POO) fueron creadas para operar mediante un conjunto de *normas* y *herramientas* que se centran en el proceso de trabajo, pretenden codificar el *saber-hacer* mediante una división del trabajo que está centrado en los objetos, que representan los conceptos o requerimientos del cliente y, por otro lado, las clases donde se intentan hacer explícito los atributos y propiedades de las rutinas y bucles de código, señalando las particularidades que se repiten entre clases (herencias) y éstas rutinas heredables, reutilizarlas en otros proyectos. Es decir, acumular una serie de rutinas similares que son re-factorizables (herencia y polimorfismo); procesos que requieren de recursos, una planificación burocrática; *estándares y normas* que sean observadas por los programadores, así como una gerencia que coaccione el cumplimiento de estas *normas y estándare*. Sin embargo, como señala Foucault, paralelamente a la estructuración de éstas metodologías, la *resistencia como respuesta* al poder presiona y estructura las acciones individuales y sociales entre los actores que intervienen en el proceso de trabajo, respuesta que se configura paralela y sincrónicamente, no opuesta sino embebida y coercitiva orientada a una lucha por el *saber-hacer* y el *saber como poder*. Se configuran una serie de métodos y procedimientos no centrados en el proceso, sino en los actores que intervienen, se conviene en que no son las metodologías burocráticas las que sustituyen las habilidades y destrezas de los que intervienen en el ciclo de desarrollo del programa, sino que hay un consentimiento implícito, donde no es la planificación rígida del saber-hacer en objetos y clases donde reside el saber resolver, sino que es la composición lógica de las rutinas y textos sígnicos donde reside la eficiencia del saber hacer; tan importante como la simbolización de los requerimientos en un diseño como significativo es la transparencia con que se crea el texto sígnico (código transparente); que los algoritmos solucionen de la mejor manera el requerimiento del cliente, lo substancial es el proceso cognitivo del saber-hacer ya que los proyectos de software a la medida son adaptativos, no estáticos, evolucionan y junto con ellos las habilidades de quien elabora los textos sígnicos. Es decir, la *resistencia como respuesta* intenta defender la *fluidez cognitiva del aprendizaje* en el proceso de trabajo, ya que los programadores perciben y no son los procesos rígidos de CMMI, IEEE quienes proveen las habilidades para *saber resolver*. En otras palabras, las ingenierías del software intentan fragmentar los espacios del proceso de

trabajo a través de un conjunto de *herramientas* y *normas* ya citadas, donde la *resistencia como respuesta* de los actores que intervienen en el proceso de trabajo configuran consensos y arreglos sociales e individuales que se centran en saber resolver.

El proceso de trabajo en el software a la medida se constituye como espacios yuxtapuestos, con fronteras difusas, no precisas. Sin embargo, podemos señalar tres espacios: El primero está dado por conceptualiziación de las necesidades del cliente (lista de requerimientos) y la formalización de dichos requerimientos en una lista que puede o no ser firmada por el cliente. El segundo espacio es el procesamiento de datos, revisión de la calidad de las rutinas de algoritmos (tester). El tercer espacio es el de la implementación del software desarrollado (aceptación o rechazo del software por parte del cliente). Las ingenierías del software a través del conjunto de *normas disciplinares* comprendidas por los diferentes *herramientas procedimientales* se enfocan en separar el espacio del procesamiento de datos con respecto a los otros dos espacios. Consideran que lo importante es la conceptualización y formalización de los requerimientos en un diseño consistente, planificado y modularizado. De tal forma, que se documenten las tareas que se van a desarrollar, redactar las dependencias que existen entre las tareas, por medio de ello se consigue estimar las horas-esfuerzo por tareas y se asigna un presupuesto. Una vez que se verifica que se haya cumplido con las tareas asignadas por parte de los programadores, se procede a implementar el software ya sea en las instalaciones del cliente o con el usuario final, donde se válida el cumplimiento de los requerimientos previamente firmados. Esta división espacial del trabajo, es ampliamente criticado (como ya lo señalamos en el apartado anterior) como burocrático, rígido, requiere tiempo para la planificación, se incrementan los costos y demanda grandes equipos de trabajo. También es arriesgado predecir los resultados, cumplimiento eficaz de los requerimientos, que esté libre de errores (bugs) ó, que se entregue a tiempo. La crítica más severa proviene del hecho de considerar el procesamiento de datos como rutinizable del tipo "maquila de software", al señalar que una vez que esta el diseño y las tareas asignadas a los programadores, es predecible los resultados, ya que los algoritmos son "fáciles de construir". Las ingenierías del software consideran que 80% del costo está en el diseño (planificación) y 20% en la construcción. Sin embargo, consideramos que las ingenierías del software insisten en un error al comparar el proceso de trabajo cognitivo con el trabajo manufacturero.

Esta pretendida división espacial del trabajo hace a un lado la contingencia e incertidumbres presentes en el software a la medida, se empeña en asignar un contrato de requerimientos rígido e inflexible que inhibe el cambio de requerimientos aún a costa de que el cliente necesite de otros requerimientos. Niega la maleabilidad del software a la medida, la eventualidad y el contexto de los requerimientos, deja a un lado la fluctuación en los requerimientos, en ocasiones el cliente no esta seguro de lo que necesita, no se reconoce que el cambio es la norma en el software a la medida. Los cambios en los requerimientos pueden no ser atribuibles a los clientes, por ejemplo cambios en políticas impositivas (hacienda, exportaciones, importaciones, etc.), cambios en los productos, en organización administrativa o productiva del cliente, de tal forma que los requerimientos no son estáticos, no son estables, no son previsibles del todo. Por el contrario la *respuesta-resistencia* por parte de los actores individuales consideran los espacios productivos como uno sólo, yuxtapuestos, sin divisiones e intercalados en el periodo de tiempo, donde los actores que intervienen lo hacen de una forma fluida, con interacciones diarias, cotidianas, generando resultados momentáneos de forma continua, en versión pequeñas pero operables por el cliente, es decir el cliente está en el centro del proceso de trabajo, no como las ingenierías del software que ubican al cliente hacia el principio y el final del proceso, por el contrario en la *fluidez cognitiva del aprendizaje* el cliente es parte central en la revisión del cumplimiento de requisitos, adaptabilidad que permite al cliente, no solo verificar el cumplimiento de requerimientos, sino el modificar la interfaz grafica, anexar, ampliar o cambiar los requisitos. Esta *fluidez cognitiva del aprendizaje* permite considerar al cliente en el centro de cada tarea-solución-resultado, jugando un papel activo y no pasivo. Esta *respuesta-resistencia* creativa, dinámica y evolutiva se centra en la habilidad y destrezas de los actores en resolver a tiempo, en una serie de *consensos* y *arreglos informales* e *individuales* valorados por todos, y no en el diseño y la tarea asignada (relaciones de poder que constriñen al proceso de trabajo, emanadas de las *normas disciplinares*); lo central es el cliente y el texto sígnico creado a través de un *proceso de trabajo fluido,* ágil, dinámico y práctico entre los individuos que intervienen. De tal forma, que el proceso de trabajo fluido se caracteriza por se adaptable, flexible, púgil embebido por una serie de prácticas sociales de aprendizajes que construyen una *socialización del conocimiento* entre los actores que participan en el proceso de trabajo.

Las Ingenierías del Software a través de las *normas y herramientas* consideran a los programadores como "componentes reemplazables", en cambio en el *proceso de trabajo fluido* los actores que participan en el proceso de trabajo concentran las habilidades, destrezas y conocimientos necesarios del saber hacer y el saber como poder. Son los programadores quienes reflexionan la "mejor manera de resolver" el requisito planteado por el cliente (arreglos sociales, arreglárselas, lógica de programador), son los programadores quienes examinan el desempeño del código fuente, es decir no es el diseño el factor clave, sino el desarrollo de texto sígnico el tema central. El objetivo fundamental no es cumplir con tareas asignadas, sino con los requerimientos del cliente, para lo cual es trascendental incorporar los cambios que el cliente considere necesarios, pero en el proceso mismo de trabajo, no hacia el final del proceso, como propone la ingeniería del software. De esta forma el cliente interviene en el proceso de trabajo, reduciéndose así la incertidumbre en el cumplimiento de requerimientos y las contingencias en el cumplimiento de tiempos y costos. En las interacciones sociales objetivas y subjetivas que configuran la *fluidez cognitiva del aprendizaje* gestiona la transformación de la información en conocimiento y aprendizaje; por tanto la comunicación ágil (oportuna) entre actores es clave para la solución de problemas, así como garantizar que la información sea fluida para responder a las contingencias.

La *respuesta-resistencia* centrada en los actores no es reactiva, es creativa; no es rebeldía frente a las relaciones de poder de las estructuras, es un poder-saber que se hace presente, que se erige paralelamente, no se subordina a las estructuras no se somete, sino que los actores reconfiguran sus acciones individuales y sociales, sin dejar a un lado la influencia de las estructuras que constriñen y determinan parte de la dinámica subjetiva de los individuos. Es esta condición subjetiva la que esta atrás del conjunto de resistencias en el proceso de trabajo cognitivo que las ingenierías del software no han sabido identificar.

Consideramos que las resistencias, consensos y arreglos individuales, sociales, formales e informales se generan paralelamente a la división espacial del trabajo que promueven las ingenierías del software. No sólo se promueve la división entre concepción y ejecución al estilo de Taylor y Ford, sino que hacen a un lado al tercer actor en el proceso de trabajo: el cliente, a quien lo considera como "proveedor de requerimientos", hacen a un lado el conjunto de subjetividades del cliente.

Las ingenierías del software no consideran los procesos que operan sobre las habilidades y destrezas que se requiere en la reflexividad cognitiva del texto sígnico (acumulación de experiencias, sentido de la responsabilidad, intencionalidad al desarrollar el código, etc.), es decir, las ingenierías del software se basan en elementos racionales como la planificación rígida y burocrática que busca asignar tareas que se cumplan en tiempo y forma, visión por demás sesgada basada en un individualismo metodológico que hace a un lado las *practicas subjetivas* y las dinámicas relacionales implícitas en el *saber hacer* y el *saber como poder* del programador. La Ingeniería del Software no considera el *sentido subjetivo del trabajo como aprendizaje*, desconoce la constelaciones de configuraciones subjetivas como las percepciones, representaciones, las emociones frente al trabajo y los sentimientos y las emociones, -como el humor característico de los hacker- el coraje, envidia, rencor, etc. que juegan un papel fundamental en el proceso de trabajo cognitivo del programador, influyen en el *como resolvió* y *porque resolvió* determinado texto sígnico o simplemente porque no resolvió la tarea asignada.

Los algoritmos creados, junto con la documentación y narraciones del código elaborado puede concebirse como *textos sígnicos* que comprenden aquellas instrucciones de algoritmos, precedidos a su vez de un compuesto de símbolos nomotéticos que esta compuesto por los lenguajes de programación; definiendo dicha composición como una *reflexión dialógica* como un proceso "sígnico" que describen y representan los símbolos-ideas del cliente en algoritmos formulados lógicamente por el programador. Para Bajtín (2005, 1982)[126] cada texto, visto como enunciado es algo único, individual e irrepetible, en lo cual consiste su sentido. Así el programador tiene ante sí, dos tipos de símbolos, el de las ideas-requerimientos del cliente plasmadas en un diseño con instrucciones y, por otro lado, el conjunto de signos que son los lenguajes de programación, un tercer tipo de símbolos subjetivos son los sentidos morales y estéticos, las representaciones, además de las estructuras que le constriñen como son las *normas y herramientas* de las Ingenierías del software.

La dificultad para medir la eficacia en el proceso de trabajo del software radica en su propia naturaleza que podemos definir como *"un proceso de trabajo cognitivo donde la materia prima son diferentes tipos de símbolos: símbolos-ideas del cliente (requisitos), símbolos-ideas de éstos requerimientos (formalización de los requerimientos), símbolos-ideas*

[126] Bajtín, M, (2005, 1982) Estética de la creación verbal. Editorial Siglo XXI, 2005. México. Pp. 296-297.

del programador (procesamiento de los requerimientos en algoritmos) y el resultado es un conjunto de signos que se crearon a mano a partir de interacciones". El resultado es un conjunto de reinterpretaciones de los requerimientos del cliente en un conjunto de *textos sígnicos* (algoritmos creados a partir de los lenguajes de programación). Es la composición de éstos *textos sígnicos* (algoritmos lógicos) que implican una serie de acciones individuales y sociales, como es la resistencia, los conflictos, el consenso, los arreglos formales e informales en el proceso de trabajo. Este tipo de acciones son a su vez constreñidos por las estructuras erigidas por las ingenierías del software (*herramientas* y *normas*), y por las condiciones subjetivas que presionan la acción individual de los actores que intervienen en el proceso de trabajo (incluye al cliente y al usuario final). Estas acciones individuales y sociales están embebidas por un conjunto de *constelación de dimensiones subjetivas,* en espacios claroscuros, opacos, donde podemos distinguir cuando menos tres tipos de subjetividades: subjetividad creativa, subjetividad sígnica y subjetividad que se objetiva.

La *subjetividad creativa*, tiene que ver con la *construcción de decisiones* de los individuos; con limites cognitivos; destrezas y habilidades de abstracción de o los individuos; *intencionalidad* de los actores que interaccionan; practicas cotidianas al interior del proceso de trabajo, como pueden ser arreglos formales e informales, consensos y resistencias, boicot y luchas por el conocimiento o espacios de poder. La *subjetividad signica*[127] estaría representada por los textos algorítmicos de los programadores, donde dichos algoritmos representan condiciones de abstracción, esfuerzo cognitivo propio, así como un alto grado de *interacción de pensamientos y movimientos reflexivos* que realiza el programador arbitrariamente, es decir en el proceso de trabajo no existe una "sólo camino" sino que el programador se abstrae, especula y refleja su pensamiento en un *texto sígnico*, precedido de un conjunto de habilidades, destrezas personales de conocimiento, flujo de información con otros, ya sea dentro o fuera del espacio laboral. La tercera subjetividad hace referencia a que se objetiva el programa fuera del espacio productivo, por fuera de la empresa (Industria de la piratería del

[127] El algoritmo resulta ser un texto de símbolos nomotéticos que a su vez está compuesto de signos que provienen del lenguaje de programación que se haya utilizado (existen mas de 2000 lenguajes); definiremos dicha texto algorítmico como una reflexión dialógica que se objetiva en un proceso "signico" que describen y representan los símbolos-ideas del cliente en algoritmos formulados lógicamente por el programador. Idea reformulada a partir Bajtín (2005, 1982) donde señala que cada texto, visto como enunciado es algo único, individual e irrepetible, en lo cual consiste su sentido. Donde el gesto natural en la representación efectuada por un actor, adquiere una importancia signica, por su carácter arbitrario, convencional y sometido a la intención del papel. Bajtín, M, (2005, 1982) Estética de la creación verbal. Editorial Siglo XXI. 2005. México. Pp. 296-297.

software), se rompe con el concepto de economías de escala.. La naturaleza de la *subjetividad creativa*, se recrea y establece a partir del primer contacto (entrevista cliente-analista o programador) donde el cliente solicita el desarrollo de un software a la medida, esta relación esta inmersa en un contexto subjetivo que puede derivar en incertidumbre, porque las interpretaciones de las necesidades del cliente pueden ser próximas o asertivas al problema, pero también pueden estar alejadas de las necesidades reales del problema. La naturaleza de la *subjetividad sígnica*, reside en la *configuración reflexiva* del conjunto de signos que componen los lenguajes de programación (existen en la actualidad mas de dos mil lenguajes de programación) en el que se decide desarrollar el programa informático sea o no lenguaje óptimo; o bien que, el conjunto de configuraciones algorítmicas –que hemos denominado *textos sígnicos*- posean o no cierto grado de cohesión lógica en su estructura interna (errores de configuración y problemas de interoperabilidad entre módulos, falta de experiencia o habilidades del programador, etc.). Podemos señalar que el proceso de trabajo es un conjunto subjetivo de "*crear a mano símbolos a través de signos*", proceso que está embebido en una *constelación de dimensiones subjetivas* que influyen en el saber-hacer del programador. Los códigos subjetivos internos que podemos distinguir son los valores, emociones, sentimientos estéticos, conocimientos que configuran las *constelaciones de dimensiones subjetivas* que se objetivan en el software que el cliente posee en su pantalla. *Un proceso subjetivo que se objetiva* independiente al proceso de trabajo. Se caracteriza por objetivarse en la pantalla, ya sea en la del cliente/usuario final o en cualquier otra pantalla independiente del cliente; el programa de software a la media posee la característica de objetivarse **n** número de veces, después de haber salido del proceso de trabajo.

La creación de textos sígnicos (algoritmos) es el aspecto lógico-formal que está atravesado por distintos grados de incertidumbre (dependerá de la complejidad del algoritmo desarrollado). La incertidumbre en el texto sígnico está en relación directa con las condiciones objetivas y subjetivas que se configuran en proceso de trabajo. Las objetivas son el grado de conocimiento del lenguaje de programación que se emplea; experiencia acumulada en cantidad de programas desarrolladas; comprensión metódica del problema planteado y "saber resolver" correctamente los requerimientos (símbolos) del cliente en otros símbolos a través del lenguaje de programación (textos sígnicos). En este proceso de re-simbolización a través de un flujo de aprendizajes (fluidez de información y conocimientos en los saber hacer),

interaccionan una serie de *dinámicas relaciónales* que se instituyen a través de juegos, consensos, negociaciones y conflictos, resistencias y boicot unos soterrados, otros ocultos; así como una serie de *prácticas subjetivas* como el ocultar el proceso de saber resolver (código oculto), no documentar el texto sígnico (código ciego); no resolver en los tiempos y requerimientos acordados (caja negra cognitiva) las habilidades y destrezas cognitivas del *saber resolver* el problema planteado, con el lenguaje de programación que se pretenda utilizar, cumpliendo las condiciones tecnológicas del proyecto. Es decir, son los tiempos y pensamientos como los movimientos reflexivos para resolver el requerimiento en un código lógico y coherente (texto sígnico).

En el proceso de trabajo del software a la medida, consideramos que coexisten tres procesos simbólicos de poder yuxtapuestos no definidos, amorfos y complementarios. 1) conceptualización de las *ideas-símbolos* de los requerimientos del cliente y la aceptación o rechazo del software desarrollado, 2) Formalización de los requerimientos en símbolos, 3) procesamiento de los símbolos (diseño) a través de signos (lenguajes de programación), reflejando *textos sígnicos* que constituyen las rutinas de código. Contrariamente a lo que insisten las ingenierías del software a través de las *herramientas procedimentales* de dividir el proceso de trabajo, estos espacios de poder-saber no están separados, por el contrario están yuxtapuestos, fluidos horizontalmente en un *continnum cognitivo.* Como ya señalamos que la aparente separación entre diseño y ejecución no es tal, al estilo taylorista, donde la gerencia detentaba el saber-poder, como una forma de "restarle poder al trabajador"; en este proceso de trabajo se yuxtaponen espacios de poder. Cuando se definen y especifican los requisitos, tareas o rutinas que deberá ejecutar el sistema informático por parte del administrador de sistema, el arquitecto, el gerente el programador experto, -dependiendo de la organización de la empresa- se conforman las primeras tareas a resolver; cuando se interacciona con los programadores para plantearles el diseño que integra al software que se va ha desarrollar acontecen los primeras interacciones de saber-poder que se llevan a cabo a través de una serie de juegos específicos entre aquellos (el diseñador, líder de proyecto o el administrador de la configuración) y los programadores (Desarrolladores, tester, documentador) en la concreción de la "colección de actividades" que integrarán el Software solicitado (Hammer y Champy, 1994). Para Foucault (1993) en la "microfísica del poder" el poder no es monolítico, consiste en una serie de juegos específicos en el que todos los participantes actúan en diferentes niveles

y con distintas intenciones e intensidades. No se puede pensar en términos del poder al estilo de Taylor, en el que por un lado está la administración o los gerentes con un poder disciplinario, señalando normas y procesos que deben se cumplidas. Para Foucault el poder nunca es enteramente controlado por alguien o algunos, sino que en cada situación, contexto y escenarios se reconfiguran nuevos recursos inesperados juegos de poder.

Consideramos que en las interacciones en el proceso de trabajo del Software a la medida se suceden relaciones de poder intensas y tensas, cuando el agente (administrador del sistema, arquitecto de software, líder de proyecto, programador experto, dependerá del grado de sofisticación en la organización de la empresa) interacciona con el programador con un cierto fin, por ejemplo que se comprenda y aprehenda el problema a resolver, problema que es planteado a partir de la lista de requisitos, esta interacción implica una serie de relaciones de saber y poder. No al estilo de imposición como señalan las Ingenierías del Software, sino que acontece un dialogo, un consenso, arreglos formales e informales en la determinación de tiempos y tareas a resolver en los módulos o tareas asignadas. Donde el resolver los problemas que significa "arreglárselas", un "arreglo social", una "lógica de programador" que resuelva los módulos o tareas asignadas, implica una serie de interacciones hacia adentro del proceso de trabajo (cara a cara) como hacia fuera del proceso de trabajo pero desde el lugar de trabajo (pantalla a pantalla) y fuera del proceso de trabajo (tiempos de vida personal). Es decir, el proceso de trabajo supone una yuxtaposición de los tiempos de producción y tiempos de vida del programador, interacciones que se llevarán mas allá del espacio productivo, dependiendo de sus intereses, motivaciones, intenciones, sentimientos frente al trabajo y aspiraciones personales, etc. Es decir, la resistencia entendida como un conjunto de acciones individuales y de grupo son creativas y reactivas, dinámicas con rigideces, resolución de problemas que no necesariamente es lo mas eficiente, se conservan las formas, contenidos y transformaciones del saber-hacer, pero no se satisface "la mejor forma resolver", sólo se "resuelve", entendido estas contradicciones y luchas de poder como un proceso, donde la circulación y transformación del saber transcurre como paradigma de poder, bajo el contexto que resolver un problema asignado implica manifestar y exponer el saber como poder. Ahora bien, el saber resolver y arreglárselas para resolver constituye la base de la circulación del saber como poder, lo cual conduce a una acumulación del saber tan importante como la acumulación y circulación del capital que señala Marx (Capital Tomo 1). Foucault (Vc.34-35) añade que las

relaciones entre saber y poder no deben pensarse como si el saber oscureciera las relaciones de poder y por tanto éstas debieran encubrirse por su carácter represivo, por el contrario Foucault, invita a que se admita que el poder genera saber, que ambos se mezclan entre si "no hay relación de poder sin constitución correlativa de un campo de saber, ni de saber que no suponga y no constituya, al mismo tiempo unas relaciones de poder".

Las relaciones de poder entre los programadores y de éstos con los gerentes y clientes es temporal, el poder no es fijo, sólo transita entre los individuos, pero el saber-hacer se acumula en términos de conocimiento y, ello se traduce en poder-saber por quienes participan y resuelven en un proyecto, ya que serán quienes posean una serie de habilidades cognitivas, destrezas procedimentales que resuelvan el problema planteado, lo cual implica poner en juego las subjetividades y acciones individuales y colectivas en las interacciones de acumulación de poder-saber y relaciones saber como poder entre los que participan en el proceso de trabajo. Por tanto el saber y el poder en el proceso de trabajo del software no es causal ni fugaz o, de suma cero, como señala Foucault; por el contrario los individuos-programadores reconfiguran su "posición" de poder en cada proceso de interacción con los otros, pero es frente a los otros que se reposicionan en términos de acumulación del poder. Por ejemplo, para que un programador senior se le reconozca como tal, no es por la antigüedad en el puesto de trabajo, no son las credenciales formales de estudio, es la habilidad para solucionar un problema, es la destreza con la cual propone soluciones, es la abstracción cognitiva del problema planteado. Es el conjunto de interacciones con los otros, ya sea al interior del proceso de trabajo como fuera de éste donde acumula saber-poder.

En las interacciones en los espacios de producción, ya sea entre el cliente y el líder de proyecto, éste y programador, entre programadores y de éstos con el cliente, acontecen una serie de juegos de poder-saber que se anulan o disipan posteriormente al interacción que le dio origen, como plantea Foucault; sin embargo suponemos que si bien es cierta una serie de juegos y relaciones de poder entre los programadores no resulta una "suma cero" para los "jugadores" que participan en el proceso de trabajo: consideramos que hay límites en la propuesta de Foucault, en términos del saber como transitoria hacia el poder. Burawoy ([1979], 1989), por el contrario, señala que en las relaciones de producción capitalistas intervienen una serie de acciones individuales, gerenciales, patronales que pueden monopolizar el poder, que mantienen el poder no necesariamente centrado en las paredes o

limites de las y para las interacciones como señalaba Foucault; sino un poder que se crea, revalora y reproduce en el proceso mismo del trabajo a partir no solo de las interacciones, sino que dichas interacciones se circunscriben a un juego e inter-juego que en el caso del proceso de trabajo del software hemos denominado *juego por resolver un problema a tiempo*. Dicha denominación no es fortuita, se sujeta a la presión del tiempo asignado al modulo; así como a las contingencias inherentes a la eficiencia o no de la solución propuesta; así como acciones subjetivas de los individuos como puede ser intra-subjetividades como las expectativas, angustias, emociones y sentimientos involucrados en la respuesta; o bien las inter-subjetividades situadas en la interacción con otros para resolver a tiempo. Ambos planos de la subjetividad de los individuos permea al proceso de trabajo.

Burawoy, señala que el monopolio del poder, refleja determinados conocimientos tácitos y explícitos como: habilidades, destrezas, experiencias, emociones, sentimientos, formalidades en el proceso, etc. (Burawoy, [1979], 1989:102 ss). Conocimientos que refuerzan el saber-poder de quien monopoliza (el taylorismo es el mejor ejemplo). Si hay quien monopoliza, está claro que existe aquel que se le arrebata el poder, es decir, se rompe con el planteamiento de Foucault quien señalaba que el poder no se arrebata, no se quita, sino que se generaba y culminaba en las interacciones de los individuos. La noción de monopolio del poder señalado por Burawoy, para el caso de las interacciones entre programadores de software, es importante porque nos permitirá conocer cuales son y como se configuran los juegos por el poder y el desplazamiento del conflicto entre quien intenta monopolizarlo y los que detentan el saber-hacer en la programación.

En el proceso de trabajo, paralelamente a *los juegos de resolver a tiempo* y las dimensiones subjetivas que le atañen, consideramos que se verifican una serie de dinámicas y prácticas conflictivas, de confrontaciones abiertas unas, soterradas otras; como el *ocultar código;* otras institucionalizadas, manifiestas, como *no documentar el código*; y otras mas de índole subjetivo como *el sabotaje*, que se traduciría en la renuncia, en no resolver a tiempo, en no cumplir con tiempos.

Consideramos que estos enfrentamientos, pugnas, conflictos y resistencias no se suceden sólo por las luchas por el monopolio del poder sino que configuran como respuesta al ejercicio del poder a través de una serie una constelación de prácticas subjetivas, como los arreglos sociales, el arreglárselas, que se constituyen como negociaciones, consensos y juegos

no por el poder, sino como "la otra cara del consenso" por el poder, expresada a través de una serie de interacciones cotidianas del proceso de trabajo como *un sentido subjetivo del trabajo como aprendizaje* que comprende una serie de juegos, arreglos y consensos que se configuran a partir de una serie de intra-subjetividades, como *satisfacciones relativas* (interpretación de intereses, intenciones o motivaciones que conduce a los individuos a intensificar o reducir las interacciones en las relaciones de poder). El conjunto de *satisfacciones relativas* estarían dadas por: familiarización con los conocimientos utilizados; aprendizaje de nuevos conocimientos (dentro o fuera del proceso de trabajo); habituación a los sistemas de producción en la empresa, etc.; la ejecución o consideración intra-subjetiva de dichas satisfacciones relativas por parte de los programadores (en algún grado no cuantificable) bastaría para que el individuo señalara o considerará la existencia de un aprendizaje, o lo contrario, la insatisfacción por el aprendizaje, llevará al individuo a reducir la relaciones de poder en el proceso de trabajo o bien hasta renunciar a la empresa. La satisfacción relativa, o insatisfacción por el aprendizaje forma parte no sólo del juego de resolver a tiempo (Burawoy) también se integra a las relaciones de conflicto en el trabajo (Eduwards).

El conjunto de *satisfacciones e insatisfacciones relativas* en el aprendizaje se traducen en una forma de juegos de poder, con tensiones, pugnas y conflictos, que denominaremos por el momento como "*juegos por resolver a tiempo*"[128], que podríamos definirlo como una acción cognitiva por resolver y solucionar un problema dado, lo cual no quiere decir que la solución sea la mas eficiente y la variable tiempo es una condición del proceso de trabajo, medido el tiempo en horas-hombre concertadas -y en algunos casos impuestas- entre el administrador del proyecto y los programadores. El *juego de resolver a tiempo*, implica varios temas, uno de ellos es el conflicto -que no abordó del todo Burawoy- como las reglas informales que constituyen el *juego de resolver a tiempo*; juego que se constriñe a una serie de complejidades, que tienen que ver con el hecho que los programadores formalizan cual es el procedimiento abstracto-algorítmico para solucionar un problema, esta formalización se concreta en el *texto sígnico* que debe ser consistente en la lógica hacia el interior del algoritmo (que no haya inconsistencias) y hacia el resto de las soluciones que se presenten (compatibilidad con los otros módulos); tanto la ausencia de inconsistencias e incompatibilidades representan una

[128] NO quiere decir, que las satisfacciones relativas sean el conjunto de las interacciones que convergen en el proceso de trabajo del software, sin embargo nos concentraremos en este punto, ya que el anterior apartado se lo dedicamos a la interacción cliente-líder de proyecto.

condiciones del *significado de resolver a tiempo*, que no es otra situación que cumplir con el desarrollo de los algoritmos que integran los módulos y entregarlos en las fechas señaladas en el plan de trabajo del proyecto.

Otra condición del *sentido subjetivo del trabajo como aprendizaje* es aquella que tiene que ver con el *que* y el *como* resolver el problema planteado, ello no significa *no* conocimiento formal del proceso, sino una problemática del tipo cognitivo, es este proceso reflexivo que esta influenciado por la subjetividad, entre el *que* y el *como* resolver el problema, que no es otra cosa que una amplia gama de posibilidades, ya que no existe "un único mejor camino", proceso que no lo resuelve por si mismo como individuo, sino que la solución implica una serie de *reglas no formales en el juego de resolver* que están embebidas en dos planos de la inter-subjetividad en la interacción con otros, en dicha interacción entran en juego experiencias, habilidades, destrezas, etc. y el plano intra-subjetivo que hace alusión al hecho de internalizar, interiorizar, aprehender el problema a resolver. En 1939 Roehtisberger y Dickson (citador por Burawoy, [1979], 1989:105), señalaban que "los trabajadores tiene sus propias reglas y su propia lógica, que en la mayoría de los casos están en contradicción con las que les han sido impuestas".

Otra condición del *sentido subjetivo del trabajo como aprendizaje* tiene que ver la nulidad de "un único mejor camino" para resolver los problemas planteados, implícitamente significa una *igualdad cognitiva* entre aquellos que participan en el proceso de trabajo, donde los gerentes y lideres, clientes y programadores; colaboran por consenso, bajo arreglos informales para resolver que toman la forma de una serie de juegos, que parten de una regla no escrita: *igualdad cognitiva*; igualdad que no significa que el diseñador conozca el *como y porque* se va resolver un problema, sino *igualdad cognitiva* en términos de que el diseñador conoce el *que* resolver, pero el *como y el porque* es parte de los programadores, parte sustancial del desarrollo, donde los juegos de saber-poder tienen un contexto de indeterminación de las normas aplicables. Al respecto Ditton, señala que las reglas al interior del proceso de trabajo se fortalecen con la participación activa de los estratos menores de la gerencia e incentivando la presentación de juegos (citado por Burawoy, [1979], 1989:106). Burawoy concluye y demuestra en su amplia obra, que cuando se llevan a cabo los juegos éstos no son independientes y no se conspira contra la empresa; por el contrario se lleva a cabo un proceso complejo entre gerentes y trabajadores, para que cada uno de los "jugadores"

desarrolle habilidades, destrezas y conocimientos necesarios para "arreglárselas" en el cumplimiento de las exigencias productivas (numero de piezas, cantidad de operaciones técnicas, etc.). Por ejemplo en *el juego de resolver a tiempo* un requerimiento exige capacidad de abstracción y habilidad cognitiva para redactar un texto sígnico que resuelva un problema; es decir los programadores en el desarrollo de software deben solucionar el problema, en interacción con los otros, pero los programadores deben "*arreglárselas*" para escribir las líneas de código, solucionar el problema; es verdad que pueden acudir con un programador mas hábil, con el líder del proyecto, con librerías o ayudas en comunidades virtuales en Internet, etc., pero hacia el final, son los programadores quienes redactan los algoritmos. Los conceptos de "arreglárselas" y "arreglos sociales" permitirá explicar parte del juego informal de resolver a tiempo.

El *arreglárselas* para resolver un problema tiene que ver con el uso de las *herramientas procedimentales* que definimos al principio, con los lenguajes de programación, con el uso correcto de las plataformas tecnológicas, e incluso una negociación densa entre el líder o el diseñador del módulo con el programador, bajo la posibilidad de que el planteamiento del diseñador no siempre es el más idóneo. Burawoy ([1979], 1989:106) demuestra que no basta una orientación instrumental para llevar a cabo el *arreglárselas* en el proceso de trabajo, sino que existe todo un conjunto de reglas informales que constriñen el juego. En el caso del proceso de trabajo del software, las condiciones que permiten una colaboración conjunta entre líderes o administradores de proyectos con los programadores se circunscribe a una serie de *arreglos sociales no escritos*, como es el caso cuando el gerente o líder de proyecto esta estimando el costo del proyecto, éste agrega un remanente de horas a favor del programador, para que así el programador cuente con un tiempo "extra" para prevenir cualquier incidente en el cumplimiento de "*resolver a tiempo*", además este remanente en caso de atraso no significaría costos para la empresa, puesto que ya están calculados en base a la experiencia; otro consenso es el hecho de permitir que el programador cuente con acceso al Chat en horas de trabajo, tengan cuentas personales (Yahoo, Hotmail, etc.); por el lado del programador éste corresponde, cediendo espacios de su vida fuera del trabajo, como resolver parte del problema en la casa; o bien investigando por cuenta propia en libros, tutoriales, Chat-room, blogs personales, entre otros tipo de actividades individuales.

Otra característica importante en la participación del juego de "*arreglárselas*" y tiene que ver con el *sentido subjetivo del trabajo como aprendizaje*, es decir los trabajadores participan en el juego para obtener un conjunto de *satisfacciones relativas* que tienen que ver con el rompimiento de la monotonía, el desgaste físico, minimizar el aburrimiento. Para el caso de la industria del software prevalece un desgaste mental, un conjunto de relaciones de poder basados mas en la *habilidad cognitiva* para resolver problemas mediante algoritmos. La búsqueda en la solución de un problema implica no sólo *el juego de resolver a tiempo* y las reglas no formales de "*arreglárselas*" y los "*arreglos sociales*", también la participación de la gerencia en el juego a través del hecho que el proceso de trabajo implica o supone una yuxtaposición de la jornada de trabajo, los tiempos de vida y los ámbitos simbólicos del programador. Superposición que no es ignorada en el proceso de trabajo, lo cual supone una ruptura entre las fronteras de la fábrica tradicional, ampliándose la jornada de trabajo, comprendiéndose una participación constante del trabajo simbólico del programador, más allá de los limites temporales de la jornada de trabajo o los físicos que impone las paredes de la empresa. Por último abordamos, una tercer regla informal hace referencia a poseer "*lógica de programador*", regla informal que esta relacionada con las satisfacciones e insatisfacciones como son los deseos, motivaciones, sentidos y construcción de significados que tiene que ver con la obtención de *satisfacciones relativas*, que en conjunto son resultado de un contexto de relaciones de poder, luchas, contradicciones, consensos y negociaciones que se propagan no solo en el proceso de trabajo, sino que se expanden mas allá de la frontera de la jornada de trabajo y abarca los tiempos de vida del programador y ámbitos simbólicos como aquellos espacios privados de autoaprendizaje, espacios virtuales de interacción que conforman parte del saber-hacer, espacios de poder que no necesariamente comparte con los otros programadores.

Para Burawoy, los juegos establecidos por trabajadores y gerencia en el piso de la fábrica no se constituyen contra la empresa, como señalan Elton Mayo y Cornelius Castoradis (Burawoy, [1979], 1989:107), no son sólo un resultado instrumental de un contexto de luchas, contradicciones y negociaciones que se propagan dentro de límites definidos por las necesidades de salarios y márgenes de beneficios a favor de los trabajadores; también son consecuencias de los deseos, motivaciones y necesidades radicales de una nueva concepción del trabajo, de un código alógico así como la obtención de *satisfacciones relativas*. Estas ideas

de Burawoy en parte nos auxilian para comprender el proceso de trabajo simbólico del Software, en el entendido que las empresas estudiadas, están más interesadas en el *juego de resolver a tiempo*, bajo las normas informales de *arreglárselas*; donde el "*arreglárselas*" significa un trabajo de colaboración entre los que intervienen en el proceso de resolver a tiempo, ésta colaboración no necesariamente corresponde al contexto de la jornada de trabajo, sino que también implica romper con la regulación de la jornada de trabajo, supone una yuxtaposición entre mundos de trabajo y mundos de vida del programador; por ejemplo un programador que por razones de falta de experiencia, información disponible, etc., no termina por solucionar un problema especifico, motivo por el cual acude a otro individuo, el cual puede ser el compañero de trabajo, el líder del proyecto, el programador senior, el programador-virtual, por Internet, acceder a información disponible en blogs, portales del tipo wiki, etc. consiguiendo la información que le ayuda u otorga ideas o conceptos para resolver parcial o totalmente el problema. Este conjunto de decisiones-acciones del programador van más allá de una alineación instrumental, se colocan más en el aspecto de las *satisfacciones relativas* que ya denominamos al principio de este apartado como *sentido subjetivo del trabajo como Aprendizaje* por parte del programador, medida subjetiva que podría determinar si continúa laborando o no en la empresa.

El saber-poder en el proceso de trabajo, el conjunto de arreglos sociales formales e informales, así como el conjunto de subjetividades presentes en el sentido del aprendizaje como las satisfacciones e insatisfacciones relativas, son cualidades que Foucault, considera como relaciones de saber-poder, donde la *resistencia* creativa y dinámica de los programadores a través de los acuerdos, consensos, negociaciones descritas, parecerían explicar el proceso de trabajo en el software; sin embargo, paralelamente se configuran conflictos y resistencias en el proceso de trabajo, unas veces soterrado, otras explícitamente presentes en dichos consensos. Consideramos que en el proceso de trabajo coexisten los siguientes conflictos:

1.- Conflicto abierto de la resistencia en el proceso de resolver a tiempo. En los contenidos subjetivos del juego por resolver ya descrito están contenidos una serie de conflictos y resistencias entre implementar procesos que las *normas disciplinarias* imponen, o bien las tareas o diseños que asigna la administración al programador, frente a la "libertad" del programador de elegir el mejor camino por resolver un algoritmo.Aunque, como señalaba

Burawoy, parecería ser una "sumisión voluntaria" ó en términos de Tripier una solución entre "igualdades cognitivas", coexisten prácticas concretas de resistencia, conflictos, choques de intereses que están impregnados en las reglas informales del juego (Edwards P.K. y Hugo Scullion [1982], 1987; y Baethge M.y H. Oberbeck [1986], 1995). Las resistencias abiertas que identificamos son aquellas mas comúnmente citadas por los entrevistados y las ingenierías del software: documentación insuficiente de las rutinas de código y prácticas de ocultar código. Esta acción de los programadores está suficientemente ilustrada por la ingeniería del software, en el proceso mismo del trabajo. Es una lucha manifiesta entre diseñadores y desarrolladores, la administración genera e implementa una serie de métodos para verificar la documentación y los programadores documentan lo menos posible, utilizan nombres de variables que sólo ellos conocen el significado, etc., la administración implementa *normas disciplinares* para que los programadores desarrollèn "código limpio" autodocumentable, es decir que sea sencillo de comprender, y en caso contrario que los programadores documenten y anoten los comentarios necesarios que expliquen el texto sígnico construido. Sin embargo, en la practica cotidiana de los programadores de software a la medida no documentan del todo, aún que existen metodologías orientadas a crear objetos-bibliotecas con repositorios de código no están eficientemente documentadas; aunque existen lenguajes de programación como Java que están estructurado de tal forma, que se debe documentar el código cuando se esta desarrollando el o los algoritmos. Aún así el programador oculta código o bien "engaña" al sistema, suscribiendo caracteres que validen "la documentación" sin haber documentado eficientemente.

2.- *Conflicto latente*: coexiste un comportamiento concreto pero soterrado, no abierto, las acciones no son explícitamente conflictivas, están latentes, presentes pero sin poder ser percibidas prontamente. Para el programador "ocultar código" o no documentarlo eficientemente lo justifica señalando que describirlo es complejo, ya que la "lógica de programador" es resolver a tiempo, arreglárselas para resolver, no es su tarea el escribir el como se resolvió, es una "caja negra" que resuelve el problema pero no "sabemos como opera".

3.- *Conflicto institucionalizado:* por ejemplo, se reconoce que no hay software libre de fallos (no hay cerro errores), es complejo medir la eficiencia y calidad en el software (no hay

cero defectos), no se entrega a tiempo (no hay justo a tiempo), entre otras incertidumbres que hacen complejo el desarrollo de software.

4.- *conflicto implícito:* hacemos referencia a aquel conflicto que no se expresa en una acción concreta, pero esta presente en la estructura de la situación del conflicto. Por ejemplo, cuando se les pregunto a los entrevistados como es que resolvían un algoritmo, señalaron una serie de características formales-informales, desde las aprendidas en la escuela, pasando por la *auto-capacitación,* hasta el conjunto de habilidades desarrolladas en el trabajo a través de la experiencia; sin embargo, la respuesta de un programador resumió el amplio espectro de respuesta en una frase: Entre las habilidades y la subjetividad "...es habilidad ¿no?... es don, se pude decir don...pero... hay personas más hábiles" (Programador CB3 DASA-ET1). La auto-capacitación esta presente en el discurso de los programadores quienes tienen una fuerte interacción virtual de auto aprendizaje. Esta profesión esta penetrada por una interacción virtual ya sea de aprendizaje, o bien para resolver problemas fuera del espacio productivo.

5.- *Conflicto estructural del poder*: Las ingenierías del software han construido una serie de *herramientas procedimentales* y *normas disciplinares* que constriñen y regulan el proceso de trabajo, se imponen como poder-saber en la organización del trabajo, divide los espacios de trabajo, propone que el programador es "un componente" mas que solo debe maquilar software, como *respuesta-resistencia* esta el conjunto de acciones subjetivas de los programadores, destacando el sentido subjetivo del aprendizaje. La construcción social del conocimiento está mediada por una *interacción* que necesariamente implica una intención por parte de aquel que posee destrezas, habilidades y experiencias sobre el menos experto. Las interacciones que dan lugar a los arreglos y consensos no están libres de intenciones, motivaciones, intereses, etc. El aprendizaje está mediado, por un lado por las subjetividades del que posee menos habilidades y destrezas y, por el otro lado, el individuo con mas experticia, guiado por sus intenciones, su cultura, emociones. intereses; selecciona y organiza los estímulos que se ponen en juego en la interacción a través de la implementación de herramientas y signos. Coexiste una primigenia división social del trabajo, explicada por la cada vez mas latente la presión estructural de las "nuevas" ingenierías en la transformación del proceso de trabajo en la industria del Software. Del proceso artesanal del software de la década de los sesenta a la intencionalidad de un proceso sistemático moderno del software, donde el mejor ejemplo son las fábricas de software. Entre estos dos periodos del proceso de

trabajo del software median una serie de transformaciones estructurales de conflicto por el poder, que podemos señalar dos posibles trayectorias: *partición del proceso de trabajo:* de un diseño y desarrollo centrado y concentrado en el programador en la etapa artesanal, se ha dividido en el diseño/requisitos por un lado y el desarrollo/calidad por el otro. Lo cual no se contradice con el hecho de que el programador está presente en ambas, como hemos señalado, debido a que estamos hablando del software a la medida para las empresas estudiadas. Sin embargo esta transformación estructural (que se aplica en las fábricas de software) está presente subjetivamente en el proceso de trabajo. También con la *separación de las herramientas-inmateriales y el signo/símbolos.* hacemos referencia al hecho que las ingenierías informáticas (ing. del software; ing. de requisitos; ing. de la usabilidad ó ergonomía del software entre otras) han formalizado un conjunto de herramientas inmateriales (ya descritas) para desarrollar código bajo "buenas prácticas", bajo metodologías comprensibles para la gerencia, a la usanza de ford-taylor en la industria del software está cuajándose una división social del trabajo, por un lado las herramientas-inmateriales y por otro lado el símbolo, entendiendo por éste al conjunto de símbolos abstractos-cognitivos que constituyen el sistema informático.

6.- *Conflicto estructural- explícito:* se configuran dos acciones concretas de resistencia, una a nivel del proceso de trabajo y otra a nivel de estructura. En el nivel del proceso de trabajo para las empresas de software a la medida cobra relevancia un modo de producción ágil. Donde no prevalece la división diseño/desarrollo, no se sigue una metodología en específico, sino el resolver conforme se va planteando el problema, es ágil, no es burocrático, sino horizontal. Es ágil, porque no se conforman equipos de trabajo complejos, sino que se aplica la revisión entre pares. Es decir, hay dos programadores intercambiables para resolver el problema, se intercambian sin previo aviso, la documentación es la mínima, la solución no está en la documentación del código, sino en la claridad con que se escriba el código. *Modo de producción libre:* a nivel estructural el modo de desarrollo del software de código libre se erige en un conflicto abierto con el modo de desarrollo del software libre. En el primer modo, se da prioridad a compartir el código, antes que las herramientas de desarrollo, se prioriza la lógica del programador y desarrollar código entendible, auto-documentado. La solución del problema no está en la documentación del código, sino en el acceso al código para que otro programador lo haga más transparente, más limpio, "menos ciego". El conjunto de

herramientas-inmateriales que se estructuran en torno de las comunidades de código libre no están orientadas a dividir el conocimiento como en las herramientas del software privado.

La aflicción del software está medida por una *constelación de dimensiones subjetivas* en el saber-hacer del proceso de trabajo, conjunto de saberes que se bosquejan como una serie de arreglos sociales formales e informales que concurren en el proceso de trabajo. Sin embargo, implícitamente las resistencias están embebidas en la solución de los problemas planteados, procedimiento cognitivo que no necesariamente es el mas eficiente ó que "es el único mejor camino", lo imprevisible, lo contingente, la incertidumbre forma parte de proceso de crear signos a través de símbolos.

Síntesis conceptual

La aflicción del software está presente en el proceso de crear signos a través de símbolos, la redacción del texto sígnico que configura los algoritmos está constreñida por un doble proceso, a nivel macro-meso y otro a nivel microsocial; el primero tiene que ver con las distintas ingenierías del software que se erigen como normas disciplinares y los distintas normas de calidad, metodologías, ciclos de vida, etc. que se conforman como herramientas procedimentales; tanto las normas como las herramientas son promesas de "delimitar" la aflicción del software, de instaurarse como "el único mejor camino" para resolver eficientemente en el proceso de trabajo a través de generar prácticas y dinámicas que induzcan desarrollar código transparente, código almacenado, código encapsulado, código despejado, etc. Es decir que se genere un texto sígnico claro, documentado, que sea evidente para el mantenimiento y la actualización del software. Las instituciones, las redes, las políticas gubernamentales, las estrategias empresariales, etc. se erigen como una barrera cognitiva, como "observadores" que promueven normas y herramientas en un proceso de trabajo que es ágil y fluido. A este nivel macro, lo definimos, como la intencionalidad de generar "fábricas de saberes". A nivel microsocial, el proceso de simbolización y significaciones en el trabajo, concurren una serie de configuraciones objetivas y subjetivas en la redacción del texto sígnico que es creado a través de una serie de dimensiones que podemos sintetizar entre tiempos-pensamientos y movimientos-reflexivos que están embebidos en una serie de constelaciones subjetivas y un flujo de aprendizajes. Las primeras están comprendidas por tres subdimensiones: subjetividad creativa, que es el proceso de construcción de símbolos en el

diseño de requisitos; la subjetividad signica, es la transformación de los símbolos a través de signos (lenguajes de programación) en textos sígnicos; la subjetividad que se objetiva, se refiere a la independencia del software del proceso de trabajo (piratería, clonación, etc.) y la dependencia del software de implementarse en las instalaciones del cliente (curva de aceptación y rechazo social). Así como la objetivación de la incertidumbre y contingencias presente a través de las aflicciones del software en la programación moderna.

Los flujos de aprendizaje en el proceso de trabajo, hace referencia a tres subdimensiones: las dinámicas relacionales, prácticas subjetivas y satisfacciones relativas, están intercaladas sucediéndose continuamente cada una de las rutinas o coyunturas que le integran, están enlazadas por un sentido subjetivo del trabajo como aprendizaje. Las dinámicas relacionales, se integran por una serie de consensos y juegos, de negociaciones y conflictos, de resistencias y sabotajes; éstas dinámicas se generan a través de una serie de satisfacciones relativas que se traducen en significaciones y sentidos, posiciones y miradas del trabajador cognitivo frente al trabajo que dan forma a un conjunto de arreglos sociales para resolver los problemas planteados en el trabajo, "arreglárselas" dentro o fuera del proceso de trabajo; sin embargo, resolver de la mejor manera, hacer frente a los detalles que debe contener una redacción sígnica compleja, implica que el trabajador cognitivo "disponga" de su creatividad y abstracción necesaria para resolver, a este proceso le denominamos lógica de programador y caja negra cognitiva. Entre las dinámicas relacionales y las satisfacciones relativas, tercian un sistema de prácticas subjetivas embebidas en el saber hacer, como es una documentación insuficiente para ocultar código; no documentar el proceso a través de generar código sucio; evitar comentarios y encubrir algoritmos desarrollando código ciego; no resolver de la mejor manera, sólo resolver el problema sin que medie una eficiencia creativa (sabotaje cognitivo), o bien, no resolver a tiempo, sobrepasar los tiempos de programación acordados (boicot cognitivo). El sentido subjetivo del trabajo como aprendizaje, esta cosificado por una serie de configuraciones subjetivas a nivel microsocial, que tiene que ver con los intereses e intenciones, motivaciones y representaciones, así como sentidos y significaciones del programador frente a la pantalla. En estas configuraciones es donde cuaja el sistema de signos y símbolos que integran el software, de la consistencia y coherencia interna es donde se genera un aspecto de la aflicción del software, entre los tiempos-pensamientos y movimientos reflexivos "creados a mano" es donde fragua el saber hacer y el saber como poder.

Bibliografía

Aboites, Jaime; Gabriela Dutréint, (Corrds.) (2003), Innovación. Aprendizaje y creación de capacidades tecnológicas. UAM-X. M.A. Porrua.

Aglietta, M. (1976): Regulación y crisis en el capitalismo. La experiencia de los Estados Unidos, Siglo XXI, Madrid, 1986.

ALADI.2003 La brecha digital y sus repercusiones en los países miembros de la LADI. Pp. 1-194 Secretaria general ALADI/SEC/Estudio 157. Rev 1 30 de julio de 2003.

Alexander, Ian. Requirements Engineering Tool Vendors and Freeware Suppliers. http://easyweb.easynet.co.uk/~iany/other/vendors.htm

AMITI, 2004 Propuesta para el uso y aprovechamiento de las Tecnologías de la Información y las Comunicaciones. (Software, hardware, etc.) Documento elaborado Asociación Mexicana de la Industria de Tecnologías de la Información PP. 1-16 (AMITI), (Pendiente dirección electrónica).

Arfuch, Leonor (2002), El espacio geográfico. Dilemas de la subjetividad contemporánea. Fondo de Cultura Económica, Buenos Aires, pp. 177-202.

Arora Ashish, Alfonso Gambardella y Salvatore Torrisi (2004), "In the footsteps of Silicon Valley? Indian and Irish Software in the International Division of Labor", en Bresnahan Timothy and Alfonso Gambardella (Comps.), Building high-tech clusters. Silicon Valley and Beyond, Cambridge University Press, Cambridge, UK, pp. 78-121.

Arora, Ashish Arora amd Suma Athreye, (2002) The software industry and India's economic development. Information Economics and Policy 14 (2002) 253–273, www.elsevier.com

Arora, Asís; Andrea Fosfuri; Alfonso Gambardella, (2002) Los mercados de tecnologías en la economía del conocimiento. Pp. 155-174. En Foray, Dominique (Consejero editorial) Revista Internacional de Ciencias Sociales. La Sociedad del Conocimiento, numero 171, Marzo de 2002.

Austerlic, Silvia, Las nuevas redes de conversación y su impacto en el medio ambiente humano y social; posibilidades y desafíos pp. 1-23. http://www.hipersociologia.org.ar/papers/austerlicsp.html

Bajtín, M, (2005, 1982) Estética de la creación verbal. Editorial Siglo XXI, 2005. México. Pp. 296-297.

Banco Mundial (2003) Aprendizaje permanente en la economía global del conocimiento. Desafió para los países en desarrollo; Editorial Alfaomega, pp. 152.

Bancomext, Industria del Software. Documento 2568, http://bancomext.com/Bancomext/

Banet, Miguel, Paradojas de los entornos virtuales. Pp.1-21 http://www.hipersociologia.org.ar/papers/banetsp.html

Becco, G. (2001). *Vigotsky y teorías sobre el aprendizaje. Conceptos centrales de la perspectiva vygotskiana.* INTERNET. www.monografías.com.

Bell, Daniel (1976), El advenimiento de la sociedad industrial, Madrid, Alianza.

Bercovich Néstor, Charles Swanke, (2003) Cooperacao e competitividade na industria de Software de Blumenau. Red de reestructuración y competitividad. Unidad de desarrollo industrial y Tecnológico. División de desarrollo productivo y empresarial numero 138, Santiago de Chile, Mar. 2003. CEPAL-ECLAC.

Bion, R.W. (1980) Aprendiendo de la experiencia. Traducción de Haydee B. Fernández. Ed. Paidos, España, 1980.

Bjorn-Andersen, N. et.al. The impact of system change in organizations. Sigthoff & Noordhoof, Ámsterdam, 1978; citado en Novara, F., tecnología de la información: concepción, introducción e implantación: un enfoque multidimensional, pp. 35-84, en Castillo, J.J. (editor) 1989, La ergonomía en la introducción de nuevas tecnologías en la empresa. Colección informes numero 8, Ministerio de Trabajo y Seguridad Social, Madrid, España.

Ble, V. Manuel y Rubén A. Villarroel A. (1991) Problemáticas y perspectivas del sector Software en la industria informática de México. UAM –I. Economía, Sept. de 1991, Tesis de Licenciatura. Pp.1-53

Borello, José; Darío Milesi; Marta Novick,; Sonia Roitter; Gabriel Yoguel. (2004) Las nuevas tecnologías de información y comunicación en la industria argentina: Difusión, uso y percepciones a partir de una encuesta realizada en la región Metropolitana de Buenos Aires. Pp. 1-31, Ponencia presentada en Seminario internacional "Redes, Tecnologías de Información y Comunicación y Desarrollo de Políticas Públicas" Littec, Instituto de Industria. Universidad Nacional de General Sarmiento Egida, Firenze. Argentina. 2004.

Boscherini Fabio, Marta Novick and Gabriel Yoguel, 2003 Nuevas tecnologías de información y comunicación: Los límites en la economía del conocimiento. Buenos Aires, Miño y Dávila editores, pp. 15-23.

Boyer, Robert. "La flexibilización del trabajo en Europa", capítulo I, Madrid, Ed. Ministerio de Trabajo de España, 1990.

Bracho F. Díaz, Arnoldo 1995, La cadena virtual de producción: hacia una fundamentación de la ingeniería. Soluciones Avanzadas.

Brave, S.; Nass, C. (2002). Emotion in human-computer interaction. In J. Jacko & A. Sears (Eds.), The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications (chap. 4). Hillsdale, NJ: Lawrence Erlbaum Associates. Disponible en: http://www.stanford.edu/~brave/papers/brave-HCI%20Handbook.pdf

Braverman, H. (1974), Trabajo y capital monopolista, México, ed. Nuestro tiempo.

Bruner, J. S. (1991). Actos de significado. Más allá de la revolución cognitiva. Madrid: Alianza. 2.

Bueno Carmen y Santos Ma. Josefa, 2003 *Nuevas tecnologías y cultura*, España, Anthropos y Universidad Iberoamericana

Business Week "21st century capitalism" 12 de Septiembre de 1994;

Campbell-Kelly, M. (1995), "Development and structure of the international software industry, 1950-1990", Business and Economic History, Volumen 24, N° 2, Winter

Cañada, J. (2005). Los elementos del diseño de interacción y la estética. Terremoto.net, 3 de Septiembre de 2005. Disponible en: http://www.terremoto.net/x/archivos/000191.html#000191

Casalet, Mónica, (2001) Construcción de ambientes favorables para el desarrollo de competencias laborales: tres estudios sectoriales. Pp. 1-74. Red de Reestructuración y Competitividad. División de Desarrollo Productivo y Empresarial. Número 106. CEPAL-ECLAC. Santiago de Chile, julio de 2001

Casas, R., y J., Dettmer (2003), Hacia la definición de un paradigma para las políticas de ciencia tecnología en el México del siglo XXI, en Santos C., (Coord.) Perspectivas y desafíos de la educación, la ciencia y la tecnología, México Escenarios del nuevo siglo, Instituto de investigaciones Sociales, UNAM. pp.197-270.

Casas, Rosalba (2004), "Enfoque para el análisis de redes y flujos de conocimiento", en Matilde, Luna (Coord.), Itinerarios del conocimiento. Formas, Dinámicas y Contenido. Un enfoque de redes, IIS-UNAM/Anthropos, Barcelona.

Casas, Rosalía y Jorge Dettmer (2004), Sociedad del conocimiento, capital Intelectual y organizaciones innovadoras. Pp.1-50, IIES-UNAM-Cátedra UEALC – FLACSO. 2004. Modulo 1.

Castells, Manuel y Pekka Himanen (2002) El estado del bienestar y la sociedad de la información, Alianza, Editorial, Madrid, 215 pp.;

Castillo, J.J. (1988, 1991), (Compilador), Las nuevas formas de organización del trabajo. Viejos retos de nuestro tiempo, Colección Informes, numero 3, Ministerio del trabajo y seguridad social.

Castillo, J.J. (1991) La informatización, trabajo y empleo en las pequeñas y medianas empresas españolas, MTSS, Madrid.

Castillo, J.J. (ed.) (1988) Las nuevas formas de organización del trabajo, MTSS, Madrid.

Castillo, J.J., Jiménez, V. y Santos, M. (1991) "Nuevas formas de organización del trabajo y de implicación directa en España", R.E.I.S., 56.

Castillo, J.J. La emergencia de nuevos modelos productivos. Producción ligera e intensificación del trabajo en España. Universidad Complutense de Madrid. Departamento de Sociología. En Trabajo y Sociedad.

Castillo, J.J. (2006), El trabajo fluido en la sociedad de la información: organización y división del trabajo en las fábricas de software. Colección Sociología del trabajo, Ed. Miño y DAvil. Madrid. Impreso en Argentina.

Chanfrault-Duchet, Marie-Françoise (1988), "Le système interactionnel du récit de vie", Sociétés, mayo, París, pp. 26-31. (mimeo, traducido);

Chudnovsky, D. (1986), "Los servicios informáticos y de telecomunicaciones en América Latina:

Chudnovsky, D., A. López y S. Melitsko (2001), "El sector de software y servicios informáticos en la Argentina. Situación actual y perspectivas de desarrollo", CENIT, Documento de Trabajo N° 27, Buenos Aires.

Cimoli, M., y Della G., M., (2003), The nature of technological change and its main implications on nacional system of innovation, en Aboites, J., y Dutréint, G., (Coords.) Innovación, aprendizaje y creación de capacidades tecnológicas, Universidad Autónoma Metropolitana, Xoch. Pp. 47-102.

CMM, Capabality Madurem Model . Capítulo 2. pp. 11-55. Metodología para el desarrollo de Software. (Pend, dirección electrónica)

Código de ética y ejercicio profesional de Ingeniería de software (Versión 5.2), recomendado por el Grupo de Trabajo. Conjunto del IEEE-CS/ACM en Ética y Ejercicio Profesional de Ingeniería de Software. Versión amplia en: http://www.ccsr.cse.dmu.ac.uk/resources/professionalism/codes/secode.html. En español: http://chapters.computer.org/dominicana/contribuciones/Codigo_Etica_Ing_Software.pdf

Coll S. César, (2003), Aprendizaje escolar contrición del conocimiento. Ed. Paidos. Primera reimpresión.

Coll, C., Martín, E., Mauri, T., Miras, M., Onrubia, J., Solé, I., y Zabala, A. (1999). *El constructivismo en el aula*. Barcelona: Graó.

Corona, T. Leonel, (1999). Teorías económicas de la Tecnología. Editorial JUS. México.

Correa, C. (1990), "The legal protection of software. Implications for latecomer strategies in newly industrializing economies and middle-income economies", OECD Development Center, Technical Paper N° 26, Paris.

Correa, C. (1999), "Propiedad intelectual y programas de computación", en 2° Congreso sobre Propiedad Intelectual. Cultura, Ciencia y Tecnología en la Universidad, Serie Ciencia y Tecnología en la UBA, Buenos Aires. (Internet)

Cubillo, Julio (1999), Cambio y continuidad en las organizaciones de gestión del conocimiento, Centro Latinoamericano de Documentación Económica y Social (CLADES), Serie información y Desarrollo num.10 Santiago de Chile, Dic. 1999., pp. 1-43 CEPAL-ECLAC.

Cuesta Q. Carlos (200) Arquitectura de software Dinámica basada en reflexión. Julio del 2002. Universidad de Valladolid, España. Pp. 1- 410. Biblioteca virtual Miguel de Cervantes. Tesis Doctoral. (Pend. Dirección electrónica)

D'Costa, A. P. (2000), "Export Growth and Path-Dependence The Locking-in of Innovations in the Software Industry", 4th International Conference on Technology Policy and Innovation, Curitiba, Agosto.

David Paul, A. and Dominique Foray (2002), "Fundamentos económicos de la sociedad del conocimiento", Comercio Exterior, vol. 52, no. 6, junio, pp. 472-490

David, P., y Foray D., (2002) Una introducción a la economía y a la sociedad del saber, en Foray D., Sociedad del conocimiento, Revista internacional de ciencias sociales, numero 171, Marzo.

De Jager, P. et.al. : 1998, Countdown Y2K: Business Survival Planning for the year 2000, Wiley; Karlson,

De la Fuente Ángel, (2003) Capital Humano y crecimiento en la economía del conocimiento. Instituto de análisis económico (CSIC) Madrid, Julio de 2003. (Pend. dirección electrónica)

De la Garza, Enrique (2005) "Neoinstitucionalismo, ¿opción ante la elección racional? : Una discusión entre la Economía y la sociología" Revista Mexicana de Sociología, Año 67, Núm. 1, Pp. 163-203

De la Garza, Enrique (2004) "¿Hacia dónde va la teoría social?", mimeo.

De la Garza Enrique, (Coord.), (2003), Tratado Latino Americano de Sociología del Trabajo. Colegio de México, FLACSO; UAM, Siglo XXI. México.

De la Garza, Enrique, 2003 Problemas clásicos y actuales de la crisis del trabajo.

De la Garza. E. (2003) Reestructuración Productiva. Empresas y Trabajadores en México al inicio del Siglo XXI. México, D.F.: STyPS. caps. I y V.

De la Garza Enrique, Carlos, Salas, (Coords.) (2003) La situación del Trabajo en México, 2003.UAM-IET-PyV.

De la Garza, Enrique (2002). "Conclusiones" En La Formación Socioeconómico Neoliberal, México. Plaza y Valdés. Pp. 222-234 .

De la Garza E, (2001), La formación socioeconómica neoliberal. México, Plaza y Valdez.

De la Garza, Enrique. (2001a) "Subjetividad, cultura y estructura". Revista Iztapalapa, Núm. 50. México. Pp. 83-104

De la Garza, Enrique. (2001b) "La epistemología crítica y el concepto de configuración" Revista Mexicana de Sociología 1/2001.pp. 109-127

De la Garza, Enrique (1999) "¿Fin del trabajo o trabajo sin fin?" En Casillo (ed.) El trabajo del futuro. Madrid. Pp. 3-7

De la Garza, Enrique (1999) "Epistemología de las teorías sobre modelos de producción" en Los retos teóricos de los estudios del trabajo, CLACSO, Buenos Aires

De la Garza Enrique, (coord.) (1998), Modelos de industrialización en México. UAM-I / CSH. México 1998.

De la Garza, Enrique. (1997) "Trabajo y mundos de la vida" en Zemelman y León (coords.). Subjetividad: umbrales del pensamiento social. Anthropos-CRIM-Coordinación de Humanidades. Barcelona. Pp. 75-92.

De la Garza, Enrique (1994). "Neoliberalismo y Estado" en Asa C. Laurell, *Estado y Políticas Sociales en el Neoliberalismo*. UAM –X. Distrito Federal. PP. 59-73

De la Garza, Enrique (1995) "Estructuralismo y positivismo en tiempo de la posmodernidad" en Zemelman (coord.)
Determinismos y alternativas en las ciencias socials de América Latina, CRIM-UNAM, Nueva Sociedad. Pp. 85-106.

De la Garza, Enrique. (1992) Crisis y Sujetos Sociales en México. CIIH-UNAM - Porrúa. México.

De la Garza, Enrique (1989) Un paradigma en el análisis del clase obrera. UAM-I. Cuadernos universitarios. México.

De la Garza, Enrique (1988) Hacia una metodología de la reconstrucción. Porrúa-UNAM, México.

Del Río, N. (1999). *Bordando sobre la zona de desarrollo próximo*. Revista de educación nueva época, no. 9. internet. http://www.jalisco.gob.mx/srias/educacion/09/9riolugo.html

Delgadillo, Gutiérrez Luis A. (2003), Modelo para evaluar la productividad en micro, pequeñas y medianas empresas de la cadena productiva de la electrónica, la informática y las

telecomunicaciones en el estado de Jalisco, México. Ponencia presentada en el 27 Congreso nacional de estadística e Investigación Operativa, Universidad de Guadalajara, Abril de 2003. (Pend. dirección electrónica)

Desmet, D.M.A.; Hekkert, P.; Hillen, M.G. (2003). Values and emotions; an empirical investigation in the relationship between emotional responses to products and human values. Proceedings of the fifth European academy of design conference, Barcelona, Spain. Disponible en: http://static.studiolab.io.tudelft.nl/gems/desmet/papervaluesemotion.pdf

D'Hertefelt, S. (2000). Emerging and future usability challenges: designing user experiences and user communities. InteractionArchitect.com, 2 February 2000. Disponible en: http://www.interactionarchitect.com/future/vision20000202shd.htm

Dillon, A. (2001). Beyond Usability: Process, Outcome and Affect in human computer interactions. Lazerow Lecture 2001, at the Faculty of Information Studies, University of Toronto, March 2001. Disponible en: http://www.ischool.utexas.edu/~adillon/publications/beyond_usability.html

DNX (2005). Usabilidad y Experiencia de Usuario. Microsoft España: Guía Práctica de Usabilidad Web. Disponible en: http://www.microsoft.com/spain/empresas/guias/usabilidad/experiencia_usuario.mspx

Dosi, G., (2003), Paradigmas tecnológicos y trayectorias tecnológicas, en Chesnais F., y Neffa J., (Comps.) Ciencia, tecnología y crecimiento económico. Argentina: CEIL PIETTE CONICET.

E., y J. Kolber, A basic Introduction to Y2K: How the year 2000 Computer Crisis Affects You?, Net era publication, Inc.
Economía de la innovación. Las visiones de Ralph Landau y Christopher Freeman. P1-115 Versión electrónica www.redalyc.com

Edwards P.K. y Hugo Scullion ([1982], 1987) La organización social del conflicto laboral. Control y resistencia en la fábrica.MTSS, Madrid España; y Baethge Martín y Herbert Oberbeck ([1986], 1995 El futuro de los empleado, Nuevas tecnologías y perspectivas profesionales en la gerencia empresarial. MITSS, Madrid, España.

Elias, Norbert (1970): Sociología fundamental. Gedisa. Madrid, 1982.

Elster, J. (1983): El cambio tecnológico. Investigación sobre la racionalidad y la transformación social. Gedisa. Barcelona, 1990.

Elster, J. (1989): Tuercas y tornillos. Una introducción a los conceptos básicos de las ciencias sociales. Gedisa. Barcelona, 1990.

Piattini, Esteban 1995) J. L. Esteban, M. Piattini. "Procesos del ciclo de vida del software". Novática, Nov./dic. 1995.

Boscherini, Fabio y Lucio Poma, Territorio, conocimiento y competitividad de las empresas. El rol de las instituciones en el espacio global, Buenos Aires/Madrid, Miño y Dávila Editores.

Feldman, M., (2002), La revolución de Internet y a geografía de la innovación, en Foray D., Sociedad del conocimiento, Revista internacional de ciencias sociales, numero 171, Marzo.

Feldman, P. Maryann, (2002), La revolución de Internet y la geografía de la innovación. Pp.60-74 En Foray, Dominique (Consejero editorial) Revista Internacional de Ciencias Sociales. La Sociedad del Conocimiento, numero 171, Marzo de 2002.

Féliz Tezanos José, (2004) La sociedad del trabajo y el mundo del trabajo. pp.1-49. UNED-Cátedra UEALC – FLACSO. 2004. Modulo 3.

Fernández, Steinko, A. (1992) "Las cualificaciones de los trabajadores en empresas españolas con células flexibles de fabricación", Sociología del Trabajo, 16, 3-25.

Fernández, Steinko, A. (1993) "Dinámica organizativa, cualificaciones y clasificación profesional en empresas españolas de alta tecnología", Economía y Sociología del Trabajo, 21/22, 93-106.

Font Playán Isabel; Arturo S. Martínez, (Coords.) (2000), Horizontes complejos en la era de la información. UAM-Ciencias Sociales y Humanidades. Serie Administración.

Forero Clemente; Jaramillo Salazar (2002), El acceso a los investigadores de los países menos desarrollados a la ciencia y la tecnología internacional pp. 175-191. En Foray, Dominique (Consejero editorial) Revista Internacional de Ciencias Sociales. La Sociedad del Conocimiento, numero 171, Marzo de 2002.

Foucault, Michel, ([1967]1976) Historia de la locura en la época clásica. FCE, Buenos Aires.

Foucault, Michel, (1999) "Verdad y poder" en Foucault, Estrategias del poder, Paidós, Buenos Aires.

Foucault, Michel, (1977) Historia de la Sexualidad.: Siglo XXI, México.

Foucault, Michel, (1991) "La gubermentalidad", en Espacios de poder. Ediciones La Piqueta. Madrid.

Foucault, Michel, [1966] (2004) Las palabras y las cosas. Siglo XXI. México.

Freeman, C. (1975), La teoría económica de la innovación industrial, Alianza, Madrid.

Fritz, Alejandro, Acerca del Freeware, Shareware y Software Comercial. http://www.hipersociologia.org.ar/papers/fritzsp.html
Fundación COTEC para la innovación tecnológica. (2004), Análisis del proceso de innovación en las empresas de servicios (Incluye Análisis del Servicio del Software) p. 175 Abril de 2004. http://www.cotec.es

Garcia Canclini, Néstor (1982) Las culturas populares en el capitalismo. Nueva Visión, México.

Garcia Canclini, Néstor (1990) "Introducción: La sociología de la cultura de Pierre Bourdieu" en Bourdieu, P. Sociología y Cultura. CONACULTA-Grijalbo, México. Pp. 9-50.

García, Montalvo José (2004) Mercado DE trabajo y educación en el contexto de la nuevas sociedad del conocimiento. Pp.1-42 Universitat Pompeu Fabra- Cátedra UEALC – FLACSO. 2004. Modulo 2.

Giddens, Anthony (2000), Un mundo desbocado, Madrid, Taurus.

Godolier Maurice (1989), Lo ideal y lo material, Madrid, Taurus Humanidades. Gintis. Herbert (1983) "La naturaleza del intercambio laboral y la teoría de la producción capitalista: en Tohara , L. (Comp.) El mercado de trabajo: teorías y aplicaciones, Madrid, Alianza Universidad, 1986, p. 176;

González Vilalta, D. (2004). ¿Qué es la Experiencia del Usuario?. Nethodical, 13 de Febrero de 2004. Disponible en: http://www.nethodical.com/archivos/000020.html

Gonzalo C. Agustín, (1991), Ingeniería del Software: Práctica de programación. Editorial RA-MA, Serie Paradigma.

Gonzalo C. Agustín, (1991), Ingeniería del Software: Práctica de programación. Editorial RA-MA, Serie Paradigma.

Gordillo M., y J. López (2000) Acercando la ciencia a la sociedad: la perspectiva CTS y su implantación educativa, en Medina, M., y T. Kwiatkowska (coords.), Ciencia, tecnología/naturaleza, cultura en el siglo XXI,

Graham, Stuart and David C. Mowery, (1999) Intellectual Property Protection in the Software Industry, pp.. 1-31 Haas School of Business U.C. Berkeley. Ponencia presentada en "International Symposium on Innovation and Patents," Hitotsubashi University, Tokyo, Japan. Feb. 12-13, 1999, (http://emlab.berkeley.edu/users/bhhall/swconf.doc)

Granollers i Saltiveri, Toni, (2004), MPIu+a. Una metodología que integra la Ingeniería del Software, la Interacción Persona-Ordenador y la Accesibilidad en el contexto de equipos de desarrollo multidisciplinares", Universidad de Lleida, Julio de 2004. España. http://griho.net/;

Grimaldi, Rosa and Salvatore Torrisi, (2001) Codified-tacit and general-specific knowledge in the division of labour among firms A study of the software industry.Volume 30, Issue 9 , December 2001, Pages 1425-1442 www.elsevier.com

Habermas, J. (1968, a): Conocimiento e interés. Taurus. Madrid, 1982.

Habermas, J. (1968, b): Ciencia y técnica como «ideología». Tecnos. Madrid, 1989.

Hardt. M. y A. Negri (2001). Empire, Cambridge: Harvard University Press.

Hartson, H.R. (1998). Human-computer interaction: Interdisciplinary roots and trends. The Jo urnal of Systems and Software
43 (1998), pp. 103-118.

Hassan Montero, Y.; Martín Fernández; F.J. (2003). Más allá de la Usabilidad: Interfaces 'afectivas'. NoSoloUsabilidad e-Magazine, 28 de Octubre de 2003. Disponible en: http://www.nosolousabilidad.com/articulos/interfaces_afectivas.htm

Hassan, Y.; Martín Fernández, F.J.; Iazza, G. (2004). Diseño Web Centrado en el Usuario: Usabilidad y Arquitectura de la Información. Hipertext.net, núm. 2, 2004. Disponible en: http://www.hipertext.net/web/pag206.htm

Hilera et al. 1997) José R. Hilera, José A. Gutiérrez, J. Javier Martínez. "Estándares en la Ingeniería del Software". Novática. Nov./dic. 1997. Número 130.

Hoch, D., C. Roeding, G. Purkert y S. Lindner (1999), Secrets of Software Success. Managements Insights from 100 Software Firms around the World, Harvard Business School Press, Boston;

Htchuel, Armand;Pascal Le Masson, Benoit Well (2002), De la gestión de los conocimientos a las organizaciones orientadas a la concepción. Pp. 29-46 En Foray, Dominique (Consejero editorial) Revista Internacional de Ciencias Sociales. La Sociedad del Conocimiento, numero 171, Marzo de 2002.

Hualde A. y Gomis Redi (2004), "La construcción de un cluster de software en la frontera noroeste de México", Revista Frontera Norte, vol. 16, no. 32, julio-diciembre, México, pp. 7-34.

IEEE Software Requirement Engineering, Second Edition, Editado por Richard H. T. y Merlin Dorfman, IEEE Computing Society, New York, NY. 1997; Wiegers Karl E. Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. Microsoft Press, 2003

IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE, 1990 Pendiente línea electrónica.

International Standard ISO/IEC 12207. "Information technology- Software life cycle processes". 1995.

Irons, L.R. (2003). Rapid Ethnography for User Experience Design. I.C. Technologies White Paper, 2003. Disponible en: http://www.ic-t.com/publications/UEDesign.pdf

ISO 9241-11. (1994). Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs)—Part 11: Guidance on Usability.

Itzigsohn J. Y J.P. Pérez S. La empleabilidad exitosa. El caso de los ingenieros electrónicos en Costa Rica. Pp.1-44 Cap.II. Entre la empleabilidad y la exclusión. Respuestas laborales a la globalización en América Latina. (exclusivo uso interno)

Jiménez, C. *Guía Metodológica para el desarrollo o adaptación del software*. Universidad Nacional de Colombia, Medellín. 120 p. 2003.

Jordan, P.W. (1998). Human factors for pleasure in product use. En: Applied Ergonomics, Vol. 29, n° 1, pp. 25-33.

Kankainen, A. (2002). Thinking model and tools for understanding user experience related to information appliance product concept. Tesis Doctoral, Helsinky University of Technology, 9 de Diciembre de 2002,. Disponible en: http://lib.tkk.fi/Diss/2002/isbn9512263076/

Katz, J. y Hilbert, M. (2003). Los caminos hacia una sociedad de la información en América Latina y el Caribe. Cepal Alfaomega. Bogotá.

Katz, M. y C. Shapiro (1985), "Network externalities, competition and compatibility", American Economic Review,Volumen 75, N° 3.

Kenney Martin and Florida Richard (2000), "Venture capital in Silicon Valley:Fueling new firm formation", in Kenney Martin (Editor), Understanding Silicon Valley. The anatomy of an entrepreneurial region, Standford University Press, California, pp. 98-123.

Kern, H. y Schumann, 1988, El fin de la división del trabajo; Revista e sociología del Trabajo. num. 2, 1988.

Kern, H. y Schumann, 1989, El fin de la división del trabajo: Racionalización en la producción industrial. Madrid, M.T.S.S

Kern, H. y Schumann M. (1989) El fin de la división del trabajo, MTSS, Madrid

Knapp Bjerén, A. (2003). La Experiencia del Usuario. En: Knapp Bjerén, A. (coord.). La Experiencia del Usuario. Madrid: Anaya Multimedia, 2003, ISBN 84-415-1044-X.

Koch, Christopher, Derek Slater and E. Baatz, The ABCs of ERP pp. 1-10 (Enterprise Resource Planning Software). Pendiente dirección electrónica

Krotz, Esteban (1993), la cultura Objetivada, México, Universidad Autónoma Metropolitana – Iztapalapa;

Kuhn, Thomas S. (1962): La estructura de las revoluciones científicas. F.C.E. México, 1990.

Kuhn, Thomas S. (1987): ¿Qué son las revoluciones científicas? y otros ensayos. Paidós/I.C.E.-U.A.B. Barcelona, 1989.

Lakatos, I. (1972): "La falsación y la metodología de los programas de investigación científica", in: I. Lakatos y A. Musgrave (eds.) (1972). pp 203-344.

Lakatos, I. y A. Musgrave (eds.) (1972): La crítica y el desarrollo del conocimiento. Actas del Coloquio Internacional de Filosofía de la Ciencia celebrado en Londres en 1965. Ediciones Grijalbo. Barcelona, 1975.

Lam, Alice, (2002) Los modelos societales alternativos del Aprendizaje e Innovación en la economía del conocimiento. Pp. 87-109, En Foray, Dominique (Consejero editorial) Revista Internacional de Ciencias Sociales. La Sociedad del Conocimiento, numero 171, Marzo de 2002.

Lerner, J. y J. Tirole (2000), "The simple economics of open source", Harvard Business School, Working Paper 00-059, marzo. internet

Lesemann. Frederic (2004), Sociedad del conocimiento: Los cambios en el mundo del trabajo y las nuevas competencias de los trabajadores. PP. 1-65. Universidad de Québec-Cátedra UEALC – FLACSO. 2004. Modulo 2.

Lope Andreu y Antonio M. Artiles, (1993), Cambio técnico recualificación, nueva época, num. 19, pp. 69-07.

López, Andrés (2003), "El sector de software y servicios informáticos en la Argentina: es posible una inserción exportadora sostenible?" en Boscherini Fabio, Marta Novick y Gabriel Yoguel (Comps.), Nuevas tecnologías de información y comunicación: Los límites en la economía del conocimiento, Buenos Aires, Miño y Dávila editores, pp. 175-2003.

Lorenzen, M. y Mannke, V. (2002), "Global Strategy and the Acquisition of Local Knowledge: How MNCs Enter Regional Knowledge Clusters", DRUID, Working Paper nro 02-08, Aalborg. Internet

Luna, M. (2003), La red como mecanismo de coordinación y las redes de conocimiento, en Luna, M., (Coord.) Itinerarios del conocimiento: formas dinámicas y contenido. Un enfoque de redes, Ed. España (Anthropos) México (Universidad Autónoma Metropolitana Iztapalapa) pp. 51-78.

Luna, Matilde (2003), Itinerarios del conocimiento: formas, dinámicas y contenido. Un enfoque de redes, Tecnología, ciencia, naturaleza y sociedad,Barcelona, Anthropos.

Lundvall, Bengt-Ake (2003), "Por qué la nueva economía es una economía del aprendizaje?", en Boscherini Fabio, Marta Novick and Gabriel Yoguel (Comps.), Nuevas tecnologías de información y comunicación: Los límites en la economía del conocimiento, Buenos Aires, Miño y Dávila editores, pp. 39-55.

Macdonald, Stuart (2001), Exploring the Hidden Costs of Patents. Notes of a talk given at Quaker House, Geneva, 16 May 2001, Quaker United Nations Office – Geneva. www.elsevier.com

Madrid: Morata.

Mark, N. y P. Rigby (1994), Ingeniería del Software, Editorial BT-Gran Bretaña, Megabyte;

Maroto1, Carlos y Jorge Zavala. La industria del software en México. (Pend, dirección electrónica)

Márquez, T., Redes (2003) contra la incertidumbre en Software (2003), en Luna, M., (Coord.) Itinerarios del conocimiento: formas dinámicas y contenido. Un enfoque de redes, Ed. Anthropos-Universidad Autónoma Metropolitana Iztapalapa. pp. 188-228.

Matilde Luna, María J. Santos y Ricardo Tirado, 2003 La formación de redes de conocimiento: Una perspectiva regional desde México, México/España, UNAM/Anthropos Editorial, pp. 13-34.

Maurizio L. y A. Negri, Trabajo inmaterial. Formas de vida y producción de subjetividad, DP&A Ed.Río de Janeiro, 2001 Disponible: http://www.rebelion.org/libros/TrabajoInmateria011202.pdf. Acceso 05/09/07

McClelland, I. (2005). 'User Experience' Design: A new form of design practice takes shape. CHI 2005, April 2–7, 2005, Portland, Oregon, USA, pp. 1096-1097. Disponible en: http://uxnet.org/devcon/DevCon-McClelland.pdf

McClure, C., (1993) CASE. La automatización del Software. Editorial RA-MA, Addison-Wesley Iberoamérica, Traducción de José M., Ortega.

Medina, Manuel; Kwiatkowska, Teresa, (Coords.) (2000) Ciencia, tecnología/naturaleza, cultura en el siglo XXI. (Tecnología, ciencia, naturaleza y sociedad). UAM-I, México. Anthropos, España.

Mercier, Delphine y Pierre Tripier, 2003, El Neo-Management y la ceguera Organizacional. Ponencia presentada en Asociación Latinoamericana de Sociología del Trabajo, 4° Congreso La Habana 9-13 de Septiembre de 2003.

Metcalfe. (1989). The diffusion of innovation: an Interpretative Survey. En Dosi G, C. Freeman R. Nelson, G. Silverberg y Soete, L. (eds.), Technical Change and Economic Theory, Pinter, London México: Editorial Crítica, Grupo editorial Grijalbo.

Meyer, Bertrand (1997) Construcción de Software orientado a objetos. 2 edición, traducción de Katrib M., Miguel; García B., Rafael; Sánchez, Salvador, editorial Prentice-All.

Meyer, Bertrand (1997) Construcción de Software orientado a objetos. 2 edición, traducción de Katrib M., Miguel; García B., Rafael; Sánchez, Salvador, editorial Prentice-All

Micheli, J. (2003). El trabajo de digitofactura en la economía postindustrial en Espacios globales: espacios del capitalismo, Ed. Carmen Bueno (en prensa). México.

Micheli, Jordy (2003). Digitofactura: Flexibilización, Internet y Trabajadores del Conocimiento. Ponencia presentada en el Congreso nacional AMET-2003. Hermosillo, Sonora.

Minc, Alain,(2001) www.capitalsimo.net Editorial Paidós.

Mochi, Aleman, Prudencio (2003) La revolución electrónica y la producción de Software. Ponencia presentada en el Congreso nacional de AMET. Hermosillo, Sonora. 2003.

Molina, T. Miguel A. (2000) Gestión de Recursos Humanos en proyectos informáticos. (Desarrolladores de software). Universidad de Castilla-la Mancha, Escuela Superior de Informática, España, May. 2000 Pendiente dirección electrónica

Morville, P. (2004). User Experience Design. SemanticStudios, 21 de Junio de 2004. Disponible en: http://semanticstudios.com/publications/semantics/000029.php

Mumford, E.y Sackman, H. (Eds) Human choice and computers. North Holland, Ámsterdam, 1979, citado en Novara, F., tecnología de la información: concepción, introducción e implantación: un enfoque multidimensional, pp. 35-84, en Castillo, J.J. (editor) 1989, La ergonomía en la introducción de nuevas tecnologías en la empresa. Colección informes numero 8, Ministerio de Trabajo y Seguridad Social, Madrid, España.

N. Dayasindhu, (2002) Technovation for the Indian software industry Volume 22, Issue 9 , September 2002, Pages 551-560 http://www.sciencedirect.com/science

N. Dayasindhua, , and S. Chandrashekarb, Indian remote sensing program: A national system of innovation? a Software Engineering Technology Labs, Infosys Technologies Ltd., Electronics City, Bangalore 560100, India2 October 2004 www.elsevier.com

Naisbit, J. (1982), Megatoends, Warner Books; Osrborne , D., The invisible Computer, MIT Press,

Nasscom (2000), "Indian IT Software and Services Industry", disponible en http://www.nasscom.org.

Nasscom (2001b), "Intellectual property right in India", http://www.nasscom.org

Naville, Pierre, (1985) El progreso técnico, la evolución del trabajo y la organización de la empresa, pp. 369-385 En Georges Friedmann y Pierre Naville, Tratado de Sociología del Trabajo Tomo I. F.C.E. 1985

N-economía, Mayo 2003, Boletín mensual www.neconomia.com (Datos de NTI en el mundo: Software, Hardware, etc.)

Neffa, J., (1990), El proceso de trabajo y la economía de tiempo. Bueno Aires ed. CREEDLA Humanitas.

Neffa, Julio César y Enrique de la Garza Toledo (Comp.), El futuro del trabajo. El trabajo del futuro. CLACSO/ASDI, CEIL-PIETTE CONICET, Trabajo y Sociedad, Buenos Aires, 2001.

Nemirovsky, A. y G. Yoguel (2000), "La creación de firmas high-tech y el desarrollo de la tecnología de información/comunicación en el Sillicon Valley. Algunas lecciones para el caso argentino", Boletín Informativo Techint 301, Enero-Marzo, Buenos Aires.

Newman, D., Griffin, P., Cole, M. (1991). La zona de construcción del conocimiento.

Nielsen J. (2003). PR on Websites: Increasing Usability. March, 2003 Jakob Nielsen's Alertbox. Disponible http://www.useit.com/alertbox/20030310.html ;

Nielsen, J.; Mack, R.L. (1994). Usability Inspection Methods. John Wiley & Sons, New York, NY.; Granollers, T.; Perdrix, F.; Lorés, J.; Grupo de Investigación GRIHO. Universidad de Lleida, España.

Nirjar, Abhishek and Andrew Tylecote, (2004), Breaking out of Lock-in: Insights from case studies into ways up the value ladder for Indian software SMEs. University of Sheffield. Management School Paper No 2004.01 January 2004. (Pend. Dirección electrónica)

Nóbile, Nicolás, Escritura electrónica y nuevas formas de subjetividad. http://www.hipersociologia.org.ar/papers/Nobilesp.html

Norman, D. (2002). Emotion and Design: Attractive things work better. Interactions Magazine, ix (4), pp. 36-42. Disponible en: http://www.jnd.org/dn.mss/emotion_design_at.html

Novick, M. (2002), "La dinámica de oferta y demanda de competencias en un sector basado en el conocimiento en Argentina", CEPAL, Serie Desarrollo Productivo.

Novick, M., (1999), La transformación de la organización del trabajo, en E. De la Garza (Coord.) Tratado latinoamericano de Sociología del Trabajo, México FCE.

OECD (2000), Information Technology Outlook 2000. ICTs, E-commerce and the information economy, OECD, Paris.

OECD (2001), "Innovative Clusters. Drivers of National Innovation Systems.", OECD Proceedings, París.

OECD (2002), OECD Information Technology Outlook. ICTs and the information economy, OECD, Paris.

OECD, (2001) Telecommunications Database.

OECD, (2003) Science, technology and industry scoreboard.

OECD, 2004) Information Technology Outlook

Ordoñez S. y Dabat A., (2006), Revolución informática, nuevo ciclo industrial e industria electrónica en México, IIEC-UNAM,

Ordoñez S. (2006) "Capitalismo del conocimiento: ¿México en la integración?", Problemas del Desarrollo, Vol. 37, num. 146, julio-septiembre.

Ordóñez S., (2004) "Nueva fase de desarrollo y capitalismo del conocimiento: elementos teóricos", Comercio Exterior, vol. 54, n° 1, enero.

Oredoñez S. (2006) "Crisis y reestructuración de la industria electrónica mundial y reconversión en México", en Comercio Exterior, Vol. 56, n° 7.

Organización Internacional del Trabajo (2002) Aprender y formarse para trabajar en la sociedad del conocimiento, Ginebra, Informe IV. 136 pp.;

Pal, Nilendu y T.R. Madanmohan (2002) Competing on Open Source: Strategies and Practise. http://opensource.mit.edu/papers

Paulk, M., et.l. (1993) Capability Maturity Model for Software. Software Engineering Institute, Carnegie Mellon University; véase: www.sei.cmu.edu

Paulk, Mark C.; et.al. (1993); Capability Maturity Model 1.1; IEEE Software, Vol. 10. No. 4, Julio 1993 pp. 18-27. CMMI-Adoption Information 2003, www.sei.cmu.edu/cmmi/adoption/adoption.html

Perazzo, R., M. Delbue, J. Ordoñez y A. Ridner (1999), "Oportunidades para la producción y exportación argentina de software", Documento de Trabajo N° 9, Agencia Nacional de Promoción Científica y Tecnológica, Buenos Aires.

Pérez, C. Gilberto, La zona de desarrollo próximo y los problemas de fondo en el estudio del desarrollo humano desde una perspectiva cultural, en http://www.jalisco.gob.mx/srias/educacion/9gilpere.html

Pérez Sáinz (Editor), Maribel Carrera, Roque Castro, Rafael del Cid, Jorge Monge, Encadenamientos globales y pequeñas empresas en Centroamérica. FLACSO, Octubre 2002.

Pérez, C., (2003) Revoluciones tecnológicas, cambios de paradigma y de marco socioinstitucionales, en Aboites, J., y Dutréint, G., (Coords.) Innovación, aprendizaje y creación de capacidades tecnológicas, UAM, Xochimilco. Pp. 13-45

Pérez, Carlota y C. Ominami (1986), La Tercera Revolución Tecnológica, Buenos Aires,

Piaget, jean y Rolando García. (2004), Psicogénesis e historia de la ciencia. Traducción de Rolando García . Ed. Siglo XXI, Décima reimpresión.

Picard, R.W.; Klein, J. (2002). Computers that Recognise and Respond to User Emotion: Theoretical and Practical Implications. En: Interacting with computers, 14, 2, 2002. Disponible en: http://vismod.media.mit.edu/pub/tech-reports/TR-538.pdf

Piña, Carlos (1989), "Sobre la naturaleza del discurso autobiográfico", Argumentos. Universidad Autónoma Metropolitana, Xochimilco, núm. 7, agosto, México, pp. 131-160.;

Popper, Karl R. (1934): La lógica de la investigación científica. Tecnos. Madrid, 1971.

Popper, Karl R. (1972): Conocimiento objetivo. Un enfoque evolucionista. Tecnos. Madrid, 1992.

Pozo, I. Juan (2003), Teorías cognitivas del aprendizaje. Facultad de Psicología, Universidad Autónoma de Madrid, España. 2003. Ed. Morata. Quinta edición.

Pozo, J (1996.) : "Teorías Cognitivas del Aprendizaje"; Eds. Morata, Madrid, ISBN: 84-7112-335-5

Pressman, R. S. *Ingeniería del Software:Un enfoque práctico*. Quinta Edición, McGraw Hill, Madrid. 601 p. 2002.

Presuman S. Roger (2002). Ingeniería del Software. Un enfoque práctico. Adaptado por Darle Ince. Mc. Graw Hill.. Quinta edición. España, 2002

Proceso Software. Pp. 1- 37 (Desarrollo estructuración de un Software), Pendiente dirección electrónica.

Programa para la Competitividad de la Industria Electrónica y de Alta Tecnología.- Secretaría de Economía. 2001. Gobierno de la República Mexicana. Pp. 1-99. (Pend. dirección electrónica)

PROSOFT, (2004) "Estudio para Determinar la Cantidad y Calidad de Recursos Humanos Necesarios para el Desarrollo de la Industria de Software en México", elaborado por la Universidad autónoma metropolitana a encargo de la Secretaria de economía.

PROSOFT: 2004. Estudio del perfil de la industria mexicana de software para definir los nichos de mercado internacional acordes al perfil y competitividad de la industria. Secretaria de Economía. Gobierno de la República Méxicana. 2004.

Revisiones estructuradas (walktroug) (Estructura-contenido de un programa de Software). Universidad de La Coruña, Facultad de Informática. España. Pendiente dirección electrónica

Rifkin, Jeremy (1996) El fin del trabajo. Paidós, Barcelona.

Robles, Gregorio, Los desarrolladores de software libre, p1-26 grex@scouts-es.org (Pend. dirección electrónica)

Roggof B (1993) Aprendices del pensamiento. El desarrollo cognitivo en el contexto social, Paidos, Barcelona.

Ruiz D. Clemente (2004), Potencialidades de las entidades federativas para desarrollar núcleos de economía digital. Facultad de economía-UNAM

Rullani, Enzo (2001), "El valor del conocimiento", en Boscherini Fabio y Lucio Poma (Comps.), Territorio, conocimiento y competitividad de las empresas, Madrid/Buenos aires, Miño y Dávila Editores, pp. 229-258.

Sallstrom Consulting, Nathan Associates. Cámara Nacional de la Industria Electrónica, de Telecomunicaciones e Informática (CANIETI). Secretaría de Economía. http://www.software.net.mx
Sallstrom, Laura y Robert, Damuth, (2003), El Papel Fundamental de la Industria del Software en el Crecimiento Económico – Foco: México Documento patrocinado por la Computing Technology Industry Association (CompTIA). http://www.software.net.mx

Sánchez Daza, Germán (2005), "La industria del software y estratégias para su desarrollo" en German Sánchez Daza (Coord.), Innovación en la sociedad del conocimiento, México, Benemérita Universidad Autónoma de Puebla, pp. 431- 453.
Santos C. M. Josefa, (Coord.) (2003), Perspectivas y desafíos de la educación, la ciencia y la tecnología. México: Escenarios del nuevo siglo. IIE-UNAM 2003

Sanz, Marcos (2003), "Metodología de desarrollo de interfaces de usuario gráficas para sistemas de información interactivos con alta usabilidad". Escuela Técnica Superior de Ingenieros de Telecomunicación de Madrid. http://griho.net/;

Schumpeter, J.A (1967): Síntesis de la evolución de la ciencia económica y sus métodos. Oikos-Tau. Barcelona, 1967.

Secretaría de Economía (2002), Programa para el desarrollo de la industria del software, Secretaría de Economía, México.

Segre M. Lidia, Rapkiewicz E. Clevi, (2001), Mercado de trabajo y formación de recursos humanos en tecnologías de la información en Brasil. ¿Encuentro o desencuentro?. Red de reestructuración y Competitividad. División de desarrollo productivo y empresarial numero 117, Santiago de Chile, Dic. 2001. CEPAL-ECLAC.

Sierra Bravo, R. (1994), Técnicas de investigación social, Editorial Paraninfo, Madrid, pp. 304-322;

Sierra, Francisco (1998), "Función y sentido de la entrevista cualitativa en investigación social", en: Jesús Galindo Cáceres (coord), Técnicas de investigación en Sociedad, Cultura y Comunicación, CONACULTA-Addison Wesley Longman, pp. 277-333;

Software Engineering Institute. Requirements Engineering Project. Requirements Engineering and Analysis Workshop Proceedings (CMU/SEI -9I - TR-30). Pittsburgh, PA: Software

Engineering Institute, Carnegie Mellon University. 1991. http://www.sei.cmu.edu/director/aboutSEI.html

Sproull, L. y Hofmeister, K. 1982, Implementation: A cognitive Approach. Working paper, Mellon University, 1982. citado en Novara, F., tecnología de la información: concepción, introducción e implantación: un enfoque multidimensional, pp. 35-84, en Castillo, J.J. (editor) 1989, La ergonomía en la introducción de nuevas tecnologías en la empresa. Colección informes numero 8, Ministerio de Trabajo y Seguridad Social, Madrid, España.

Stallman, R. (1983).FSF Manifiesto. www.fsf.org

Steinmueller, W. Edward, (2002), Las economías basadas en el conocimiento y las tecnologías de la información ybla comunicación. Pp. 193-210. En Foray, Dominique (Consejero editorial) Revista Internacional de Ciencias Sociales. La Sociedad del Conocimiento, numero 171, Marzo de 2002.

Suárez, C. (2003) Del aprendizaje en red a una red de aprendizaje, *El Tintero*, 3 (10) Universidad Virtual – ITESM. Disponible en: http://www.ruv.itesm.mx/portal/infouv/boletines/tintero/articulos/cristobal.htm

Suman, Michael, El desarrollo del trabajo: Nuevas contradicciones, pp. 83-97 en Juan J. Castillo (editor) El Trabajo del futuro, editorial complutense. Madrid, España.

Tao de la Programación, Traducido por Geoffrey James, Transcripción de Seth Robertson, Versión Española por TESI. Disponible en www.com.pendiente

Thompson 1989, Jugando a ser trabajadores cualificados. Cultura de fábrica y enorgullecimiento por la cualificación laboral entre los obreros del automóvil de Coventry, en Sociología del Trabajo, nueva época, num.7, pp, 105-140;

Toraine, A. (1985), La organización profesional de la empresa, en Freedman, G., Y P. Naville, Tratado de Sociología del Trabajo Tomo I, México FCE pp.384-404

Torrisi, S. (1998), Industrial Organization and Innovation. An International Study of the Software Industry, Edward Elgar, Cheltenham.

Trejo Delarbre, Raúl, (2001), Vivir en la Sociedad de la Información Orden global y dimensiones locales en el universo digital Número 1 / Septiembre - Diciembre 2001 http://www.campus-oei.org/revistactsi/numero1/trejo.htm (IEES-UNAM)

Van Wegberg, M. y P. Berends (2000), "Competing communities of users and developers of computer software: competition between open source software and commercial software". NIBOR Working Paper, mayo INTERNET

Varas C. Marcela, Curso: Gestión de Proyectos de Desarrollo de Software. (Explica la gestión de "construir" el programa de Software). Pp. 1-56 (Pendiente dirección electrónica)

Vigotsky, L. (1988). El desarrollo de los procesos psicológicos superiores. México: Editorial Crítica, Grupo editorial Grijalbo.;

Vigotsky, L. (1995). Pensamiento y lenguaje. Buenos Aires: Ediciones Fausto.

VIGOTSKY, L. (2000) *El desarrollo de los procesos psicológicos superiores.* Barcelona: Crítica.

Villavicencio, Daniel (2004), Trabajo, Relaciones laborales e innovación, pp. 1-32 UAM-X-Cátedra UEALC – FLACSO. 2004. Modulo 2.

Weber, Max, "Concepto de la Sociología y del "significado" en la acción social", en Economía y sociedad,

Wenger, F. (2001) *Comunidades de práctica. Aprendizaje, significado e identidad.* Barcelona: Paidós.

Wertsch, J. (1988) *Vigotsky y la formación social de la mente.* Barcelona: Piadós.

Wiley; Karlson, E., y J. Kolber, A basic Introduction to Y2K: How the year 2000 Computer Crisis Affects You?, Net era publication, Inc.

Yoguel G, Novick M, Milesi D, Roitter S y Borello, J. (2003). Información y conocimiento: la difusión de TICs en la industria manufacturera argentina. Revista de la CEPAL, Santiago de Chile.

Yoguel, G. (2000), "Creación de competencias en ambientes locales y redes productivas", en Revista de la CEPAL, N° 71, Santiago de Chile.

Yoguel, G. (2003), "Innovación y aprendizaje: las redes y los sistemas locales", en Yoguel, G. y Boscherini, F., (2001), "El desarrollo de las capacidades innovativas de las firmas y el rol del sistema territorial.", en Revista Desarrollo Económico, IDES.

Yoguel, G. (2003), "Innovación y aprendizaje: las redes y los sistemas locales", en Grupo de Políticas PyME, "Aportes para una estrategia PyME en la Argentina", CEPAL, Buenos Aires

Yoguel, G. y Boscherini, F., (2001), "El desarrollo de las capacidades innovativas de las firmas y el rol del sistema territorial.", en Revista Desarrollo Económico, IDES.

Yoguel, G.; Borello, J.; Erbes, A.; Robert, V.; Roitter, S. (2004). Competencias tecnológicas de los trabajadores informáticos argentinos. Más allá de las restricciones de demanda y oferta. Littec e-papers.

Yoguel, Gabriel y Mariana Fuchs (2003), Estudios Sobre empleo. Componente D: Desarrollo de redes de conocimiento. Estudio 1.EG.33.3 CEPAL-ONU. Universidad General de Sarmiento. Marzo de 2003, pp.1-67 (pendiente dirección electrónica)

Yoguel, Gabriel, (2004), Información y conocimiento: Las vinculaciones entre difusión de TIC´s y las competencias tecnológicas. Pp. 1-42 Cátedra UEALC – FLACSO. 2004. Modulo 3.

Yourdon E. y J, Yourdon: 1998, Time Bomb 2000, Prentice-Hall;

Zarifian, Philippe. El modelo de competencia y los sistemas productivos. Montevideo: Cinterfor, 1999. Papeles de la oficina técnica, OIT.

Zavala, R. Jesús (2004), La fábrica del Software: Organización de la era de la sociedad del conocimiento. Pp.34 en Análisis Estratégico II Ensayo. Abril de 2004.

Zemelman, Hugo (1992) Los Horizontes de la Razón. II Tomos, CRIM- Anthropos. Madrid.

Zemelman, Hugo. (1997) "Sujetos y subjetividad en la construcción metodológica". En León y Zemelman, (coords.). Subjetividad: umbrales del pensamiento social. Barcelona: Anthropos-CRIM-Coordinación de Humanidades. Pp. 21-35

Zemelman, Hugo; León, Emma, "Sujetos y subjetividad en la construcción metodológica", en Sujetos y subjetividad, No. 6, UAM-X, México 1997, p. 21.

Líneas consultadas en Internet:

Acceso de Internet en México", viernes, 02 de febrero de 2007. Acceso 20 Septiembre de 2007. http://monitorpolitico.com

Aflicción del software" en la programación moderna, en www.oonumerics.org/ . Acceso Junio de 2007

Base de datos significado disponible http://es.wikipedia.org/wiki/Bases_de_datos Acceso 15 de octubre de 2007.

David Garlan. "Software architectures". Presentación en transparencias, disponibles en http://www.sti.uniurb.it/events/sfm03sa/slides/garlan-B.ppt

El marco de referencia empresarial de Zachman se puede acceder desde http://www.software.org/pub/architecture/zachman.asp.

Estándar IEEE 1471-2000 se puede encontrar en http://www.techstreet.com/cgi-bin/detail?product_id=879737

Estándares del Software DoDAF (hogar del C4ISR) en http://www.software.org/pub/architecture/dodaf.asp

Estándares del Software RM-ODP en http://www.dstc.edu.au/Research/Projects/ODP/ref_model.html

Estándares del Software TOGAF en http://www.software.org/pub/architecture/togaf.asp

Firefox, significado disponible en www.mozilla.org

Información de conferencias http://homepages.cs.ncl.ac.uk/brian.randell/NATO/. Acceso 15 Julio de 2007

Ing. en Procesos Discretos y Automáticos significado disponible en http://www.inegi.gob.mx/est/tmp/c20071016151534282.xls Acceso Junio de 2007

La estrategia arquitectónica misma se encuentra delineada en un documento de Michael Platt en http://msdn.microsoft.com/architecture/overview/default.aspx?pull=/library/enus/dnea/html/eaarchover.asp

Las páginas de cabecera de la estrategia de arquitectura de Microsoft están disponibles en http://msdn.microsoft.com/architecture/

Las páginas de Patterns & Practices están en http://www.microsoft.com/resources/practices/completelist.asp

Licencias de software libre http://creativecommons.org/licenses/by-nc-nd/2.5/es/legalcode.es Acceso 16 Junio de 2006

Linux significado disponible . http://es.wikipedia.org/wiki/Linux Acceso 15 de octubre de 2007.

Opensource.org 2004 http:/www.opensource.com

PMI, significado disponible en http://es.wikipedia.org/wiki/Project_Management_Institute Acceso 20.04.07

PSP significado disponible http://es.wikipedia.org/wiki/Personal_Software_Process

Que es al contexto y necesidades de cada organización http://es.wikipedia.org/wiki/RUP

Que es la Guerra del Software http://etsiit.ugr.es/alumnos/mu01/guerraSoftware.html acceso 15.04.07

Revolución de la Inteligencia significado disponible en http://es.wikipedia.org

Software Engineering Institute en la Universidad Carnegie Mellon de Pittsburgh, Pennsylvania (http://www.sei.cmu.edu/ata/ata_init.html).

Software libre significado disponible en http://es.wikipedia.org/wiki/Software_libre Acceso 15 de octubre de 2007.

Páginas consultadas en Internet:

Acceso de Internet en México", viernes, 02 de febrero de 2007. Acceso 20 Septiembre de 2007. http://monitorpolitico.com

Aflicción del software" en la programación moderna, en www.oonumerics.org/ . Acceso Junio de 2007

Base de datos significado disponible http://es.wikipedia.org/wiki/Bases_de_datos Acceso 15 de octubre de 2007.

David Garlan. "Software architectures". Presentación en transparencias, disponibles en http://www.sti.uniurb.it/events/sfm03sa/slides/garlan-B.ppt

El marco de referencia empresarial de Zachman se puede acceder desde http://www.software.org/pub/architecture/zachman.asp.

Estándar IEEE 1471-2000 se puede encontrar en http://www.techstreet.com/cgi-bin/detail?product_id=879737

Estándares del Software DoDAF (hogar del C4ISR) en http://www.software.org/pub/architecture/dodaf.asp

Estándares del Software RM-ODP en http://www.dstc.edu.au/Research/Projects/ODP/ref_model.html

Estándares del Software TOGAF en http://www.software.org/pub/architecture/togaf.asp

Firefox, significado disponible en www.mozilla.org

Información de conferencias http://homepages.cs.ncl.ac.uk/brian.randell/NATO/. Acceso 15 Julio de 2007

Ing. en Procesos Discretos y Automáticos significado disponible en http://www.inegi.gob.mx/est/tmp/c20071016151534282.xls Acceso Junio de 2007

La estrategia arquitectónica misma se encuentra delineada en un documento de Michael Platt en http://msdn.microsoft.com/architecture/overview/default.aspx?pull=/library/enus/dnea/html/eaarchover.asp

Las páginas de cabecera de la estrategia de arquitectura de Microsoft están disponibles en http://msdn.microsoft.com/architecture/

Las páginas de Patterns & Practices están en http://www.microsoft.com/resources/practices/completelist.asp

Licencias de software libre http://creativecommons.org/licenses/by-nc-nd/2.5/es/legalcode.es Acceso 16 Junio de 2006

Linux significado disponible . http://es.wikipedia.org/wiki/Linux Acceso 15 de octubre de 2007.

Opensource.org 2004 http:/www.opensource.com

PMI, significado disponible en http://es.wikipedia.org/wiki/Project_Management_Institute Acceso 20.04.07

PSP significado disponible http://es.wikipedia.org/wiki/Personal_Software_Process

Que es al contexto y necesidades de cada organización http://es.wikipedia.org/wiki/RUP

Que es la Guerra del Software http://etsiit.ugr.es/alumnos/mu01/guerraSoftware.html acceso 15.04.07

Revolución de la Inteligencia significado disponible en http://es.wikipedia.org

Software Engineering Institute en la Universidad Carnegie Mellon de Pittsburgh, Pennsylvania (http://www.sei.cmu.edu/ata/ata_init.html).

Software libre significado disponible en http://es.wikipedia.org/wiki/Software_libre Acceso 15 de octubre de 2007.

Universidad Autónoma Metropolitana
Casa abierta al tiempo
División de ciencias sociales y humanidades
Unidad Iztapalapa

Anexo

I

Guías de entrevistas

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Gerentes

Guía de entrevista para gerentes de empresas desarrolladoras de software

Esta entrevista tiene por objetivo hacer un diagnóstico de las nuevas formas de trabajo que se configuran en el marco de las Tecnologías de la información y Comunicación. La información que aquí se provea será anónima, se utilizará sólo con fines académicos y una vez agrupada y analizada será proporcionada a las empresas encuestadas para que sea utilizada en el diseño de políticas de calidad, eficiencia y productividad. En ningún caso se proporcionará información de un establecimiento en particular, garantizándose la confidencialidad

Nombre de la empresa: _____ Hora de inicio: _____

Entidad y municipio o localidad: _____ Fecha de aplicación: _____

Número de folio: _____ Código de la entrevista: _____

I. LA SITUACIÓN ACTUAL DE LA EMPRESAS DESARROLADORAS DE SOFTWARE.

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1) La empresa y su entorno	Indagar sobre las tres principales razones de éxito de la empresa dentro de la industria del software. En lo relativo a: 1. Entorno de la Tecnología utilizada. 2. Capital humano 3. Capacidades creativas del programador. En caso de no haber éxito, que explique las razones, en función de cómo las "presente" el entrevistado.	¿La tecnología es la adecuada o bien, el pago de patentes o derechos significa una perdida competitiva? ¿El capital humano, posee las clasificaciones adecuadas, los conocimientos suficientes? ¿Los programadores son creativos, poseen las habilidades y destrezas que los proyectos requieren?.	Llevar una lista de las principales herramientas tecnológicas en cuanto A: Lenguajes de programación, B. Plataformas tecnológicas C. Bases de datos. Llevar una lista de las principales capacidades, destrezas y habilidades de los programadores.
2) Calidad y productividad de la empresa.	¿La calidad de los proyectos generados en su empresa son satisfactorios?. ¿En la productividad y calidad en el desarrollo de un proyecto, han sido eficientes los equipos de trabajo, los programadores? ¿Qué factores y en que orden consideraría usted, que debe desarrollarse un proyecto para que sea eficiente? ¿Una metodología de calidad: CMM, SPICE, ISO 900, etc. usted considera que son suficientes para desarrollar proyecto con calidad?	¿Por qué considera que es satisfactoria cuáles son los principales argumentos? ¿Los equipos de trabajo, como se organizan, cuanto duran, quien los elije y como se distribuyen las tareas? ¿Por lo regular, cual es el proceso de calidad en su empresa y como miden la productividad de los programadores? ¿Existen otros factores importantes de calidad y productividad que no se aplican en su empresa?, ¿Por qué no se aplican?, ¿Cuáles son las barreras o impedimentos?	***Presentar una hoja en blanco,*** *Que dibuje un esquema conceptual sobre el proceso del proyecto, así como el de revisión de la calidad y la productividad* ***En este esquema conceptual, hacer hincapié en aquellos "segmentos" donde se concentren los principales problemas de calidad y productividad.***

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
3. Caos organizativo e incertidumbre.	En diversa literatura, se cita que los proyectos de SW constantemente no se entregan a tiempo, no cumplen las expectativas de lo que el cliente solicito, el costo inicial se modifica, etc., esto, porque la industria está en una fase de "caos organizativo" ¿Hasta donde es cierto esto?, ¿Qué perspectivas le ve a esta incertidumbre? ¿Cuáles son los principales factores de este "caos organizativo", de esta incertidumbre?	¿Por qué considera que estos son los factores más importantes de dicho "caos organizativo? ¿Existen otros factores importantes que también han influido en dicha situación? *¿A su empresa como le ha afectado* esta situación?, ¿Qué medidas se han tomado al respecto?.	***Presentar tarjeta 1** sobre los factores que han influido en este "caos organizativo"* *En su hoja de respuestas* ***marque los que el entrevistado señale según el orden e importancia que él o ella señale.*** ***De ser posible, guiela entrevistado a hacer un mapa conceptual***
4. Crisis del Software o aflicción crónica.	Se considera por la literatura especializada que el desarrollo de Software, esta en una fase de "caos organizativo", en un período de fuerte incertidumbre en el desarrollo de proyectos, lo cual en su momento (fines de los setenta) la Ingeniería del Software, le denomino "crisis del Software", y también como "aflicción crónica del software", entre otros conceptos.	Esta crisis se debe a varios mitos y representaciones erróneas de lo que es el proceso de desarrollo del Software: Permitir que el entrevistado plantee desde su perspectiva la existencia o no de esta "crisis" o "aflicción" del Software	***Presentarles el esquema de Gartner de las "Fases de las Empresas de Software"*** ***En este mapa, permitir que el entrevistado plantee su perspectiva de estas etapas.***

II. CONTRATACION Y DIVISION DEL TRABAJO

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. Contratación de programadores	¿Cuándo se contrata a un programador, cuales son los principales requisitos formales: tipo de profesión, ingles, conocimientos tecnológicos, años de experiencia, etc. ?, ¿Y cuales los informales: habilidades, destrezas, creatividad, etc.?.	¿En la empresa, les proporcionan cursos?, ¿El programador participa en la propuesta, en los contenidos de estos cursos?, ¿Por qué?, ¿Los programadores, por iniciativa propia estudian lenguajes de programación, bases de datos o plataformas tecnológicas?, ¿Estos cursos son virtuales o presenciales?	*Orientar la entrevista hacia el tema del* ***conocimiento formal, tácito, y aprendizaje y habilidades no codificada*** *para conocer las* ***razones por que predomina n la contratación de los programadores.***
2. Tipo de contratación	¿En su empresa, como es el tipo de contratación para los programadores?	¿Por qué se elige este tipo de contratación?	*Indagar sobre las ventajas de este tipo de contrataciones.*
3. Contratación de programadores Off-Shore	¿En su empresa han contratado programadores "extras"? ¿En su empresa han contratado programadores "free lance"? (independientes)	¿Por qué se han contratado este tipo de programadores?	*Indagar sobre* ***que es lo que leva a este tipo de contrataciones: El proyecto, el "peso" de la nomina, la especialización de éstos programadores, etc.***.

II. DIVISION DEL TRABAJO

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. Contratación de programadores	¿Si de usted dependiera contratar a los programadores, cuales serían los principales requisitos formales: tipo de profesión, ingles, conocimientos tecnológicos, años de experiencia, etc.?, ¿Y cuales los informales: habilidades, destrezas, creatividad, etc.?.	¿Cuáles son para usted las principales habilidades y destrezas de los programadores de su empresa? ¿Cuáles son las principales debilidades de los programadores?	*Orientar la entrevista hacia el tema del* ***conocimiento formal, tácito, y aprendizaje y habilidades no codificada*** *para conocer las* ***razones por que predomina la contratación de los programadores.***
2. Prototipos de los módulos desarrollados por los programadores	Que opinión tiene del siguiente comentario: Uno de los requisitos para todo programador, con el fin de verificar calidad y eficiencia de las LDC desarrolladas, es generar prototipos en un corto tiempo (Días o semanas) Estos prototipos permiten verificar errores y compatibilidad en las distintas partes del programa, además de la integración entre los distintos módulos del mismo proyecto.	Sin embargo, el desarrollo de estos prototipos encuentra su revés en la carencia de REUTILIZACION de las LDC que componen los distintos módulos del proyecto. ¿Cómo salvar esta contrariedad?; ¿Esto implica que se esta en la "edad" de la piedra en el desarrollo de SW, al no poder reutilizar rutinas o LDC?.	*Orientar al entrevistador, en el hecho de que si existen desarrollo de PROTOTIPOS porque existe un grado importante de INCERTIDUEMBRE en las empresas de SW.* *En que era del Software estamos:* ***Presentar mapa de GARDENER***
3. Innovación en el proceso de desarrollo de un proyecto.	Si en el desarrollo de LDC por el programador, éste debe de hacer uso de sus capacidades creativas, de concentración, de sus habilidades de inventar, de innovar, de sus potenciales cognitivas para conceptualizar las necesidades, los requerimientos del proyecto solicitado por el cliente ¿Qué hace la empresa para motivar esta capacidad de innovación?	¿Se le proporcionan cursos especiales al programador?, ¿El programador participa en la propuesta, en los contenidos de estos cursos?, ¿Los programadores, por iniciativa propia estudian lenguajes de programación, bases de datos o plataformas tecnológicas?, éstos cursos son virtuales o presenciales. ¿Existe resistencia o apatía por parte de los programadores a participar en cursos?	*Presentar un mapa conceptual, de que las empresas desarrolladoras de SW son altamente innovadoras de conocimiento nuevo y útil.*
4. Contratación de programadores Off-Shore	¿Usted ha contratado programadores "extras"? ¿En su empresa han contratado programadores "free lance"? (independientes)	¿Por qué se han contratado este tipo de programadores?	*Indagar sobre* ***que es lo que conduce a este tipo de contrataciones: El proyecto, el "peso" de la nomina, la especialización de éstos programadores, etc.***

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
9. Crisis del Software o aflicción crónica.	Se considera por la literatura especializada que el desarrollo de Software, esta en una fase de "caos organizativo", en un período de fuerte incertidumbre en el desarrollo de proyectos, lo cual en su momento (fines de los setenta) la Ingeniería del Software, le denomino "crisis del Software", y también como "aflicción crónica del software", entre otros conceptos.	Esta crisis se debe a varios mitos y representaciones erróneas de lo que es el proceso de desarrollo del Software: 1. Por tradición, se considera que la tarea de programación es una actividad individual y privada, de modo que muchos programadores tienen poco contacto social y prefieren trabajar en forma individual. ¿Cómo afecta esta concepción en el desarrollo? 2. Los lideres de proyectos, creen que si erraron en los tiempos de entrega, incorporando mas programadores al proyecto lo solucionarán. Lo cual es un error ya que "añadir más gente a un proyecto atrasado lo retrasa aún más" ¿Esta consideración aún tiene vigencia? 3. A medida que un proyecto es más complejo se incrementa la "aflicción crónica del software" ¿Aún existe esta consideración?.	***Presentarles el esquema de Gartner de las "Fases de las Empresas de Software"*** ***Permitir que el entrevistado plantee su perspectiva con respecto a este mapa.*** ***Presentar lista de:*** ***Mitos de la gestión (3);*** ***Mitos del Cliente (2);*** ***Mitos de los programadores (3.*** ***Presentar estos mitos, uno por uno, para que el entrevistado plantee cual es su perspectiva con respecto a estos mitos.***
10. La no Reusabilidad de rutinas.	Usted esta de acuerdo en la siguiente aseveración: A pesar de contar con herramientas de cuarta generación, en la programación de rutinas de código no se reutilizan rutinas. Lo cual implica empezar de cero el proyecto o módulo .	¿Todavía persiste esta acción de iniciar casi de cero las rutinas de programación? ¿Por qué persiste?, ¿Qué llevo a modificar esta aseveración?	

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Lideres de proyecto

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
4. Equipos de trabajo.	¿Los equipos de trabajo, como se organizan, cuanto duran, quien los elije y como se distribuyen las tareas?	¿Los equipos de trabajo están integrados por personal de la empresa? ¿Han participado programadores externos?	***Orientar al entrevistador hacia el ¿Cómo? Se estructuran y organizan los equipos de trabajo.***
5. Importancia de los equipos de trabajo.	Hasta donde esta de acuerdo con la siguiente aseveración: ¿La formación de equipos de trabajo es tan importante que, cuando los proyectos de SW fracasan, generalmente se debe a problemas del trabajo en equipo y no ha aspectos técnicos?	¿Cuáles son las principales razones, por las que un equipo de trabajo puede entra en conflicto?	***Orientar al entrevistador a que aclare, hasta donde los problemas en las entregas a tiempo de un proyecto, se debe a este problema.***
6. Importancia del Cliente en el diseño del proyecto.	Hasta donde esta de acuerdo con la siguiente aseveración: ¿La participación del Cliente es fundamental en el diseño de los requisitos, usabilidad, costos y tiempos de entrega, de tal forma que, cuando los proyectos de SW fracasan, generalmente se debe a problemas de entendimiento con el cliente y no con los equipos de trabajo o con aspectos técnicos?	¿Cuáles son las principales razones, por las que el diseño de un proyecto, podría entrar en contradicciones con el cliente?	***Orientar al entrevistador a que aclare, hasta donde los problemas en las entregas a tiempo de un proyecto, se debe a este problema con el cliente***
7. Programas de Calidad	¿Una metodología de calidad, por ejemplo: CMM, SPICE, ISO 900, MOPROSFT, etc. usted considera que son suficientes para desarrollar proyecto con calidad?, ¿Qué sea eficiente el proceso?.	¿Existen otros factores importantes de calidad y productividad que no se aplican en su empresa?, ¿Por qué no se aplican?, ¿Cuáles son las barreras o impedimentos?	***Orientar al entrevistados, que nos señale –en caso de existir- de que depende la calidad en un proyecto de SW más allá de las metodologías.***
8. Caos organizativo e incertidumbre.	En diversa literatura, se cita que los proyectos de SW constantemente no se entregan a tiempo, no cumplen las expectativas de lo que el cliente solicito, el costo inicial se modifica, etc., esto, porque la industria está en una fase de "caos organizativo" ¿Hasta donde es cierto esto?, ¿Qué perspectivas le ve a esta incertidumbre? ¿Cuáles son los principales factores de este "caos organizativo", de esta incertidumbre?	¿Por qué considera que estos son los factores más importantes de dicho "caos organizativo? ¿Existen otros factores importantes que también han influido en dicha situación? ¿A su empresa como le ha afectado esta situación?, ¿Qué medidas se han tomado al respecto?.	***Presentar tarjeta 1*** *sobre los factores que han influido en este "caos organizativo"* *En su hoja de respuestas* ***marque los que el entrevistado señale según el orden e importancia que él o ella señale.*** ***De ser posible, guíela entrevistado a hacer un mapa conceptual***

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Lideres de proyecto

Guía de entrevista para líderes de proyecto en empresas desarrolladoras de software

Esta entrevista tiene por objetivo hacer un diagnóstico de las nuevas formas de trabajo que se configuran en el marco de las Tecnologías de la información y Comunicación. La información que aquí se provea será anónima, se utilizará sólo con fines académicos y una vez agrupada y analizada será proporcionada a las empresas encuestadas para que sea utilizada en el diseño de políticas de calidad, eficiencia y productividad. *En ningún caso se proporcionará información* de un establecimiento en particular, garantizándose la confidencialidad

Nombre de la empresa: _____	Hora de inicio: _____
Entidad y municipio o localidad: _____	Fecha de aplicación: _____
Número de folio: _____	Código de la entrevista: _____

I. LA ORGANIZACIÓN DEL TRABAJO.

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. El proyecto y y, las etapas	Indagar sobre las tres principales razones de éxito de la empresa dentro de la industria del software. En lo relativo a: 1. Conocimientos formales de los programadores. 2. Aprendizaje, habilidades creativas y destrezas de los programadores. En caso de no haber éxito, que explique las razones, en función de cómo las "presente" el entrevistado.	Si bien es cierto, el uso de tecnología adecuada al proyecto es importante, ¿Que tanto influyen las capacidades creativas, el aprendizaje y habilidades del "como saber hacer" de los programadores?. ¿Los programadores son creativos, poseen las habilidades y destrezas que los proyectos requieren?.	Solicitarle al Entrevistado que nos enumere de mayor a menor importancia el conjunto de habilidades y destrezas requeridas en el proceso de programación. Llevar una lista de las principales capacidades, destrezas y habilidades de los programadores.
2. Los módulos asignados al programador	Si bien es cierto la cantidad de programadores depende del proyecto, ¿Cómo asigna usted las tareas, tiempos de cumplimiento y revisión de la calidad a los programadores?	¿Cuáles son los principales aspectos que considera usted para asignarles tareas a los programadores? ¿Los programadores participan en el diseño de los módulos del proyecto?	***Solicitarle al entrevistado que elabores un MAPA CONCEPTUAL de cómo se asignan las tareas.***
3. Calidad y productividad de la empresa.	¿La calidad de los proyectos que han sido desarrollados en su empresa han sido satisfactorios?. ¿En la productividad y calidad en el desarrollo de un proyecto, han sido eficientes los equipos de trabajo? ¿Qué factores y en que orden consideraría usted, que debe desarrollarse un proyecto para que sea eficiente?	¿Por qué considera que es o, no satisfactoria la calidad en los proyectos? ¿Cuáles son los principales argumentos? ¿Por lo regular, cual es el proceso de calidad en su empresa y como miden la productividad de los programadores?	***Presentar una hoja en blanco,*** *Que dibuje un esquema conceptual sobre el proceso del proyecto, así como el de revisión de la calidad y la productividad*

III. ORGANIZACIÓN DEL TRABAJO

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1) Proceso o etapas en un proyecto en las que intervienen los programadores	Cuándo se solicita el desarrollo de un proyecto de SW por un cliente, cuales son las etapas o fases?, ¿En cuales participa el programador?, ¿En cuales el cliente?	¿Cómo se elige al programador que participara en el diseño del proyecto: Por sus conocimientos, por su antigüedad, por sus habilidades	*Orientar la entrevista hacia el tema de cómo se divide el proyecto y como es la participación del programador y el cliente.* *Si es posible elaborar un mapa conceptual de cuando intervienen.*
2. Intervención de agentes exógenos	En su empresa ha sido necesario, contar con la participación de expertos (académicos, programadores de otras empresas, o incluso de extranjeros en el desarrollo de un proyecto?	¿Cuáles han sido los principales motivos: un proyecto complejo, utilización de lenguajes de programación especializados, mantenimiento de programas no desarrollados por la empresa	***Orientar la entrevista a identificar las causas de participación de agentes externos, así como la posible existencia de "redes".***
3. Eficiencia en la organización del trabajo.	En el proceso de desarrollo de los módulos del proyecto, ¿Considera usted que los programadores son eficientes y competentes en el trabajo realizado?	En su experiencia como gerente ¿Qué es lo que ha visto al respecto? ¿En otras empresas desarrolladoras sucede los mismo?	*Orientar la entrevista hacia la percepción que tiene el entrevistado de la "forma" en que participan los programadores. .*

IV. CULTURA LABORAL

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. utilización de metodologías o "empirismo"	¿En su empresa cuándo un programador desarrolla líneas de código, documenta el desarrollo de estas líneas?,	Por qué considera que no lo hace? ¿Qué hace la empresa para apremiarlo a que adopte esta¿	*Orientar la entrevista hacia el tema de la documentación de las LDC, ya que esta documentación, se utilizará para que otro programador comprenda la lógica del programa o le de mantenimiento.*
2. Criterios de cultura laboral.	Le presentaremos una serie de afirmaciones sobre aspectos fundamentales de la cultura laboral. Nos interesa que nos señale si esta en acuerdo o desacuerdo con cada una de ellas y que nos comente sus razones por las que está de acuerdo o desacuerdo.	¿En su experiencia como gerente o dueño de la empresa ¿Qué es lo que ha visto al respecto?, ¿Se repiten estos "patrones" en otras empresas desarrolladoras de software?	***Presentar una tarjeta los aspectos de la cultura laboral en los programadores de software e indagar sobre las razones por las que está de acuerdo o en desacuerdo***

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
3. Caos organizativo e incertidumbre.	En diversa literatura, se cita que los proyectos de SW constantemente no se entregan a tiempo, no cumplen las expectativas de lo que el cliente solicito, el costo inicial se modifica, etc., esto, porque la industria está en una fase de "caos organizativo" ¿Hasta *donde es cierto esto?*, ¿Qué perspectivas le ve a esta incertidumbre? ¿Cuáles son los principales factores de este "caos organizativo", de esta incertidumbre?	¿Por qué considera que estos son los factores más importantes de dicho "caos organizativo? ¿Existen otros factores importantes que también han influido en dicha situación? ¿A su empresa como le ha afectado esta situación?, ¿Qué medidas se han tomado al respecto?.	***Presentar tarjeta 1** sobre los factores que han influido en este "caos organizativo"* *En su hoja de respuestas* ***marque los que el entrevistado señale según el orden e importancia que él o ella señale.*** ***De ser posible, guíela entrevistado a hacer un mapa conceptual***
4. Crisis del Software o aflicción crónica.	Se considera por la literatura especializada que el desarrollo de Software, esta en una fase de "caos organizativo", en un período de fuerte incertidumbre en el desarrollo de proyectos, lo cual en su momento (fines de los setenta) la Ingeniería del Software, le denomino "crisis del Software", y también como "aflicción crónica del software", entre otros conceptos.	Esta crisis se debe a varios mitos y representaciones erróneas de lo que es el proceso de desarrollo del Software: Permitir que el entrevistado plantee desde su perspectiva la existencia o no de esta "crisis" o "aflicción" del Software	***Presentarles el esquema de Gartner de las "Fases de las Empresas de Software"*** ***En este mapa, permitir que el entrevistado plantee su perspectiva de estas etapas.***

II. CONTRATACION Y DIVISION DEL TRABAJO

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. Contratación de programadores	¿Cuándo se contrata a un programador, cuales son los principales requisitos formales: tipo de profesión, ingles, conocimientos tecnológicos, años de experiencia, etc. ?, ¿Y cuales los informales: habilidades, destrezas, creatividad, etc.?.	¿En la empresa, les proporcionan cursos?, ¿El programador participa en la propuesta, en los contenidos de estos cursos?, ¿Por qué?, ¿Los programadores, por iniciativa propia estudian lenguajes de programación, bases de datos o plataformas tecnológicas?, ¿Estos cursos son virtuales o presenciales?	*Orientar la entrevista hacia el tema del* ***conocimiento formal, tácito, y aprendizaje y habilidades no codificada*** *para conocer las* ***razones por que predomina n la contratación de los programadores.***
2. Tipo de contratación	¿En su empresa, como es el tipo de contratación para los programadores?	¿Por qué se elige este tipo de contratación?	*Indagar sobre las ventajas de este tipo de contrataciones.*
3. Contratación de programadores Off-Shore	¿En su empresa han contratado programadores "extras"? ¿En su empresa han contratado programadores "free lance"? (independientes)	¿Por qué se han contratado este tipo de programadores?	*Indagar sobre* ***que es lo que leva a este tipo de contrataciones: El proyecto, el "peso" de la nomina, la especialización de éstos programadores, etc.***.

III. ORGANIZACIÓN DEL TRABAJO

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1) Proceso o etapas en un proyecto en las que intervienen los programadores	Cuándo se solicita el desarrollo de un proyecto de SW por un cliente, cuales son las etapas o fases?, ¿En cuales participa el programador?, ¿En cuales el cliente?	¿Cómo se elige al programador que participara en el diseño del proyecto: Por sus conocimientos, por su antigüedad, por sus habilidades	*Orientar la entrevista hacia el tema de cómo se divide el proyecto y como es la participación del programador y el cliente.* *Si es posible elaborar un mapa conceptual de cuando intervienen.*
2. Intervención de agentes exógenos	En su empresa ha sido necesario, contar con la participación de expertos (académicos, programadores de otras empresas, o incluso de extranjeros en el desarrollo de un proyecto?	¿Cuáles han sido los principales motivos: un proyecto complejo, utilización de lenguajes de programación especializados, mantenimiento de programas no desarrollados por la empresa	***Orientar la entrevista a identificar las causas de participación de agentes externos, así como la posible existencia de "redes".***
3. Eficiencia en la organización del trabajo.	En el proceso de desarrollo de los módulos del proyecto, ¿Considera usted que los programadores son eficientes y competentes en el trabajo realizado?	En su experiencia como gerente ¿Qué es lo que considera al respecto? ¿En otras empresas desarrolladoras sucede lo mismo?	*Orientar la entrevista hacia la percepción que tiene el entrevistado de la "forma" en que participan los programadores. .*

IV. CULTURA LABORAL

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. utilización de metodologías o "empirismo"	¿En su empresa cuándo un programador desarrolla líneas de código, documenta el desarrollo de estas líneas?,	Por qué considera que no lo hace? ¿Qué hace la empresa para apremiarlo a que adopte esta¿	*Orientar la entrevista hacia el tema de la documentación de las LDC, ya que esta documentación, se utilizará para que otro programador comprenda la lógica del programa o le de mantenimiento.*
2. Criterios de cultura laboral.	Le presentaremos una serie de afirmaciones sobre aspectos fundamentales de la cultura laboral. Nos interesa que nos señale si esta en acuerdo o desacuerdo con cada una de ellas y que nos comente sus razones por las que está de acuerdo o desacuerdo.	¿En su experiencia como gerente o dueño de la empresa ¿Qué es lo que ha visto al respecto?, ¿Se repiten estos "patrones" en otras empresas desarrolladoras de software?	***Presentar una tarjeta los aspectos de la cultura laboral en los programadores de software e indagar sobre las razones por las que está de acuerdo o en desacuerdo***

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
programadores

Guía de entrevista para programadores de proyecto en empresas desarrolladoras de software

Esta entrevista tiene por objetivo hacer un diagnóstico de las nuevas formas de trabajo que se configuran en el marco de las Tecnologías de la información y Comunicación. La información que aquí se provea será anónima, se utilizará sólo con fines académicos y una vez agrupada y analizada será proporcionada a las empresas encuestadas para que sea utilizada en el diseño de políticas de calidad, eficiencia y productividad. En ningún caso se proporcionará información de un establecimiento en particular, garantizándose la confidencialidad

Nombre de la empresa: _____	Hora de inicio: _____
Entidad y municipio o localidad: _____	Fecha de aplicación: _____
Número de folio: _____	Código de la entrevista: _____

I. LA ORGANIZACIÓN DEL TRABAJO.

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. El proyecto y los módulos	Indagar las tres razones que les permiten a los programadores gestionar o generar líneas de código de una manera eficiente. En lo relativo a: 1. Conocimientos formales de los programadores. 2. Conocimientos informales generados a través de redes. 3. Aprendizaje, habilidades creativas y destrezas de los programadores. En caso de que el programador considere que no ha tenido "éxito" en la generación de LDC, que explique las razones, en función de cómo las "presente" el entrevistado.	¿La educación formal le fue suficiente para desempeñar las tareas que le asignan en su trabajo? ¿Cómo ha desarrollado usted su habilidad para desarrollar sus rutinas de programación? ¿Se apoya usted en rutinas ya desarrolladas en otros programas para generar las nuevas rutinas? ¿Se apoya usted en compañeros de la empresa o de otras empresas para gestionar sus rutinas?	Solicitarle al Entrevistado que nos enumere de mayor a menor importancia el conjunto de habilidades y destrezas requeridas en el proceso de generación de rutinas. Llevar una lista de las principales capacidades, destrezas y habilidades de los programadores. Pedirle al entrevistado que nos elabore un "mapa" de cómo logra generar sus rutinas de código.
2. Los módulos asignados al programador	Si bien es cierto la cantidad de programadores depende del proyecto, ¿Cómo le asignan a usted las tareas, tiempos de cumplimiento y revisión de las rutinas de código?	¿Cuáles son los principales aspectos que consideran de usted para asignarle tareas o actividades?. ¿Usted participan en el diseño de los módulos del proyecto?	***Solicitarle al entrevistado que elabores un MAPA de las habilidades, conocimientos o destrezas que consideran de él para asignarle tareas.***

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
3. Calidad y productividad de la empresa.	¿La calidad de los módulos que han sido desarrollados por usted, han sido satisfactorios?. ¿En la productividad y calidad en el desarrollo de un proyecto, han sido eficientes sus compañeros de equipo de trabajo? Esta usted de acuerdo en la siguiente aseveración: La gestión de la calidad es más beneficiosa en la medida en que sea posible incorporar elementos de ella en etapas más tempranas del ciclo de desarrollo del SW. Porque un error de desarrollo entre más cerca de su fuente de origen se detecte será más fácil corregir y por lo mismo el costo de aquella corrección será mucho menor. Porque. Es cierta o falsa esta aseveración.	¿Por qué considera que es o no satisfactoria la calidad en los proyectos? ¿Cuáles son los principales argumentos? ¿Por qué considera que la eficiencia es positiva o negativa en el equipo de trabajo. ¿Por lo regular, cual es el proceso para verificar los errores de calidad que sigue usted?. ¿Cómo mide la eficiencia de las rutinas desarrolladas por usted?.	***Presentar una hoja en blanco,*** *Que dibuje un esquema sobre el proceso de calidad de las rutinas, así como las técnicas que utiliza en la documentación de sus rutinas.*
4. Equipos de trabajo.	Esta usted de acuerdo con la siguiente aseveración: A medida que los proyectos de software son más complejos, se hace más difícil la coordinación de las rutinas generadas por los programadores, porque en los equipos de trabajo se incrementa el numero de programadores. ¿Los equipos de trabajo, como se organizan, cuanto duran, quien los elije y como se distribuyen las tareas?	¿Los equipos de trabajo están integrados por personal de la empresa? ¿Han participado programadores externos? ¿Para usted es difícil trabajar en equipos de trabajo? ¿Por qué?. ¿Usted aprende y desarrolla nuevas habilidades cuando trabaja en equipo?	***Orientar al entrevistador hacia el ¿Cómo? Se estructuran y organizan los equipos de trabajo.***
5. Importancia de los equipos de trabajo.	Hasta donde esta de acuerdo con la siguiente aseveración: ¿La formación de equipos de trabajo es tan importante que, cuando los proyectos de SW fracasan, generalmente se debe a problemas del trabajo en equipo y no ha aspectos técnicos?	¿Cuáles son las principales razones, por las que un equipo de trabajo puede entra en conflicto? ¿Para usted es difícil trabajar en equipos de trabajo? ¿Por qué?. ¿Usted aprende y desarrolla nuevas habilidades cuando trabaja en equipo?	***Orientar al entrevistador a que aclare, hasta donde los problemas en las entregas a tiempo de un proyecto, se debe a este problema.***

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
6. La no Reusabilidad de rutinas.	Usted esta de acuerdo en la siguiente aseveración: A pesar de contar con herramientas de cuarta generación, en la programación de rutinas de código no se reutilizan rutinas. Lo cual implica empezar de cero el proyecto o módulo .	¿Todavía persiste esta acción de iniciar casi de cero las rutinas de programación? ¿Por qué persiste?, ¿Qué llevo a modificar esta aseveración?	
7. Importancia del Cliente en el diseño del proyecto.	Hasta donde esta de acuerdo con la siguiente aseveración: ¿La participación del Cliente es fundamental en el diseño de los requisitos, usabilidad, costos y tiempos de entrega, de tal forma que, cuando los proyectos de SW fracasan, generalmente se debe a problemas de entendimiento con el cliente y no con los equipos de trabajo o con aspectos técnicos?	¿Cuáles son las principales razones, por las que el diseño de un proyecto, podría entrar en contradicciones con el cliente? ¿Usted ha tenido o tiene contacto con los clientes cuando desarrolla las rutinas de programación?	***Orientar al entrevistador a que aclare, hasta donde los problemas en las entregas a tiempo de un proyecto, se debe a este problema con el cliente***
8. Programas de Calidad	¿Usted implementa algún proceso de calidad en sus rutinas? ¿Por qué?. ¿La empresa le incentiva a aplicar algún proceso de calidad en sus rutinas?	¿Existen técnicas de calidad que usted conoce y que no se aplican en su empresa?, ¿Por qué no se aplican?, ¿Cuáles son las barreras o impedimentos?	***Orientar al entrevistados, que nos señale –en caso de existir- de que depende la calidad en un proyecto de SW más allá de las metodologías.***
9. Caos organizativo e incertidumbre.	En diversa literatura, se cita que los proyectos de SW constantemente no se entregan a tiempo, no cumplen las expectativas de lo que el cliente solicito, el costo inicial se modifica, etc., esto, porque la industria está en una fase de "caos organizativo" ¿Hasta donde es cierto esto?, ¿Qué perspectivas le ve a esta incertidumbre? ¿Cuáles son los principales factores de este "caos organizativo", de esta incertidumbre?	¿Por qué considera que estos son los factores más importantes de dicho "caos organizativo? ¿Existen otros factores importantes que también han influido en dicha situación? ¿Usted percibe en su empresa este "caos organizativo"?, ¿Como le ha afectado esta situación?, ¿Qué medidas han tomado al respecto?.	***Presentar tarjeta 1*** *sobre los factores que han influido en este "caos organizativo"* *En su hoja de respuestas* ***marque los que el entrevistado señale según el orden e importancia que él o ella señale.*** ***De ser posible, guiela entrevistado a hacer un mapa conceptual***

10. Crisis del Software o aflicción crónica.	Se considera por la literatura especializada que el desarrollo de Software, esta en una fase de "caos organizativo", en un período de fuerte incertidumbre en el desarrollo de proyectos, lo cual en su momento (fines de los setenta) la Ingeniería del Software, le denomino "crisis del Software", y también como "aflicción crónica del software", entre otros conceptos.	Esta crisis se debe a varios mitos y representaciones erróneas de lo que es el proceso de desarrollo del Software: 1. Por tradición, se considera que la tarea de programación es una actividad individual y privada, de modo que muchos programadores tienen poco contacto social y prefieren trabajar en forma individual. ¿Cómo afecta esta concepción en el desarrollo? 2. Los lideres de proyectos, creen que si erraron en los tiempos de entrega, incorporando mas programadores al proyecto lo solucionarán. Lo cual es un error ya que "añadir más gente a un proyecto atrasado lo retrasa aún más" ¿Esta consideración aún tiene vigencia? 3. A medida que un proyecto es más complejo se incrementa la "aflicción crónica del software" ¿Aún existe esta consideración?.	***Presentarles el esquema de Gartner de las "Fases de las Empresas de Software"*** ***Permitir que el entrevistado plantee su perspectiva con respecto a este mapa.*** ***Presentar lista de:*** ***Mitos de la gestión (3);*** ***Mitos del Cliente (2);*** ***Mitos de los programadores (3.*** ***Presentar estos mitos, uno por uno, para que el entrevistado plantee cual es su perspectiva con respecto a estos mitos.***

II. DIVISION DEL TRABAJO

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. Contratación de programadores	Al momento de contratarle ¿Cuáles fueron los principales requisitos formales: tipo de profesión, ingles, conocimientos tecnológicos, años de experiencia, etc.?, ¿Y cuales los informales: habilidades, destrezas, creatividad, etc.?. Que determinaron su contratación.	¿Cuáles son para usted las principales habilidades y destrezas que los programadores deben de desarrollar para que les contraten? ¿Para usted, cuáles son las principales debilidades de los programadores, como deben de superarlos?	*Orientar la entrevista hacia el tema del* ***conocimiento formal, tácito, y aprendizaje y habilidades no codificada*** *para conocer las* ***razones por que predomina la contratación de los programadores.***
2. Prototipos de los módulos desarrollados por los programadores	Que opinión tiene del siguiente comentario: Uno de los requisitos para todo programador, con el fin de verificar calidad y eficiencia de las LDC desarrolladas, es	Sin embargo, el desarrollo de estos prototipos encuentra su revés en la carencia de REUTILIZACION de las LDC que componen los distintos módulos del proyecto.	*Orientar al entrevistador, en el hecho de que si existen desarrollo de PROTOTIPOS porque existe un grado importante de INCERTIDUEMBRE en las*

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
	generar prototipos en un corto tiempo (Días o semanas). Estos prototipos permiten verificar errores y compatibilidad en las distintas partes del programa, además de la integración entre los distintos módulos del mismo proyecto..	¿Cómo salvar esta contrariedad?; ¿Esto implica que se esta en la "edad" de la piedra en el desarrollo de SW, al no poder reutilizar rutinas?.	*empresas de SW.* *En que era del Software estamos:* ***Presentar mapa de GARDENER***
3. Innovación en el proceso de desarrollo de un proyecto.	En el desarrollo de las rutinas de código, usted, debe acudir a sus capacidades creativas, su habilidad de concentración, echar mano de sus destrezas y habilidades de inventar, de innovar, de sus potenciales cognitivas para conceptualizar las necesidades o requerimientos del proyecto solicitado por el cliente ¿Qué hace la empresa para motivar esta capacidad de innovación en usted?	¿Se le proporcionan cursos especiales? ¿A usted se le incentiva de alguna forma, ya sea económica, puntajes, diplomas, etc.? ¿Usted a recurrido a otros compañeros de la empresa o fuera de ella para conceptualizar las rutinas que necesita?, ¿Cómo se han ayudado?. ¿Usted ha acudido a redes de programadores para conceptualizar las rutinas que necesita?. ¿Cómo se han ayudado?. ¿Existe resistencia o apatía por parte de usted en participar en cursos? ¿Por qué?	*Presentar un mapa conceptual, de que las empresas desarrolladoras de SW son altamente innovadoras de conocimiento nuevo y útil.*
4. Contratación de programadores Off-Shore	¿Usted ha trabajado como programador "extras"? ¿Cómo "free lance"? (independientes). ¿Porque?,	¿Considera que desarrolla nueva habilidades al emplearse como free lance?	

III. ORGANIZACIÓN DEL TRABAJO

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1) Proceso o etapas en un proyecto en las que intervienen los programadores	Cuándo se solicita el desarrollo de un proyecto de SW por un cliente, cuales son las etapas o fases en las que participa usted?	¿Cómo lo eligen a usted para que participara en el diseño del proyecto: Por sus conocimientos, por su antigüedad, por sus habilidades	*Orientar la entrevista hacia el tema de cómo se divide el proyecto y como es la participación del programador y el cliente.* *Si es posible elaborar un mapa conceptual de cuando y como intervienen.*
2. Intervención de agentes exógenos	En el proceso de gestionar rutinas de programación ha sido necesario, contar con la participación de expertos, como académicos, programadores de otras empresas, o incluso de extranjeros en el desarrollo de un proyecto?	¿Cuáles han sido los principales motivos: un proyecto complejo, utilización de lenguajes de programación especializados, mantenimiento de programas no desarrollados por la empresa?	***Orientar la entrevista a identificar las causas de participación de agentes externos, asi como la posible existencia de "redes".***
3. Eficiencia en la organización del trabajo.	En el proceso de desarrollo de los módulos del proyecto, ¿Considera usted que los retrasos o defectos en las rutinas, se debe a que no existen "métodos" o "técnicas" que garanticen una operabilidad eficiente entre los módulos totales que integran al proyecto.	En su experiencia como programador ¿Qué es lo que considera al respecto? ¿En otras empresas desarrolladoras sucede lo mismo?	*Orientar la entrevista hacia la percepción que tiene el entrevistado de la "forma" en que participan los programadote en el desarrollo de rutinas.*

IV. CULTURA LABORAL

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. utilización de metodologías o "empirismo"	Que opina del siguiente comentario: Si nadie observa los métodos que los ingenieros de software utilizan, nadie sabrá como trabajaron. ¿Usted en su empresa cuándo desarrolla líneas de código, documenta el desarrollo de estas líneas?	¿Por qué documenta o porque no lo hace? ¿Qué hace la empresa para apremiarlo a que adopte este procedimiento?	*Orientar la entrevista hacia el tema de la documentación de las LDC, ya que esta documentación, se utilizará para que otro programador comprenda la lógica del programa o le de mantenimiento.*
2. Criterios de cultura laboral.	Le presentaremos una serie de afirmaciones sobre aspectos fundamentales de la cultura laboral. Nos interesa que nos señale si esta en acuerdo o desacuerdo con cada una de ellas y que nos comente sus razones por las que está de acuerdo o desacuerdo.	¿En su experiencia como programador ha observado estos aspectos? ¿Se repiten estos "patrones" en otros programadores de software?	***Presentar una tarjeta los aspectos de la cultura laboral en los programadores de software e indagar sobre las razones por las que está de acuerdo o en desacuerdo***

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Clientes

Guía de entrevista para clientes de empresas desarrolladoras de software

Esta entrevista tiene por objetivo hacer un diagnóstico de las nuevas formas de trabajo que se configuran en el marco de las Tecnologías de la información y Comunicación. La información que aquí se provea será anónima, se utilizará sólo con fines académicos y una vez agrupada y analizada será proporcionada a las empresas encuestadas para que sea utilizada en el diseño de políticas de calidad, eficiencia y productividad. En ningún caso se proporcionará información de un establecimiento en particular, garantizándose la confidencialidad

Nombre de la empresa: _____	Hora de inicio: _____
Entidad y municipio o localidad: _____	Fecha de aplicación: _____
Número de folio: _____	Código de la entrevista: _____

I. LA ORGANIZACIÓN DEL TRABAJO.

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. cumplimiento de los requisitos en el proyecto.	Indagar las tres razones que garantizan satisfacción o malestar por el proyecto entregado. En lo relativo a: 1. El proyecto que usted ha solicitado considera que cumple con los requisitos que usted solicito. 2. Se le entrego en tiempo y bajo los costos acordados. Señale en que fase o etapa del proyecto no se le dio importancia a su participación o atención del programa solicitado.	¿Considera usted que los programadores que participaron en el proyecto, poseen las capacidades, habilidades y conocimientos suficientes?, ¿Por qué?	Solicitarle al Entrevistado que nos enumere de mayor a menor importancia el conjunto de habilidades y destrezas requeridas en un programador. Llevar una lista de las principales capacidades, destrezas y habilidades de los programadores. Pedirle al entrevistado que nos elabore un "mapa" de cómo logra generar sus rutinas de código.
2. Calidad y productividad del proyecto.	¿Esta usted satisfecho con la calidad del programa que se le entrego?. En el tiempo que transcurrió el desarrollo del programa, tuvo contacto con los programadores? ¿Considera importante el contacto con los programadores?, ¿Por qué?	¿Por qué considera que es o no satisfactoria la calidad en el programa? ¿Cuáles son los principales argumentos? ¿Por qué considera que la calidad es positiva o negativa en el programa.	***Presentar una hoja en blanco,*** *Que dibuje un esquema si es que tuvo contacto con los programadores.*

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Clientes

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
3. Importancia del Cliente en el diseño del proyecto.	Hasta donde esta de acuerdo con la siguiente aseveración: ¿La participación del Cliente es fundamental en el diseño de los requisitos, usabilidad, costos y tiempos de entrega, de tal forma que, cuando los proyectos de SW fracasan, generalmente se debe a problemas de entendimiento con el cliente y no con los equipos de trabajo o con aspectos técnicos?	¿Cuáles son las principales razonasen la que han entrado los programadores en contradicciones con usted?- ¿Usted ha tenido o tiene contacto con los clientes cuando desarrolla las rutinas de programación?	***Orientar al entrevistador a que aclare, hasta donde los problemas en las entregas a tiempo de un proyecto, se debe a este problema con el cliente***
4. Caos organizativo e incertidumbre.	En diversa literatura, se cita que los proyectos de SW constantemente no se entregan a tiempo, no cumplen las expectativas de lo que el cliente solicito, el costo inicial se modifica, etc., esto, porque la industria está en una fase de "caos organizativo" ¿Hasta donde es cierto esto?, ¿Qué perspectivas le ve a esta incertidumbre? ¿Cuáles son los principales factores de este "caos organizativo", de esta incertidumbre?	¿Por qué considera que estos son los factores más importantes de dicho "caos organizativo? ¿Existen otros factores importantes que también han influido en dicha situación? ¿Usted percibe en su empresa este "caos organizativo"?, ¿Como le ha afectado esta situación?, ¿Qué medidas han tomado al respecto?.	***Presentar tarjeta 1*** *sobre los factores que han influido en este "caos organizativo"* *En su hoja de respuestas* ***marque los que el entrevistado señale según el orden e importancia que él o ella señale.*** ***De ser posible, guíela entrevistado a hacer un mapa conceptual***
5. Crisis del Software o aflicción crónica.	Se considera por la literatura especializada que el desarrollo de Software, esta en una fase de "caos organizativo", en un período de fuerte incertidumbre en el desarrollo de proyectos, lo cual en su momento (fines de los setenta) la Ingeniería del Software, le denomino "crisis del Software", y también como "aflicción crónica del software", entre otros conceptos.	Esta crisis se debe a varios mitos y representaciones erróneas de lo que es el proceso de desarrollo del Software: 1. Por tradición, se considera que la tarea de programación es una actividad individual y privada, de modo que muchos programadores tienen poco contacto social y prefieren trabajar en forma individual. ¿Cómo afecta esta concepción en el desarrollo? 2. Los lideres de proyectos, creen que si erraron en los tiempos de entrega, incorporando mas programadores al proyecto lo solucionarán. Lo cual es un error ya que "añadir más gente a un proyecto atrasado lo retrasa aún más" ¿Esta consideración aún tiene vigencia?	***Presentarles el esquema de Gartner de las "Fases de las Empresas de Software"*** ***Permitir que el entrevistado plantee su perspectiva con respecto a este mapa.*** ***Presentar lista de:*** ***Mitos de la gestión (3);*** ***Mitos del Cliente (2);*** ***Mitos de los programadores (3.*** ***Presentar estos mitos, uno por uno, para que el entrevistado plantee cual es su perspectiva con respecto a estos mitos.***

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
		3. A medida que un proyecto es más complejo se incrementa la "aflicción crónica del software" ¿Aún existe esta consideración?.	

II. DIVISION DEL TRABAJO

TEMA	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
1. Prototipos de los módulos desarrollados por los programadores	Que opinión tiene del siguiente comentario: Uno de los requisitos para todo programador, con el fin de verificar calidad y eficiencia de las LDC desarrolladas, es generar prototipos en un corto tiempo (Días o semanas). Estos prototipos permiten verificar errores y compatibilidad en las distintas partes del programa, además de la integración entre los distintos módulos del mismo proyecto..	Sin embargo, el desarrollo de estos prototipos encuentra su revés en la carencia de REUTILIZACION de las LDC que componen los distintos módulos del proyecto. ¿Cómo salvar esta contrariedad?; ¿Esto implica que se esta en la "edad" de la piedra en el desarrollo de SW, al no poder reutilizar rutinas?.	*Orientar al entrevistador, en el hecho de que si existen desarrollo de PROTOTIPOS porque existe un grado importante de INCERTIDUEMBRE en las empresas de SW.* *En que era del Software estamos:* ***Presentar mapa de GARDENER***

III. ORGANIZACIÓN DEL TRABAJO

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
7. Eficiencia en la organización del trabajo.	En el proceso de desarrollo de los módulos del proyecto, ¿Considera usted que los retrasos o defectos en las rutinas, se debe a que los programadores no aplican, no implementan "métodos" o "técnicas" que garanticen una operabilidad eficiente entre los módulos totales que integran al proyecto.		

IV. CULTURA LABORAL

TEMAS	PREGUNTAS BÁSICAS	COMPLEMENTARIAS	OBSERVACIONES
8. utilización de metodologías o "empirismo"	Que opina del siguiente comentario: Si nadie observa los métodos que los ingenieros de software utilizan, nadie sabrá como trabajaron. ¿Usted en su trato con los programadores, considera que esta aseveración es cierta?, ¿Por que?		*Orientar la entrevista hacia el tema de la documentación de las LDC, ya que esta documentación, se utilizará para que otro programador comprenda la lógica del programa o le de mantenimiento.*
9. Criterios de cultura laboral.	Le presentaremos una serie de afirmaciones sobre aspectos fundamentales de la cultura laboral. Nos interesa que nos señale si esta en acuerdo o desacuerdo con cada una de ellas y que nos comente sus razones por las que está de acuerdo o desacuerdo.		***Presentar una tarjeta los aspectos de la cultura laboral en los programadores de software e indagar sobre las razones por las que está de acuerdo o en desacuerdo***

Universidad Autónoma Metropolitana
Casa abierta al tiempo
División de ciencias sociales y humanidades
Unidad Iztapalapa

Anexo I I

Tarjetas para entrevista

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Tarjetas para Gerente y Líder de Proyecto

TARJETA 1: FACTORES QUE HAN DETERMINADO "EL ESTADIO ARTESANAL" EN LA INDUSTRIA DEL SOFTWARE"

FACTORES	ORDEN DE IMPORTANCIA
a) Falta de innovación de metodologías sencillas.	☐
b) Falta de formas modernas de organizar los equipos de trabajo.	☐
c) Falta de inversión en capacitación del personal.	☐
d) La alta rotación de los programadores.	☐
e) Falta de motivación del personal.	☐
f) Falta de comprender oportunamente las necesidades del Cliente.	☐
g) Falta de una planificación real y cumplimiento de tareas.	☐
h) Otra (especifique): _____	☐
i) Otra (especifique): _____	☐

TARJETA 2: FACTORES QUE HAN DETERMINADO LA CALIDAD DEL SOFTWARE EN LA INDUSTRIA

FACTORES	ORDEN DE IMPORTANCIA
a) Los bajos salarios en México.	☐
b) La competencia de la India.	☐
c) La falta de programas académicos acorde a las necesidades de la industria.	☐
d) Falta de Integración entre las Universidades y las empresas desarrolladoras.	☐
e) La falta de financiamiento con bajas tasas de interés.	☐
f) La falta de infraestructura competitiva, de calidad, energía y acceso barato.	☐
g) La falta de programadores calificados.	☐
h) Los bajos salarios en India y China.	☐
i) Lo reciente de esta Industria en México, ha implicado que no existan capacidades y habilidades entre los programadores.	☐
j) La existencia de muchas empresas pequeñas.	☐
k). Otros (especifique):	☐
l). Otros (especifique):	☐

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Tarjetas para Gerente y Líder de Proyecto

TARJETA 3: ASPECTOS DE LA CULTURA LABORAL EN LA INDUSTRIA DEL SOFTWARE

ASPECTOS	ACUERDO	DESACUERDO
a) Los programadores son rebeldes y conflictivos	☐	☐
b) Los programadores tienen poca iniciativa, necesitan que les ordenen	☐	☐
c) Los programadores responden mejor cuando se les castiga	☐	☐
d) Los programadores son flojos y tramposos. Por eso no documentan.	☐	☐
e) Los programadores odian a los supervisores y jefes.	☐	☐
f) Los programadores son sucios y descuidados.	☐	☐
g) Los programadores no tienen experiencia laboral.	☐	☐
h) Los programadores tienen pocas habilidades y poca calificación.	☐	☐
i) A los programadores se les dificulta aprender Metodologías de calidad.	☐	☐
j) Los programadores cometen muchos errores.	☐	☐
k) Otro. (Especifique):_____	☐	☐
l) Otro. (Especifique):_____	☐	☐
m) Otro. (Especifique):_____	☐	☐

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
Tarjetas para Gerente y Líder de Proyecto

TARJETA 4: ASPECTOS DE ¿Por qué NO DOCUMENTAN LOS PROGRAMADORES?

ASPECTOS	ACUERDO	DESACUERDO
a) Los programadores NO quieren que los demás conozcan las secuencias de cómo desarrollaron las líneas de código porque pierden poder y control del programa	☐	☐
a) Los programadores NO quieren que los demás conozcan las secuencias de cómo desarrollaron las líneas de código porque dejan de ser importantes e indispensables para la empresa.	☐	☐
a) Los programadores NO quieren que los demás conozcan las secuencias de cómo desarrollaron las líneas de código porque así pueden exigir a la empresa que les pague mas salario.	☐	☐
d) Los programadores son flojos y sienten que es innecesario documentar.	☐	☐
e) Los programadores no les exigen la documentación.	☐	☐
f) Los programadores NO están acostumbrados.	☐	☐
g) La Empresa NO exige la documentación.	☐	☐
h) Los programadores No disponen del tiempo, porque no saben planificar su tiempo.	☐	☐
i) Los programadores No disponen del tiempo, porque la empresa les asigna mucho trabajo.	☐	☐
j) Los programadores reutilizan líneas de código del cual desconocen el "proceso lógico".	☐	☐
k) Otro. (Especifique):_____	☐	☐
l) Otro. (Especifique):_____	☐	☐

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
TARJETA PARA PROGRAMADOR

TARJETA 1: FACTORES QUE HAN DETERMINADO "UN ESTADIO ARTESANAL" EN LA INDUSTRIA DEL SOFTWARE

FACTORES	ORDEN DE IMPORTANCIA
a) Falta de innovación de metodologías sencillas.	☐
b) Falta de formas modernas de organizar los equipos de trabajo.	☐
c) Falta de inversión en capacitación del personal.	☐
d) La alta rotación de los programadores.	☐
e) Falta de motivación del personal.	☐
f) Falta de comprender oportunamente las necesidades del Cliente.	☐
g) Falta de una planificación real y cumplimiento de tareas.	☐
h) Otra (especifique): _____	☐
i) Otra (especifique): _____	☐

TARJETA 2: FACTORES QUE HAN DETERMINADO UNA INEFICIENTE CALIDAD DEL SOFTWARE

FACTORES	ORDEN DE IMPORTANCIA
a) Los bajos salarios en México.	☐
b) La competencia de la India.	☐
c) La falta de programas académicos acorde a las necesidades de la industria.	☐
d) Falta de Integración entre Universidades y empresas desarrolladoras.	☐
e) Falta de financiamiento con bajas tasas de interés.	☐
f) Falta de infraestructura tecnológica competitiva, con calidad y acceso barato.	☐
g) Falta de programadores calificados.	☐
h) Los bajos salarios en India y China.	☐
i) Lo reciente de esta Industria en México, ha implicado que no existan capacidades y habilidades entre los programadores.	☐
j) Existencia de muchas empresas pequeñas.	☐
k). Otros (especifique):	☐
l). Otros (especifique):	☐

Casa abierta al tiempo

Universidad Autónoma Metropolitana. http://www.izt.uam.mx/poes/
Proyecto: Trabajadores del Conocimiento. Proceso de trabajo y cultura laboral entre los programadores de Software.
TARJETA PARA PROGRAMADOR

TARJETA 3: ASPECTOS DE LA CULTURA LABORAL DE LOS PROGRAMADORES.

ASPECTOS	ACUERDO	DESACUERDO
a) Los gerentes o líderes de proyecto no aprecian nuestro trabajo	☐	☐
b) Los gerentes o líderes de proyecto no nos tienen confianza	☐	☐
c) Los gerentes o líderes de proyecto son autoritarios y arbitrarios	☐	☐
d) Los gerentes o líderes de proyecto deberían convivir con los programadores	☐	☐
e) Los gerentes o líderes no comprenden nuestros problemas personales y familiares	☐	☐
f) Los gerentes o líderes de proyecto no escuchan mi opinión.	☐	☐
g) La empresa no nos ofrecen suficiente capacitación e incentivos	☐	☐
h) La empresa sólo quieren ganar dinero y no nos capacita	☐	☐
i) Sólo trabajo porque me pagan. Y espero otra oportunidad mejor.	☐	☐
j) Mi familia es más importante que mi trabajo	☐	☐
k) Hay muchos rencores y odios en la empresa, lo cual genera un ambiente nocivo.	☐	☐
l) En cuanto pueda me consigo otro trabajo	☐	☐
m) No recomendaría a mis amigos trabajar en esta empresa	☐	☐
n) En este empresa pagan poco y exigen mucho	☐	☐
o) Mi trabajo en esta empresa es aburrido	☐	☐
p) Solo quiero aprender y desarrollar mis habilidades, para después buscar otro trabajo.	☐	☐
q) En mi lugar de trabajo hay mucho ruido, polvo, y poca iluminación	☐	☐
r) No me motivan para dar el extra.	☐	☐
s) No me interesa la empresa ni el trabajo.	☐	☐

TARJETA 4: ASPECTOS DE ¿Por qué NO DOCUMENTAMOS LAS LINEAS DE CÓDIGO DEL PROGRAMA?

ASPECTOS	ACUERDO	DESACUERDO
a) Los programadores NO queremos que conozcan las secuencias de cómo desarrollamos las líneas de código porque perdemos Poder y Control del programa	☐	☐
b) Los programadores NO queremos que conozcan las secuencias de cómo desarrollamos las líneas de código porque dejamos de ser importantes e indispensables para la empresa.	☐	☐
c) Los programadores NO queremos que los demás conozcan las secuencias de cómo desarrollamos las líneas de código porque así podemos exigir a la empresa que nos incremente el salario.	☐	☐
d) Los programadores No documentamos por que las líneas de código es una "innovación" que nos pertenece y, al no pagárnosla la empresa, preferimos no documentar.	☐	☐
e) Como no nos exige la empresa documentar, pues no lo hacemos.	☐	☐
f) Como NO estamos acostumbrados a documentar, pues no lo hacemos.	☐	☐
g) Como la empresa NO nos paga la documentación, pues no lo hacemos.	☐	☐
h) NO documentamos, porque la empresa nos asigna mucho trabajo.	☐	☐
i) NO documentamos, porque reutilizamos líneas de código que desconocemos el "proceso lógico".	☐	☐
j) No documentamos, porque de esa manera aseguro mi trabajo en la empresa.	☐	☐
k) Otro. (Especifique):_____	☐	☐
l) Otro. (Especifique):_____	☐	☐

Universidad Autónoma Metropolitana
División de ciencias sociales y humanidades
Unidad Iztapalapa

Anexo III

Tipos y metodologías del Software

Según autores como Meyer (1997), Cuevas (1991) Norris y Rugby (1997), las categorías genéricas para la aplicación del Software son difíciles de establecer aún más las fronteras de las aplicaciones, sin embargo se puede aventurar una categorización de los tipos potenciales de aplicación de Software:

1° Software de sistemas.

- Es una colección de programas escritos para servir a otros programas.
- El área del software de sistemas se caracteriza por:
  - La fuerte interacción con el hardware de la computadora
  - Su gran utilización por múltiples usuarios
  - La operación concurrente que requiere planificación
- Compartimentación de recursos.
  - Sofisticada gestión de procesos
  - Estructura de datos complejos
- Múltiples interfaces externas.

2° Software de tiempo real.

- Es el Software que mide, analiza, controla sucesos del mundo real conforme ocurren, se llama de tiempo real.
- Los elementos del software de tiempo real incluyen:
  - Una componente de acumulación de datos que recolecta y formatea la información de un entorno externo
  - Una componente de análisis que transforma la información según requiera la aplicación
  - Una componente de control/salida que responda al entorno externo
  - Una componente de monitorización que coordina a todas las demás componentes de forma que puedan mantener la respuesta en tiempo real (típicamente en el rango de 1 milisegundo a 1 segundo).

3° Software de gestión.

- Se refiere al procesamiento de la información comercial.
- Los sistemas discretos han evolucionado hacia el software de sistemas de información de gestión, que acceda a una o mas bases de datos grandes que contienen la información comercial.
- Características de las aplicaciones en esta área:
  - Reestructuran los datos existentes en orden a facilitar las operaciones comerciales o gestionar la toma de decisiones
  - Realizan cálculo interactivo: Por ejemplo el procesamiento de transacciones en puntos de venta.

4° Software de ingeniería y científico.

- Se ha caracterizado por los algoritmos de manejo de números de carácter científico.
- Las nuevas aplicaciones del área de ingeniería/científica se han alejado de los algoritmos convencionales numéricos.

- El diseño asistido por computadora (CASE), la simulación de sistemas y otras aplicaciones interactivas, han comenzado a tomar características del software de tiempo real e incluso de sistemas

5° Software empotrado (embebido).

- Se utiliza para controlar productos y sistemas de los mercados industriales y de consumidores.
- Características:
  - Reside en memoria de sólo lectura.
  - Puede ejecutar funciones muy limitadas, por ejemplo el control digital de un horno de microondas, bombas de gasolina, sistemas de frenado de los automóviles, etc.
  - Suministrar una función significativa y única.
  - Capacidad de control.

6° Software de computadoras personales.

- Es uno de los diseños del software más innovativos en el campo del software.
- Entre sus múltiples aplicaciones esta:
  - Tratamiento de textos
  - Hojas de calculo
  - Gráficos
  - Entretenimientos
  - Gestión de base de datos
  - Aplicaciones financieras, comerciales y personales
  - Redes externas o acceso a base de datos

7° Software basado en la Web.

- Las páginas web que se despliegan después de buscarlas en un explorador es software que incorpora instrucciones ejecutables (CGI, HTML, PERL, JAVA, etc.). En esencia Internet es un amplio consumidor/proveedor de software.

8° Software de inteligencia artificial.

- Hace uso de algoritmos no numéricos para resolver problemas complejos que no son adecuados para el cálculo o análisis directo.
- Actualmente el área mas activa es la de los sistemas expertos, también llamados sistemas basados en el conocimiento.
- Otras áreas de aplicación:
  - Reconocimiento de patrones.
  - Prueba de teoremas.
  - Juegos virtuales.

Para comprender el contexto de la aflicción del software es importante conocer los mitos que se han configurado alrededor del proceso de trabajo.

Mitos del Software

Muchas de las causas de la aflicción crónica del software pueden ser encontradas en una mitología crónica que surge durante los primeros años del desarrollo del software. Los mitos del software propagaron información errónea y confusión. Los mitos del software tienen varios atributos que los hacen insidiosos: Aparecen como declaraciones de hechos; tuvieron un sentido intuitivo; frecuentemente fueron promulgados por expertos que "estaban al día"; surgen en los primeros años del desarrollo. Citemos los más importantes.

Mitos del Gestor de proyectos.

Los gestores de proyectos están normalmente bajo la presión de cumplir con los presupuestos, vigilar que no se retrase el proyecto y mejorar la calidad. El gestor se sujeta de un mito del software, aunque tal creencia sólo disminuya la presión temporalmente.

Mito: ¿Por qué debemos cambiar nuestra forma de desarrollar el Software?, si estamos haciendo el mismo tipo de programación ahora, que hace diez años.

Realidad: Aunque el dominio de la aplicación puede ser el mismo, la demanda de una mayor productividad y calidad, y el papel critico del software en objetivos comerciales estratégicos, ha aumentado sustancialmente.

Mito: Tenemos un libro de estándares y procedimientos para construir software.

Realidad: ¿Pero se usa?,¿conocen los trabajadores su existencia?,¿refleja las practicas modernas en desarrollo del software?,¿es completo?. En muchos casos la respuesta a todas estas preguntas es no.

Mito: Nuestra gente dispone de las herramientas de desarrollo de software más avanzadas, después de todo les compramos las computadoras mas nuevas.

Realidad: Se necesita mucho más que el último modelo de computadora, herramientas de software, las cuales son mucho más importantes que el hardware para conseguir buena calidad y productividad.

Mito: Si fallamos en la planificación podemos añadir más programadores y adelantar el tiempo perdido.

Realidad: El desarrollo de software no es un proceso mecánico como la fabricación. Añadir gente a un proyecto software retrasado lo retrasa aun más. Cuando se añaden nuevas personas, la necesidad de aprender y comunicarse con el equipo puede y hace que se reduzca la cantidad de tiempo gastado en el desarrollo del producto. Puede añadirse gente, pero sólo de una manera planificada y bien conocida.

Mitos del cliente.

Un cliente que solicita una aplicación software puede ser interno a la compañía o una compañía exterior. El cliente cree en los mitos que existen sobre el software debido a que los gestores y trabajadores responsables hacen muy poco para corregir la mala información. Los mitos conducen a que el cliente se cree una falsa expectativa y finalmente, quede insatisfecho con el desarrollo del software.

Mito: Una declaración general de los objetivos es suficiente para comenzar a escribir los programas, podemos dar los detalles más adelante.

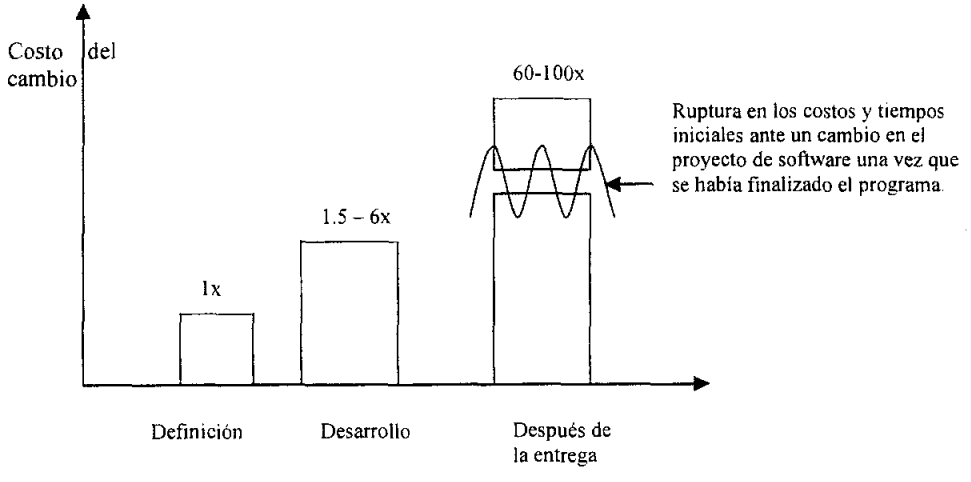
Realidad: Una mala definición inicial es la principal causa del trabajo baldío en software. Una descripción formal y detallada del dominio de la información, funciones, rendimiento, interfaces, ligaduras de diseño y criterios de validación es esencial. Estas características pueden determinarse sólo después de una exhaustiva comunicación entre el cliente y el analista.

Mito: Los requerimientos del proyecto cambian continuamente, pero los cambios pueden acomodarse fácilmente ya que el software es flexible.

Realidad: Es verdad que los requisitos iniciales señalados por el cliente cambian, pero, el impacto varia según el momento en que se introduzca. (Ver Figura 4)

Si el cliente define sus necesidades y el desarrollador capta el ¿Qué? es lo que desea el cliente y resuelve acertadamente el ¿Cómo? resolver la codificación de lo que se solicita, se llegara a un acuerdo final, una definición inicial (1x), así, los cambios que solicite el cliente o el gestor del proyecto pueden pronto acomodarse fácilmente, con relativamente poco costo. Cuando los cambios se solicitan durante la etapa del diseño-desarrollo, el impacto en el costo crece considerablemente (1.5 a 6x); empero cuando se solicita al final de un proyecto, cuando este ya se entrego al cliente y éste decide que no cumple lo "que yo quería" y solicita nuevas tareas no definidas desde el inicio, los cambios pueden producir un orden de magnitud más caro que el mismo cambio pedido al principio (Ver figura 4).

Figura 4
Impacto del cambio en los requerimientos del Software



Fuente: Pressman:2002:9 Figura 1.3

Mitos de los desarrolladores.

Los mitos en los que aún creen muchos desarrolladores se han fomentado durante cuatro décadas de cultura Informática.

Mito: No hay realmente ningún método para el análisis, diseño y prueba que funcione bien, el desarrollador simplemente va a la computadora y empieza a codificar.

Realidad: Existen en la industria métodos comprobados para el diseño, análisis y prueba, ninguno es infalible, pero el uso de una metodología para el desarrollo del software está implícito en todos ellos.

Mito: Una vez que se capturan las líneas de codificación del programa y este funcione, el trabajo ha terminado.

Realidad: Mientras más pronto se comience a escribir código, más se tarda en terminarlo. El desarrollo del software abarca tres actividades: Definición; Desarrollo; Mantenimiento. Según datos industriales señalados por Lientz (1980), Jones (1991), Putnam (1997), entre otros, indican que entre 50% y 70% de todo el esfuerzo dedicado a un programa se realizara después de que se le haya entregado al cliente por primera vez.

Mito: Hasta que no tengo el programa ejecutándose, realmente no tengo forma de establecer calidad, eficiencia y requerimientos.

Realidad: Uno de los mecanismos más efectivos para garantizar la calidad del software puede aplicarse desde el principio de un proyecto, la revisión estructurada (Walktroug). La revisión del software es filtro de calidad que se ha comprobado que es más efectivo que la prueba, para encontrar ciertas clases de defectos en el software

Mito: Lo único que se entrega al terminar el proyecto es el programa funcionando.

Realidad: El programa es solo una parte de una configuración del software, que incluye muchos elementos. Por ejemplo, la documentación proporciona bases para un buen desarrollo y para proporcionar guías para la tarea de mantenimiento.

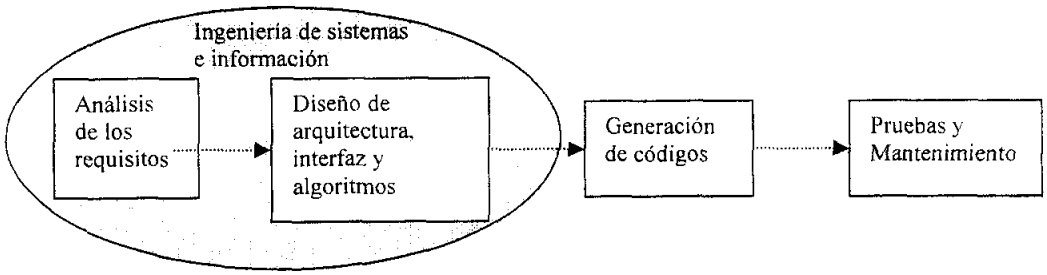
Mito: Una vez que el Software se está usando, el mantenimiento es mínimo y puede manejarse sobre la base de hacerlo como se pueda.

Realidad: La mitad de un presupuesto se gasta en mantenimiento, por tanto el mantenimiento del software debe de: organizarse, Planificarse y Controlarse.

Estos mitos y sus realidades, son importantes porque persisten hoy en día, y una de las hipótesis del presente trabajo, es que la superación de las realidades aquí expuestas entre Gestor-Cliente-Desarrollador, conducirá a destrabar la denominada "aflicción crónica del Software".

Metodologías o Paradigmas en el Software.

Modelo lineal secuencial: también llamado modelo secuencial, sugiere un enfoque sistemático, que comienza con un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento:

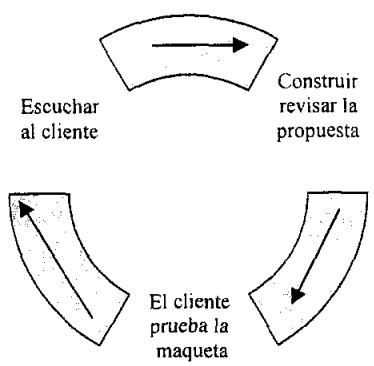


Modelo de construcción de prototipos: Un Cliente a menudo define un conjunto de objetivos generales para el proyecto que solicita, pero en numerosas ocasiones no identifica los requisitos detallados de entrada, del proceso y salida. En otras ocasiones el gestor del proyecto, puede no estar seguro de los procedimientos codificados (algoritmos), de la capacidad de adaptación del sistema operativo o de la interacción desarrollador-computadora. En este caso este modelo es el adecuado, ya que comienza con la recolección de los requisitos, acto en el cual el desarrollador y el cliente encuentran y definen conjuntamente los objetivos globales del proyecto; el cual se traduce en un "diseño rápido" de los requisitos visibles del cliente-usuario (por ejemplo requisitos de entrada y salida). El prototipo lo evalúa el cliente-usuario y se utiliza para afinar el software que se desarrollará.

En esta propuesta, según Brooks (1975) el primer sistema funciona muy lento, es grade y torpe. Y se debe comenzar de nuevo, por tanto es problemático este modelo, ya que el cliente ve lo que parece ser una versión del trabajo deseado, y no sabe que con el fin de "terminar para hacer una prueba" no se tiene cuidado en la calidad o la facilidad para darle mantenimiento.

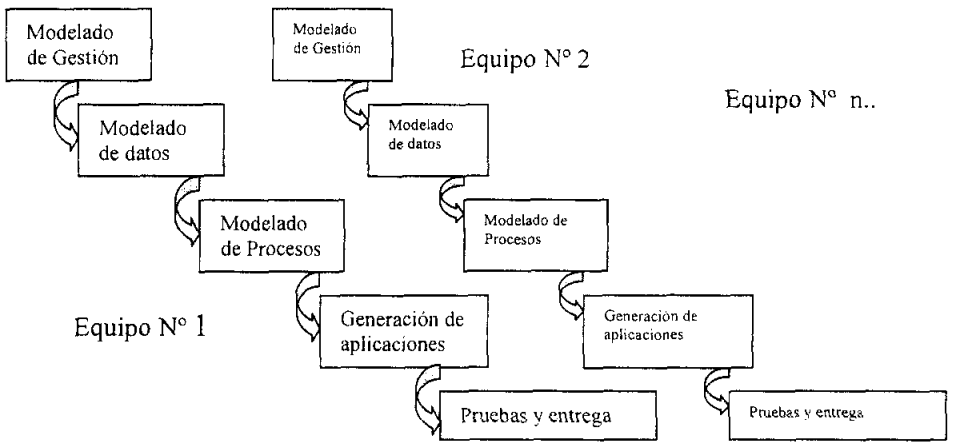
Por tanto debe de hacerse correcciones y el cliente no lo entiende y a su vez el cliente "pide pequeños ajustes" lo cual altera la líneas de algoritmos especificados, lo cual hace lenta una segunda versión. Por otro lado, el desarrollador con las prisas de implementar el prototipo hace uso de "trucos" en sistemas operativos o lenguajes inadecuados. Sin embargo aún así podría ser una alternativa este modelo, siempre y cuando los actores se pongan de acuerdo en definir que el prototipo sólo es para definir los requisitos.

Gráfica del Modelo de Prototipos



Modelo de Desarrollo Rápido de Aplicaciones (DRA): Es un proceso de desarrollo lineal secuencial que enfatiza un ciclo de desarrollo corto. El modelo DRA, según Kerr (1994) y Martin (1991) es una adaptación del modelo lineal secuencial que permite desarrollar un proyecto completamente funcional en 60-90 días. Es una construcción basada en componentes que se utiliza para aplicaciones de sistemas de información. Comprende lo siguiente:

Gráfica del Modelo DRA



Este modelo enfatiza la reutilización de componentes que ya han sido comprobados, sin embargo deben probarse los componentes nuevos y todas las interfaces. Este modelo puede modularse, donde cada fase no tarde más de tres meses. Las interfaces pueden ser desarrolladas por diferentes equipos por separado e integradas en un solo conjunto.

Modelos evolutivos de proceso de software: Esta propuesta es para sistemas complejos donde los requisitos de gestión y de productos cambian conforme el desarrollo, lo cual implica que quizá al final el producto no sea el esperado, debido a las modificaciones constantes. Por lo tanto, se debe introducir una versión limitada para cumplir con la presión del tiempo, es decir los ingenieros del Software necesitan un modelo de proceso que sea diseñado explícitamente para acomodarse a un producto que evolucione con el tiempo.

Recordemos que los modelos anteriores como el lineal-secuencial está pensado que se va a entregar un sistema completo, un producto terminado. El de prototipos, se diseña para ayudar al cliente a comprender los requisitos.
En general no se considera la naturaleza evolutiva del Software. Los modelos evolutivos son iterativos. Se caracterizan por permitir a los ingenieros a desarrollar versiones cada vez más completas de Software. Se proponen tres modelos con estas características: Modelo Incremental; Modelo espiral; Modelo espiral WIN WIN (victoría-victoría); Modelo de desarrollo concurrente, entre otros.

Modelo basado en tecnologías de Objetos: Esta propuesta proporciona el marco de trabajo técnico para generar un modelo basado en componentes. Este paradigma enfatiza la creación de clases que encapsulan tanto los datos como los algoritmos que se utilizan para manejar los datos. Si se diseñan e implementan adecuadamente, las clases orientadas a objetos son reutilizables por las diferentes aplicaciones y arquitecturas de sistemas informáticos.

Este modelo permite reducir en un 70% los tiempos de desarrollo, 84% los costos del proyecto y la productividad mejora en un 26.2% (Yourdon:1994. Con estos datos se comprueba la robustez de esta propuesta de ensamblar los componentes. Para lograr esta ventaja competitiva en los desarrolladores de Software, Jacobson y Booch y Rumbaugh (1999) proponen un proceso unificado de desarrollo de software a partir de utilizar un solo lenguaje de programación denominado Lenguaje de Modelado Unificado (UML). Este UML define los componentes que se utilizarán para construir el sistema y las interfaces que conectarán a los componentes.

Modelo de métodos formales: comprende un conjunto de actividades que conducen a la especificación matemática del software. Estos métodos permiten al ingeniero de software especificar, desarrollar, y verificar un sistema. Algunas organizaciones están aplicando una derivación de este método denominado Ingeniería de Software de sala limpia (Mills, Dyer, Linger:1987). Con este método de matemáticas puras se eliminan muchos problemas como ambigüedad, inconsistencias, etc. El ingeniero con este modelo puede detectar errores, se postula que se podrán gestionar programas libres de errores. Empero se requiere personal altamente capacitado y es difícil la comunicación con los clientes por parte de los desarrolladores.

Técnicas de cuarta generación (T4G): Estas propuestas abarcan un amplio espectro de herramientas de software que tienen como eje la facilidad otorgada al ingeniero de software para la especificación de las características de un proyecto a alto nivel. Las herramientas que se desarrollan, generan automáticamente las líneas de código fuente basándose en las especificaciones del técnico.

En otras palabras, parece ser que cuanto mayor sea el nivel de especificidad del software, más rápido podrá construirse el programa. Este paradigma se orienta hacia la posibilidad de especificar el software utilizando formas de lenguaje especializado o notas graficas que describa el problema que hay que resolverle al cliente. Estas técnicas de cuarta generación cuando se combinan con enfoques de ensamblaje de componentes puede convertirse en el enfoque dominante hacia el desarrollo del software.

Tecnologías de proceso: Los modelos de proceso citados arriba se deben de adaptar para utilizarse eficientemente por los equipos de trabajo. Para ello se han desarrollado herramientas poderosas de tecnologías de procesos para analizar los procesos actuales, organizar las tareas de trabajo, controlar y supervisar el progreso y gestión de la calidad y productividad técnica ((Bandinelli:1994).

Las herramientas de tecnologías de proceso, permite que una empresa desarrolladora de software construya un modelo automatizado del marco de trabajo común del proceso, del conjunto de tareas y actividades. Una vez organizado el proceso, se puede utilizar otras herramientas de tecnologías de procesos para asignar, supervisar y controlar todas las tareas del modelo de proceso. Cada uno de los desarrolladores del proyecto puede desarrollar herramientas de control de las tareas, cumplimiento y garantía de calidad. Todo equipo de programación debe tener una organización interna. Esta depende del proyecto. Sin embargo, pueden identificarse tres estructuras básicas: Grupos democráticos; Grupos con jefes; grupos bajo jerarquía administrativa.

Grupos democráticos: equipos sin egoísmo; mi programa -nuestro programa; metas y decisiones por consenso; liderazgo rotativo en función de tareas y capacidades; los productos son discutidos por todos los miembros.
Ventajas: cada miembro contribuye a las decisiones; los miembros aprenden unos de otros; existe gran comunicación; no existe presión individual pero si para todo el equipo.
Inconvenientes: una considerable cantidad de comunicación para todas las decisiones; necesidad de que todos trabajen juntos y coordinadamente; falta de actividad y responsabilidad de manera individual; menor iniciativa.
Adecuado: para proyectos investigación, para desarrollos largos y difíciles.

Grupos con jefe: son grupos estructurados; hay liderazgo; existen funciones claras.
Ventajas: decisiones centralizadas; reducción de comunicaciones; sirve para capacitar.
Inconvenientes: enfoques parciales; sensibilidad a capacidades técnicas y administrativas del jefe.
Adecuados: aplicaciones de procesamiento de datos: paquetes (tipo financiero); jefe experimentado; programadores poco calificados; capacitación en ambientes reales.

Grupos bajo jerarquía administrativa: ocupa posición intermedia entre los dos anteriores; hay un líder del proyecto; existen funciones claras; se requiere mucha administración/coordinación.
Ventajas: semi-descentralización en la toma de decisiones; flexibilidad en la comunicación.
Inconvenientes: proyección de buenos programadores (técnicos) a puestos administrativos, Buen nivel técnico no implica una buena administración.

En estos grupos las funciones son:
Jefe: Planifica, Coordina, Revisa actividades técnicas, Diseña el producto, Instrumenta partes críticas, Toma decisiones importantes, Asigna trabajo programadores.
Respaldo: Ayuda al jefe en sus actividades; Lo reemplaza cuando sea necesario, realiza tareas en general bajo supervisión del jefe.
Técnicos: de 2 a 5 programadores, codifica, Depuran, Documentan, Prueban.
Bibliotecario: Actúa como controlador y coordinador de la configuración del software, Documenta, hace listados fuentes, procesa datos, cataloga e indexa módulos, ayuda a los miembros de los equipos a Investigar, evaluar, Preparar documentos, etc. En un proyecto debe existir un bibliotecario, pues el controla la documentación, y la coordinación de los programadores.

Una estructura aceptada por la mayoría de los gestores de proyectos de software lo componen los siguientes participantes:

- Gestores superiores: que definen los aspectos de negocios que a menudo tienen una significativa influencia en el proyecto.
- Gestores del proyecto (Técnicos): Deben de planificar, motivar, organizar y controlar a los profesionales que realizan el trabajo de Software.
- Profesionales del Software: proporcionan las capacidades técnicas necesarias para la ingeniería de un producto o aplicación.
- Clientes: especifican los requisitos para la ingeniería del software y otros elementos que tienen menor influencia en el resultado.

- Usuarios finales: interaccionan con el software una vez que se ha entregado la producción.

Para ser eficaz, el equipo del proyecto debe organizarse de manera que maximice las habilidades y capacidades de cada persona. Y este es el trabajo del jefe de equipo.
Los jefes de equipo: La gestión de un proyecto es una actividad intensamente humana, y por esta razón deben de poseer capacidades en la solución de problemas, debe tener la habilidad de la concentración en entender problemas que deben resolverse, saber generar empatías en el flujo de ideas, y hacerles saber con hechos a los miembros del equipo que la calidad no debe verse comprometida.

Los Equipos de Software: Según vimos en los paradigmas del Software existen muchas formas de organizar el personal. Una vez que el gestor de proyecto elige determinada estructura organizativa del personal ya no puede modificarlo. La mejor estructura depende del estilo de gestión de una organización, número de personas que compondrá el equipo, niveles de preparación y dificultad general del problema. Mantei (1981) propone tres organigramas de equipos genéricos: Descentralizado democrático; Descentralizados controlado y Centralizado controlado. Por su lado, Constantine (1993) propone cuatro "paradigmas de organización" para los equipos de ingeniería: Paradigma cerrado; Paradigma aleatorio; Paradigma abierto y Paradigma sincronizado.

La coordinación y comunicación: Los proyectos de software pueden tener problemas críticos por diversas razones. El tamaño o escala de muchos esfuerzos condiciendo a complejidades, confusiones y dificultades significativas para coordinar a los miembros de un equipo de software. El grado de incertidumbre da como resultado un continuo flujo de cambios que impactan al equipo. La interoperatividad se ha convertido en una característica clave en muchos sistemas.

Estas características (escala, incertidumbre e interoperatividad) propias del software moderno deben ser abordadas a través de metodologías efectivas que coordinen los equipos de trabajo y se establezcan al interior de estos, una comunicación formal e informal entre los miembros del equipo y entre múltiples equipos. La comunicación se establece por escrito, en reuniones organizadas y a través de otros medios informales más personales, menos interactivos e impersonales.