Universidad Autónoma Metropolitana

Unidad Iztapalapa - División de Ciencias Básicas e Ingeniería

Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles

T E S I S

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS
(CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN)

PRESENTA:

LIC. JOSÉ FIDEL URQUIZA YLLESCAS

ASESORES:
M. EN C. ALFONSO MARTÍNEZ MARTÍNEZ
M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ

JURADO CALIFICADOR:

Presidente: DRA. HANNA JADWIGA OKTABA (UNAM)
Secretario: ING. LUIS FERNANDO CASTRO CAREAGA (UAM-I)
Vocal: M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ (UNAM)

México, D.F., Julio 2013

A mis queridas y amadas antecesoras,
mi madre y mi abuela:
Patricia Lorena Yllescas Hernández y
Rosa Marina Hernández Mojica vda. de Yllescas
Honrando su ejemplo de trabajo y dedicación.

Agradecimientos

A mis asesores Alfonso Martínez Martínez y Guadalupe Ibargüengoitia González.
Por haber aceptado ser mis asesores de tesis, compartir ideas, interés, tiempo, consejos, paciencia, trabajo, dedicación, empeño para desarrollarme como una mejor persona tanto en el ámbito docente como en el ámbito profesional y todo su apoyo para terminar este trabajo.

A mis sinodales Luis Fernando Castro Careaga y Hanna Jadwiga Oktaba.
Por la revisión de la tesis, la confianza que depositaron en mí y enseñarme los aportes de la ingeniería de software.

A las y los maestros del Posgrado en Ciencias y Tecnologías de la Información.
Reyna Carolina Medina Ramírez, Elizabeth Pérez Cortés, Víctor Manuel Ramos Ramos, Sergio De los Cobos Silva, Miguel Ángel Gutiérrez Andrade y Ricardo Marcelín Jiménez, con quienes tuve el privilegio de tomar clase y quienes me compartieron sus conocimientos.

A Blanca Rosa Pérez Salvador y Susana Orozco Segovia.
Por todo el apoyo, confianza y cariño que siempre me han dado.

A Alfonso Prieto Guerrero.
Por haberme apoyado en momentos importantes en mi formación y su interés por obtener el grado de maestro en ciencias.

A mis padres y mi abuela.
Patricia Lorena, Humberto y Rosa Marina, quienes amo y que siempre me han motivado, aconsejado, apoyado e inspirado a continuar estudiando para lograr mis metas.

A mis hermanas.
Patricia Gabriela y Mariana Patricia, quienes adoro y han estado presentes en mi vida.

A mi tía y tío.
Ligia Teresa y Juan Martín, quienes han estado pendientes de mí pese a la distancia.

A mis amigas y amigos.
Citlali, Verónica, Adriana, Elisa, Dolores, Sonia, Yessica, Jazmin, Fabián, Alfonso, Ismael, Rosendo, Fernando, Óscar, Alberto e Hino quienes han estado conmigo desde hace tiempo y en los que siempre he podido confiar.

A Erick Andrey Serratos Álvarez.
Por facilitarme los Paquetes de Puesta en Operación que fueron de gran utilidad para esta tesis.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT).
Por el apoyo económico en los estudios de maestría.

"Solo sé que no sé nada"
Sócrates.

Hoja de datos

1. Datos del Alumno
   Lic. en C. C. Urquiza Yllescas
   José Fidel
   Universidad Autónoma Metropolitana
   Unidad Iztapalapa
   Ciencias Básicas e Ingeniería
   Posgrado en Ciencias y Tecnologías de la Información

2. Datos del asesor
   M. en C.
   Martínez Martínez
   Alfonso Universidad Autónoma Metropolitana
   Unidad Iztapalapa
   Ciencias Básicas e Ingeniería
   Posgrado en Ciencias y Tecnologías de la Información

3. Datos de la asesora
   M. en C.
   Ibargüengoitia González
   María Guadalupe Elena
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Departamento de Matemáticas
   Ciencias de la Computación

4. Datos de la sinodal
   Dra.
   Oktaba
   Hanna Jadwiga
   Universidad Nacional Autónoma de México
   Facultad de Ciencias
   Departamento de Matemáticas
   Ciencias de la Computación

5. Datos del sinodal
   Ing.
   Castro Careaga
   Luis
   Universidad Autónoma Metropolitana
   Unidad Iztapalapa
   Ciencias Básicas e Ingeniería
   Posgrado en Ciencias y Tecnologías de la Información

6. Datos del trabajo escrito
   Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles
   112 páginas

Índice general

Índice de figuras

Resumen

Este trabajo de tesis propone un modelo para definir elementos generales que permitan establecer directrices e incorporar explícitamente la arquitectura de software en metodologías ágiles sin contraponerse a los valores y principios del *Manifiesto Ágil*. Para el desarrollo del modelo se utiliza el Patrón de Procesos de MoProSoft, pues permite definir los elementos formales que lo integran. Además, se incluyen tareas propias adaptadas de los Paquetes de Puesta en Operación para el desarrollo de la arquitectura de software de tal manera que éstas no se contrapongan con el *Manifiesto Ágil*. Así mismo, se establece una generalización del ciclo de vida de las metodologías, considerando actividades y tres etapas (Inicio, Iteraciones y Transición) en base a las metodologías *Scrum*, XP, ASD, AUP y Kanban. Finalmente se propone una guía que permite instanciar el modelo hacia la metodología ágil XP, replicable para el caso de las otras metodologías.

Palabras clave: Metodologías Ágiles, Ciclo de Vida, Arquitectura de Software, Procesos, Generalización, Guía, Instancia,

Capítulo 1

Introducción

Las metodologías ágiles son procesos para desarrollar software de manera rápida. En febrero del año 2001, sucedió una reunión en Utah-EEUU, dónde surgió el término "ágil". Un proceso "ágil" es ligero y suficiente, en consecuencia, el término "ágil" implica ser eficaz y fácil de manejar[18].

Los desarrolladores ven con buenos ojos a las metodologías de desarrollo ágil, porque constituyen soluciones a los proyectos donde los requerimientos del sistema cambian constantemente.

En un desarrollo ágil, los clientes y desarrolladores deben de interactuar constantemente. Los proyectos son construidos con las mejores herramientas y el progreso es medido por la cantidad de código escrito por los desarrolladores para contribuir a la entrega en tiempo de una versión del sistema[47].

La ingeniería de software juega un papel muy importante en el desarrollo y en lograr elementos de calidad como la portabilidad, mantenibilidad, funcionalidad, fiabilidad y extensibilidad del software, que se expresa en la arquitectura de software[30]. La arquitectura de software es una de las partes importantes en la ingeniería de software y se refiere a la estructura o estructuras que componen un sistema, la cual comprende elementos de software, las propiedades externamente visibles de aquellos elementos y las relaciones entre ellas[13].

La arquitectura de software se perfila en etapas tempranas del desarrollo y tiene un papel fundamental para lograr la satisfacción de los atributos de calidad del sistema y para guiar su desarrollo. En los últimos años, la arquitectura de software ha cobrado una mayor importancia dentro de la investigación en la ingeniería de software, por lo que se han establecido métodos para desarrollar arquitecturas robustas y de calidad. Sin embargo, el desarrollo de la arquitectura de software es una práctica poco común dentro de la industria, en especial si se usan metodologías de desarrollo ágiles[3]. Se ha afirmado que en las metodologías ágiles el diseño de la arquitectura de software no es una actividad importante[10].

En la literatura se han documentado varias metodologías ágiles para el desarrollo de software: Scrum[48], Programación Extrema (*eXtreme Programming*, XP)[15], Cristal (Crystal)[23], Desarrollo Adaptable de Software (*Adaptive Software Development*, ASD)[33], Proceso Unificado Ágil (*Agile Unifed Process*, AUP)[9], Kanban[32], entre otras. También se han documentado métodos que se enfocan en la arquitectura de software, propuestos principalmente por el Instituto de Ingeniería de Software (*Software Engineering Institute*, SEI)[36]: Taller de Atributos de Calidad (*Quality Attribute Workshop*, QAW)[12], Diseño Guiado por Atributos (*Attribute-Driven Design*, ADD)[52], Método de Análisis/Evaluación de Arquitectura (*Architecture Tradeoff Analysis Method*, ATAM)[44], Vistas y más allá (*Views and Beyond*,

V&B)[20], entre otros.

En este trabajo se propone un modelo para incorporar métodos de documentación, diseño y evaluación de arquitecturas de software a metodologías ágiles a nivel general, que después se puedan adaptar a algunas metodologías ágiles particulares. Este documento está dividido de la siguiente manera:

- En el capítulo 2 se introduce a las metodologías ágiles. Se explica, qué son las metodologías ágiles, qué es el manifiesto ágil, las diferencias que hay entre las metodologías ágiles y metodologías tradicionales, sus perspectivas, roles y habilidades que deben tener los desarrolladores, qué tanto se puede reutilizar y finalmente cuáles son las metodologías ágiles más usadas y una breve explicación de varias metodologías, así como el ciclo de vida de algunas de ellas. También, se introduce a la arquitectura de software. Se explica, qué es la arquitectura de software, de dónde provienen, cuál es su importancia, algunos métodos que se utilizan para el desarrollo de la arquitectura. Se da una descripción de los métodos (QAW, ADD, ATAM y V&B), así como su posible introducción a las metodologías ágiles. A su vez, se revisan los elementos que componen a algunos procesos y que puedan ser útiles para definir el modelo referido en este trabajo. También, se muestra una recopilación muy breve y resumida sobre trabajos que han realizado diversos autores con planteamientos académicos o empíricos, en los cuales exponen propuestas para llevar la arquitectura de software en algunas metodologías ágiles. Como consecuencia de las discusiones en cada una de las secciones anteriores se presenta el planteamiento del problema. Finalmente, se presentan los objetivos de este trabajo.

- En el capítulo 3 se presenta la metodología y las herramientas que se utilizaron para obtener los mejores resultados de este trabajo.

- En el capítulo 4 se presentan los resultados obtenidos de esta "Idónea Comunicación de Resultados". Los resultados mostrados van enfocados hacia: Generalización del Ciclo de Vida de las Metodologías Ágiles, Paquetes de Puesta en Operación, Modelo Genérico, Guía para realizar la Instancia del Modelo hacia una Metodología Ágil. Finalmente, se presenta la administración del proyecto.

- En el capítulo 5 se discuten los resultados obtenidos.

- En el capítulo 6 se muestran las conclusiones de este trabajo y se presenta el trabajo futuro.

Capítulo 2

Antecedentes

2.1. Metodologías Ágiles

2.1.1. ¿Qué son las metodologías ágiles?

Las metodologías ágiles son procesos para desarrollar software de manera rápida y su introducción se basa en ofrecer una alternativa a los procesos de desarrollo de software "tradicionales"[1], que han sido caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. En febrero de 2001, en una reunión que se llevó a cabo en Utah-EEUU, nace el término "ágil". Las personas que participaron en esa reunión fueron: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas; personas con mucha experiencia y expertos en el desarrollo de software.

El término **ágil implica ser eficaz y fácil de manejar**, en ese sentido, un proceso ágil es ligero y suficiente.

2.1.2. Manifiesto Ágil

Algunas de las personas mencionadas en la sección 2.1.1 ya habían creado sus propios métodos para desarrollar software de manera ágil. Por lo tanto, ellos siempre tuvieron muy en cuenta el proveer respuestas rápidas y adaptables a los cambios que fueran surgiendo en el transcurso del desarrollo del sistema de software; entre otras cosas. Por ese motivo en la reunión mencionada en la sección 2.1.1 los participantes establecieron **valores y principios** que deberían permitir a los equipos desarrollar software de manera rápida. En consecuencia, firman un documento que se le conoce como "Manifiesto Ágil" que se cita a continuación y expone los valores y principios que se deben seguir en una metodología ágil[16].

- **Individuos e interacciones** sobre procesos y herramientas.
- **Software funcionando** sobre documentación extensiva
- **Colaboración con el cliente** sobre negociación contractual
- **Respuesta ante el cambio** sobre seguir un plan

[1] Entendamos como no-ágiles

Los practicantes de las metodologías ágiles no le restan valor a los elementos de la derecha. Sin embargo, valoran más los elementos de la izquierda. Además, en el *Manifiesto Ágil* se establecen 12 principios[17]:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.

8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.

10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

2.1.3. Metodologías Ágiles vs Procesos de Desarrollo de Software

Los procesos de desarrollo de software se han ocupado desde hace ya varías décadas y han dado muy buenos resultados en proyectos grandes cuyo tiempo de entrega sea a largo plazo. Sin embargo, en la mayoría de los proyectos hoy en día se exigen tiempos de desarrollo más cortos a comparación de los tiempos que se manejan en los procesos de desarrollo de software. Por este motivo, ocupar los procesos "tradicionales" pueden no ser los adecuados, en especial para aquellos proyectos en dónde los requerimientos, por parte del cliente, cambian constantemente.

Uno de los problemas que aqueja a la Ingeniería de Software, en especial a los procesos de desarrollo, tiene que ver con el rechazo de los desarrolladores a hacer uso de la Ingeniería de Software, en su totalidad, porque los procesos son muy rigurosos en aspectos cómo roles que se asignan a los integrantes del equipo, actividades y una documentación muy detallada.

Por otro lado, podemos tener varias ventajas si se emplean metodologías ágiles. Por ejemplo, la naturaleza de los métodos ágiles permite el desarrollo iterativo de versiones, estableciendo el beneficio de la entrega de aquella funcionalidad que es crítica para el cliente,

e inmediatamente después de haberla entregado, avanzar con la siguiente o inclusive con la aplicación completa; y todo esto garantizando calidad, pues parte fundamental del desarrollo ágil es la integración de las pruebas de forma continua durante el ciclo de desarrollo, lo que conlleva el asegurar la calidad del trabajo tan pronto como se está desarrollando [31].

A continuación se muestra el Cuadro 2.1 con las principales diferencias entre las metodologías ágiles y las metodologías tradicionales:

Metodologías ágiles	**Metodologías tradicionales**
Se basan en buenas prácticas de programación	Se basan en estándares para el desarrollo de software
Respuesta y adaptación a los cambios a lo largo del proyecto	Resistencia a los cambios a lo largo del proyecto
Menos fases y etapas	Se desarrolla por fases y etapas
Constante interacción con el cliente	Se hacen reuniones con el cliente
Menos asignación de roles	Más asignación de roles
Grupos de trabajo pequeños y en el mismo lugar	Grandes grupos de trabajo y pueden estar distribuidos en diferentes locaciones
Poca documentación	Más documentación
No hay una fase para el diseño de la arquitectura	Es esencial, por lo que hay fases para contemplar el diseño de la arquitectura

Cuadro 2.1: Diferencias entre las metodologías ágiles y las metodologías tradicionales [18]. El texto en color azul indica puntos clave posibles para la introducción una arquitectura de software.

2.1.4. Perspectivas de las Metodologías Ágiles

Las metodologías ágiles son una muy buena opción para desarrollar cierto tipo de proyectos, entre ellas comparten características similares. Las metodologías ágiles toman en cuenta los valores y principios del *Manifiesto Ágil*. En la Figura 2.1 se muestra cómo es la perspectiva de las metodologías ágiles desde la calidad del producto, la productividad, el costo de desarrollo de sistemas y la satisfacción de los involucrados, sobre las metodologías tradicionales. Observamos que la mayoría de los encuestados mencionan que la productividad es mayor, en las ágiles, que en las metodologías tradicionales. De manera similar la calidad y la satisfacción del cliente. Sin embargo, un porcentaje bajo expresa que era inferior. Por otro lado, un porcentaje alto menciona que los costos son más bajos, en las ágiles, que en las metodologías tradicionales. Por lo tanto, como se expresa en la Figura 2.1 y el Cuadro 2.1 podemos concluir que las metodologías ágiles tienen más ventajas sobre las metodologías tradicionales en cierto tipo de proyectos por lo que la perspectiva que proyectan las ágiles es mejor.

2.1.5. Habilidades de los desarrolladores que ocupan metodologías ágiles

En las secciones anteriores hablamos un poco de las metodologías ágiles y los procesos de desarrollo de software, así como sus características y perspectivas que comparten de manera similar entre ellas.

La mayoría de los métodos ágiles parten del supuesto de que se está a cargo de expertos técnicos y con individuos muy motivados. Estos expertos tienden a habilitar a los miembros

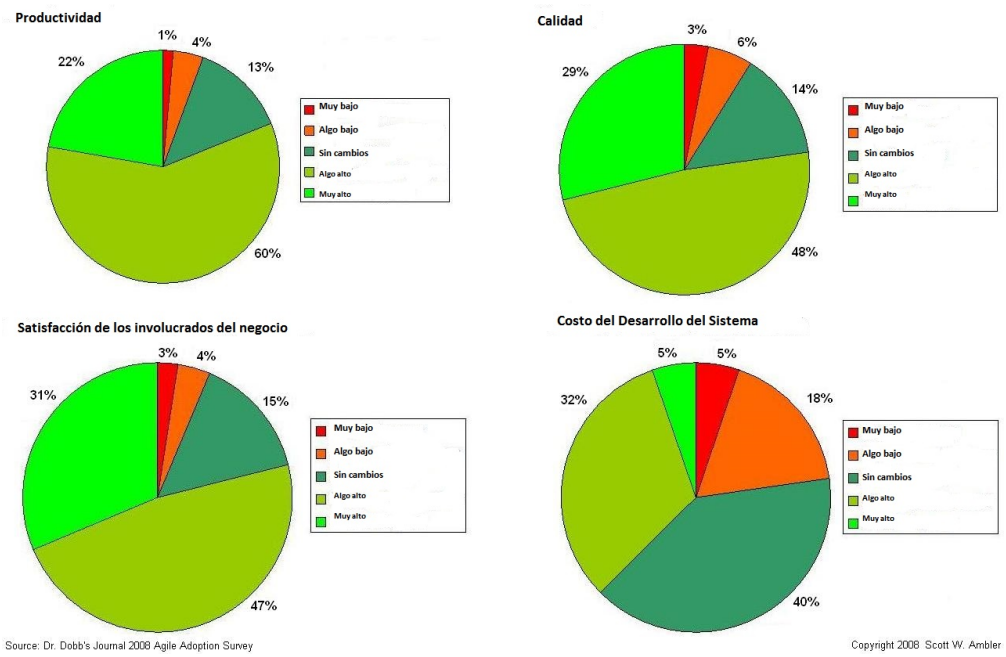

Figura 2.1: Perspectiva de las metodologías ágiles sobre las tradicionales[6].

del equipo del proyecto en los lugares adecuados, tomando en cuenta sus habilidades, ya que estos deben tener conocimiento para poder identificar el camino correcto a seguir y tomen decisiones apropiadas[41]. Los desarrolladores deben tener la experiencia necesaria para definir y adaptar los procesos apropiadamente, es decir, la organización puede formar equipos brillantes y altamente experimentados para resolver problemas de manera efectiva además de la evolución de los procesos mientras se están ejecutando[50].

Los roles de los miembros pueden variar según sea la metodología. Por un lado en XP tenemos roles como: programador, cliente, *tester*[2], *tracker*[3]; entre otros. En *Scrum* también podemos encontrar roles que son asignados, aunque los nombres cambian un poco a comparación de XP, sin embargo, tenemos los siguientes: *Product Owner*[4], *Team Members*[5], etc [41].

2.1.6. Reutilización en las metodologías ágiles

Los mecanismos de reutilización y generalización se espera a que ayude en costos y tiempos para el desarrollo de software. En las metodologías ágiles la reutilización y generalización no se consideran como metas en el desarrollo de aplicaciones[50]. Una desventaja que se tiene en los métodos ágiles es que tiene soporte limitado para la construcción de artefactos reutilizables y no es claro cómo los procesos ágiles puede adaptar la reutilización adecuadamente[50].

[2]Ayuda al cliente a escribir las pruebas funcionales.
[3]Es el encargado de dar seguimiento a las estimaciones y proporciona realimentación al equipo.
[4]Es la persona designada por el cliente para que se encargue del proyecto.
[5]Son los encargados de cubrir todas las necesidades que se presentar para generar el producto final

2.1.7. Metodologías Ágiles más usadas

Desde que surgió el término "ágil" de manera oficial en el 2001 (ver sección 2.1.1), se han venido conociendo metodologías ágiles para desarrollar software, formuladas principalmente por varios de los firmantes del *Manifiesto Ágil* [16]. Entre los más reconocidos por sus propuestas son: *Kent Beck*, *Alistair Cockburn* y *Ken Schwaber*, y varias de estas metodologías ágiles surgieron a finales de los 90's y principios del 2000.

Desde hace unos años las metodologías ágiles han estado teniendo un gran impacto. En una encuesta realizada en el año 2006, de acuerdo a la Figura 2.2, observamos que de la variedad de metodologías ágiles que hubo en la encuesta, DSDM es la que menos ha sido adoptada por los desarrolladores ágiles, sin embargo, XP es la que más han adoptado. Podemos encontrar metodologías ágiles intermedias como FDD, Scrum, AUP y Agile MSF, sin embargo, XP es la metodología ágil que más prefieren los desarrolladores ágiles.

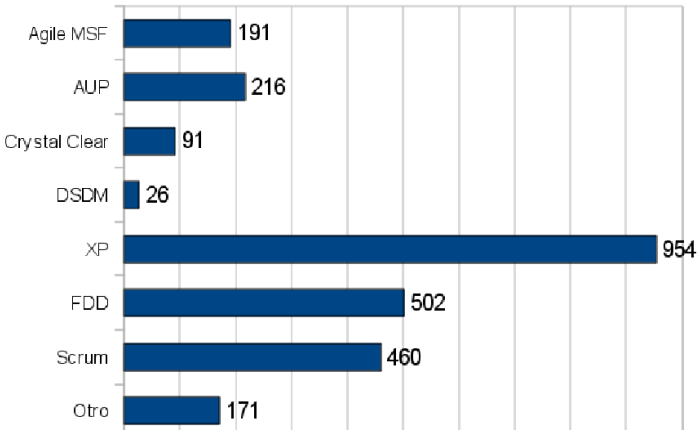

Figura 2.2: Adopción de las metodologías ágiles [5]

En las siguientes secciones se dará una breve explicación de varias de estas metodologías ágiles y el ciclo de vida de algunas de ellas. El ciclo de vida es de gran importancia para entender los detalles de ejecución de cada metodología.

2.1.7.1. Scrum[27]

Scrum es un marco para el trabajo ágil que tiene que ver con el desarrollo del software. El trabajo en *Scrum* también se organiza en ciclos pero estos se les conoce como *Sprints*[6]. Durante cada *Sprint*, el equipo selecciona un conjunto de requerimientos del cliente de una lista priorizada. De esta manera desde el principio del proyecto se van desarrollar características del sistema que tienen un alto valor para el cliente. Al final de cada *Sprint* se entrega un producto de software con las propiedades para ejecutarse en el ambiente requerido por el cliente.

Scrum no es un proceso prescriptivo, es decir, no describe qué hacer en cada circunstancia. *Scrum* es utilizado para trabajo complejo. Acorde a esto, *Scrum* solo ofrece un marco de trabajo y un conjunto de prácticas que mantienen visible y guían los esfuerzos para obtener el resultado más valioso posible.

Scrum permite adaptación, inspección continúa y propicia la innovación. Esto puede producir un producto útil para el cliente, puede desarrollar el espíritu del equipo y satisfacción el

[6]Son iteraciones de trabajo que tienen una duración entre dos a cuatro semanas.

trabajo, generar alta productividad y satisfacción del cliente, logrando las metas financieras propuestas y del mercado.

El ciclo de vida de *Scrum* se puede apreciar en la Figura 2.3 y se describe a continuación.

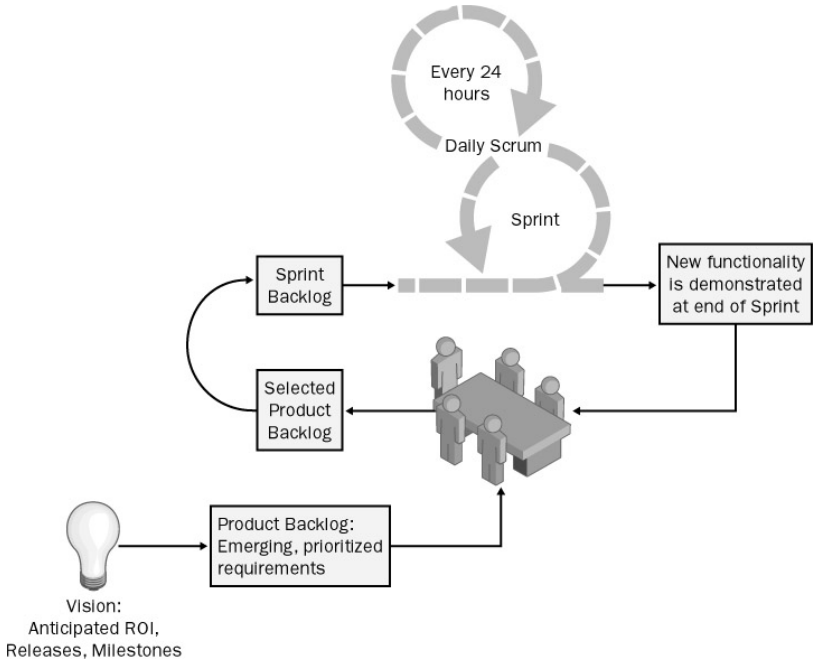

Figura 2.3: Ciclo de vida de *Scrum*

Un proyecto de *Scrum* comienza desarrollando la visión del sistema y puede ser bastante sencilla y general sin mucho detalle. Posiblemente se puede establecer en términos del mercado en lugar de términos del sistema, pero tiene que irse aclarando conforme el proyecto avance. *Product Owner* es el rol responsable de encontrar el financiamiento del proyecto para hacer que se obtenga lo que indica la visión del sistema, de tal manera que se maximice el retorno de inversión (*Return of Inversion*, ROI). *Product Owner* debe formular un plan, así que incorpora un *Product Backlog*. El *Product Backlog* es una lista de requerimientos funcionales y no funcionales que, cuando se conviertan en funcionalidad, cumplirán dicha visión. Se prioriza el *Product Backlog* para que los ítems que probablemente generen más valor sean colocados hasta arriba, así tendrán mayor prioridad. Además se agrupan de acuerdo a las diferentes versiones incrementales que pueda tener. El *Product Backlog* priorizado es un comienzo y su contenido, prioridades y agrupamiento en versiones, usualmente, cambian durante el proyecto. Los cambios en el *Product Backlog* reflejan los requerimientos cambiantes del negocio y qué tan rápido o tan lento el rol de *Team* puede transformarlos en funcionalidad.

Todo el trabajo se realiza en *Sprints*. Cada *Sprint* es una iteración de 30 días del calendario consecutivos. Cada *Sprint* se inicia con la reunión de planificación del *Sprint*: *Sprint Planning Meeting*. Ahí *Product Owner* y *Team* colaboran juntos para saber lo que se va a hacer para el próximo *Sprint*. *Product Owner* selecciona los requerimientos de más alta prioridad y le dice a *Team* qué es lo deseado, éste le dice al *Product Owner* qué tanto puede convertir en funcionalidad para el próximo *Sprint*. La *Sprint Planning Meeting* no puede durar más de 8 horas.

La *Sprint Planning Meeting* tiene dos partes. En las primeras cuatro horas *Product Owner* presenta la parte del *Product Backlog* de más alta prioridad a *Team*. Éste lo cuestiona acer-

ca del contenido, propósito, significado e intenciones del *Product Backlog*. También elabora, actualiza o revisa los estimados para los requerimientos y confirma su exactitud tanto como sea posible. Cuando *Team* conoce lo suficiente y antes de que terminen las primeras 4 horas, selecciona qué tanto del *Product Backlog* cree que puede convertir en *Increment of Potentially Shippable Product Functionality*, al final del *Sprint*. *Team* se compromete con *Product Owner* a que realizará su mejor esfuerzo. Durante la segunda parte que también dura 4 horas de la reunión *Team* planifica el *Sprint*. Debido a que *Team* es responsable de administrar su propio trabajo necesita un plan tentativo para comenzar el *Sprint*. Las tareas que componen este plan se plasman en el *Sprint Backlog*; conforme se desenvuelve el *Sprint* pueden surgir más tareas que deben registrarse en el *Sprint Backlog*. Cada día el equipo se reúne durante 15 minutos, a esta reunión se le llama *Daily Scrum Meeting*. En ésta cada miembro de *Team* responde tres preguntas: ¿Qué has hecho desde la última *Daily Scrum Meeting* con respecto a este proyecto? ¿Qué planeas hacer con respecto a este proyecto, entre este momento y la próxima *Daily Scrum Meeting*? ¿Qué impedimentos te estorban para cumplir tus compromisos con respecto a este *Sprint* y este proyecto?. El propósito de la reunión es sincronizar el trabajo diariamente, de todos los miembros de *Team* y agendar cualquier encuentro que se requiera para continuar trabajando. También se busca tener una visión global del proyecto, descubrir cualquier dependencia, atender las necesidades personales y ajustar el plan de trabajo, de acuerdo a las necesidades del día.

Al final de cada *Sprint*, se lleva a cabo la reunión de revisión del *Sprint*: *Sprint Review Meeting*. En el primer segmento de máximo 4 horas *Team* presenta a *Product Owner* y a cualquier involucrado, lo que se desarrolló durante el *Sprint*. Es una reunión informal en la que se presenta la funcionalidad y se pretende que la gente aporte y ayude a determinar lo que *Team* realizará para el próximo *Sprint*. En el segundo segmento, *ScrumMaster* y *Team* sostienen una reunión de retrospectiva. Esta parte de la reunión, que debe durar 3 horas como máximo *ScrumMaster* exhorta a *Team* para revisar el proceso de desarrollo, dentro de la estructura del proceso y sus prácticas de *Scrum*, para hacerlo más efectivo y agradable.

Sprint Planning Meeting, *Daily Scrum Meeting*, *Sprint Review Meeting* son las prácticas de *Scrum* que constituyen la inspección empírica y adaptación.

Los valores que promueve la metodología ágil *Scrum* van en el sentido de generar un compromiso entre *Team*, *Product Owner* y *ScrumMaster*. Otro valor se centra en el enfoque de *Team* y *ScrumMaster* en el sentido de que deben realizar el trabajo sin distraerse removiendo los obstáculos y evitar interrupciones en el trabajo de *Team*. Un valor se enfoca en la apertura, es decir, se debe hacer visible el trabajo y las prioridades. Finalmente, un valor importante es el respeto que debe existir entre los integrantes individuales de *Team*.

Los valores de *Scrum* no son explícitamente iguales a los valores del *Manifiesto Ágil*. Sin embargo, la forma de trabajo de *Scrum*, como se explica en esta sección, se ve reflejada en los valores y principios del *Manifiesto Ágil*. En ese sentido, el desarrollo de software utilizando en *Scrum* también es ágil.

2.1.7.2. Programación Extrema (eXtreme Programming, XP)

XP es una metodología ágil que se puede aplicar a proyectos para el desarrollo de software. Se centra en "buenas prácticas"[7] de programación, promoviendo el trabajo en equipo y un buen ambiente de trabajo. Se basa en retroalimentación entre el cliente y el equipo de desarrollo, en una buena comunicación "frente-a-frente", el coraje para enfrentar los cambios

[7]Significa que se está de acuerdo, en general, en que la aplicación de estas habilidades, herramientas y técnicas puede aumentar las posibilidades de éxito de una amplia variedad de proyectos.

que puedan haber sobre los requerimientos y la simplicidad en las soluciones siempre que sea posible[18]. En XP los requerimientos, parte fundamental de todo programa a desarrollar, son tomados como escenarios[8] y se les conoce como *historias del usuario* que posteriormente son divididas en una serie de tareas. En XP los programadores trabajan en parejas y desarrollan las pruebas para cada tarea. Una vez desarrolladas las pruebas, el código que se genera, debe ejecutarse satisfactoriamente y si pasan las pruebas el código se integran al sistema.

Las metodologías ágiles tienden a ser procesos iterativos, pero aún así hay metodologías ágiles que siguen fases y etapas para el desarrollo de software. XP no es la excepción y tiene el ciclo de vida que se muestra en la Figura 2.4.

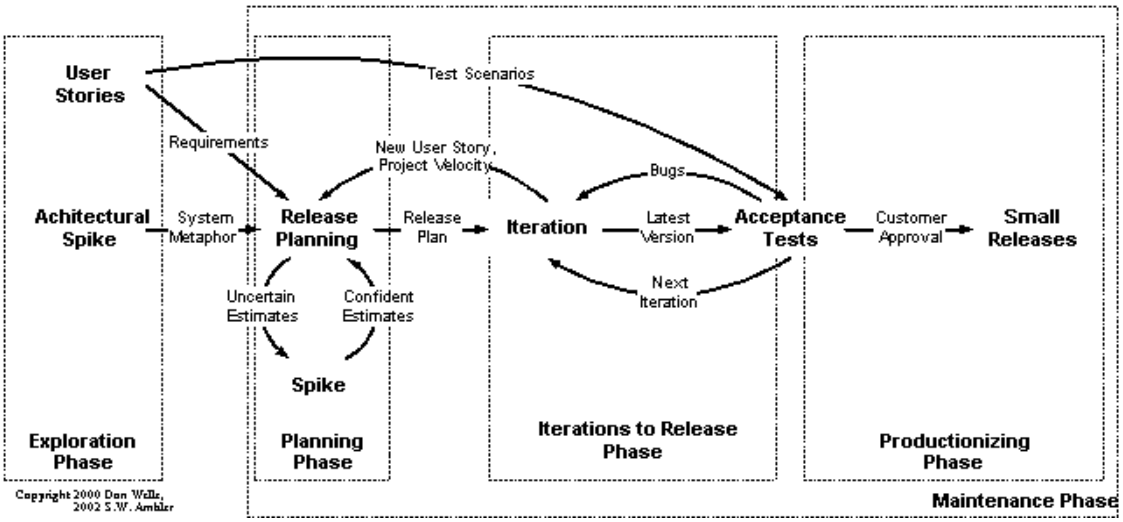

Figura 2.4: Ciclo de vida XP[4].

Se tienen las siguientes fases:

- **Exploración (Exploration):** En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

- **Planificación de la Entrega (Planning):** En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el *punto*. Un *punto*, equivale a una *semana ideal* de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la *velocidad* de desarrollo, establecida en

[8] Un escenario es una descripción parcial y concreta del comportamiento de un sistema en una determinada situación

puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración. La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

- **Iteraciones para la entrega (Iterations to Release):** Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

- **Producción (Productionizing):** La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

- **Mantenimiento (Maintenance):** Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

- **Muerte del Proyecto (Death of Project):** La muerte del proyecto ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo[43].

Como mencionamos anteriormente los valores de XP se enfocan en la comunicación, retroalimentación, coraje y simplicidad. Podemos observar que estos valores se ven reflejados en el *Manifiesto Ágil*. Por mencionar un valor del Manifiesto, tenemos que se debe valorar más a los individuos e interacciones sobre los procesos y herramientas. XP tiene como valor la comunicación lo que facilita, en gran medida, intercambiar la información de manera rápida y continua. Como consecuencia se va promover valorar más a los individuos e interacciones ayudando a realizar el desarrollo del proyecto los más ágil.

2.1.7.3. Desarrollo Adaptable de Software (Adaptive Software Development, ASD)[41]

ASD es un proceso de desarrollo de software que surgió del desarrollo rápido de aplicaciones trabajado por Jim Highsmith y Sam Bayer. ASD se guía en la creencia de la adaptación continua y se orienta en aceptar cambios continuamente.

ASD sustituye al tradicional del ciclo de cascada con una serie de repeticiones: especular, colaborar y ciclos de aprendizaje. De esta forma proporciona aprendizaje continuo y adaptación a estados que emergen en el proyecto.

El ciclo de vida básico de ASD se muestra en la Figura 2.5.

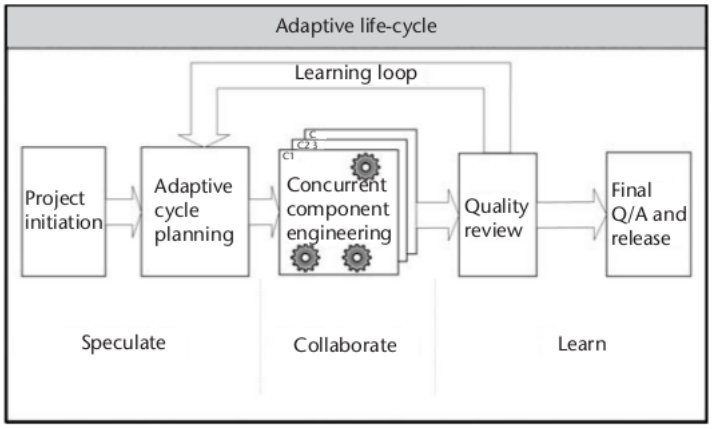

Figura 2.5: Ciclo de vida ASD.

- **Especular (Speculate):** Iniciación y Planificación
  - El primer paso del proyecto en la especulación es un taller de iniciación, dura pocos días a varias semanas, dependiendo del tamaño del proyecto y su alcance. Durante la *iniciación*, el equipo del proyecto y el patrocinador o el cliente determinan los parámetros guía del proyecto, incluyendo: la misión, objetivos y restricciones, organización para el proyecto, los requerimientos del sistema, las estimaciones iniciales del tamaño del producto y el alcance y los riesgos clave para el proyecto.
  - *La planeación* incluye los siguientes pasos:
    - Determinar el periodo de tiempo del proyecto.
    - Determinar el número óptimo de ciclos y de tiempo del proyecto de cada ciclo.
    - Escribir una declaración de objetivos para cada ciclo.
    - Asignar componentes primarios a ciclos.
    - Asignar la tecnología y componentes de soporte a los ciclos.
    - Elaborar una lista de tareas del proyecto.
- **Colaborar (Collaborate):** Diseño de características concurrentes
  - El contenido de esta fase de cada ciclo es planeado en la fase de planificación y se realiza en el periodo de tiempo del proyecto.

- Como podemos observar en la Figura 2.5, esta fase tiene varios "cuadros", eso significa que los miembros del equipo o subequipos generalmente están trabajando de manera concurrente e integrando sus productos de trabajo.

- **Ciclo de aprendizaje (Learning loop):**
  - La práctica de iniciación sienta las bases para el proyecto de "Aprendizaje de bucle". Este bucle ve el proyecto de manera cíclica a la especulación, colaboración, el aprendizaje como el producto emerge gradualmente de los ciclos de adaptación.
- **Aprender (Learn):** Revisión de la calidad
  - Revisiones de grupos de clientes.
  - Inspecciones de software.
  - Post mórtem.
- **Aprender (Learn):** Final y liberación
  - Esta fase es el final para el cliente. Se enfoca en poner toda la información acerca del producto en las manos de los clientes.

ASD se enfoca en la adaptación continua y aceptación de los cambios que surjan a lo largo del proyecto. Para esto se debe tener una excelente comunicación con los integrantes del equipo lo que permite agilidad al momento de realizar cambios en el proyecto. Por otro lado, la forma de trabajar de ASD se puede ver reflejada en el *Manifiesto Ágil* en el sentido de que los cambios son bienvenidos a lo largo del desarrollo lo que permite que ASD sea flexible y abierto.

2.1.7.4. Proceso Unificado Ágil (Agile Unifed Process, AUP)

La metodología AUP es una versión reducida del Proceso Unificado de Rational (RUP)[42]. Es decir, AUP retoma elementos y conceptos que se utilizan en RUP pero en vez de realizar todas las actividades que puedan llevarse a cabo se simplifican y se aplican técnicas ágiles para proporcionar una forma sencilla y fácil de desarrollar software de manera ágil[9].

La Figura 2.6 muestra el ciclo de vida de AUP. En esta Figura podemos observar que hay cuatro fases las cuales se describen en el Cuadro 2.2. Por otro lado, se muestran siete disciplinas que podemos observar en el Cuadro 2.3. Cada fase de AUP tiene un número variable de iteraciones dependiendo del proyecto, y mientras estamos iterando en la fase correspondiente, de manera vertical se va trabajando sobre las disciplinas. Es decir, en la Figura observamos que en la fase Incepción (*Inception*) las disciplinas en las que se trabaja la mayor parte del tiempo son: Administración de la Configuración (*Configuration Management*), Administración del Proyecto (*Project Management*) y Ambiente (*Environment*). En esa fase aún no se tiene nada de código y por ese motivo el trabajo en las disciplinas Pruebas (Test) y Puesta en Operación (Deployment) es prácticamente nulo. Entonces a medida en que se van realizando las iteraciones de cada fase en forma horizontal las disciplinas se van trabajando de manera vertical.

En la Figura 2.7 se muestran las iteraciones principales de AUP las cuales son:

1. Versión de Desarrollo (*Development Release*): esta iteración se enfoca en la puesta en operación del producto terminado en un ambiente de garantía/calidad o un prototipo y se representa con los rombos de color amarillo.

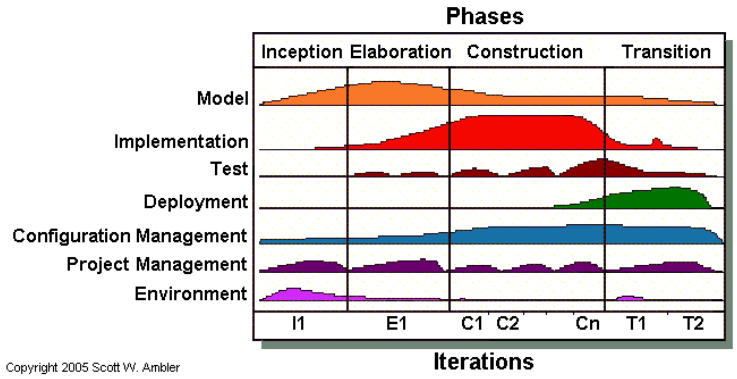

Figura 2.6: Ciclo de vida de AUP[9].

Fases	
Incepción (*Inception*)	Tiene como meta identificar el alcance inicial del proyecto, una arquitectura potencial para el sistema, obtener el financiamiento inicial del proyecto y la aceptación de los involucrados.
Elaboración (*Elaboration*)	La meta es probar la arquitectura del sistema.
Construcción (*Construction*)	La meta es construir software de manera incremental que responda a las necesidades de mayor prioridad de los involucrados.
Transición (*Transition*)	La meta es la de validar y realizar la puesta en operación del sistema en su entorno de producción.

Cuadro 2.2: Fases de AUP.

2. Versión de Producción (*Production Release*): esta iteración se enfoca en la puesta en operación del producto en el área de producción y se representa con los rombos grandes de color verde.

La Figura 2.7 muestra que primero se empieza con la iteración de versión de desarrollo. Es decir, en cada iteración de la versión de desarrollo se van realizando los requerimientos más significativos del sistema. Una vez que se termina la iteración de desarrollo el producto se prueba y se evalúa la calidad para así continuar con la siguiente versión de desarrollo. También, puede ser que en la primera iteración de versión de desarrollo se realice un diseño rápido de aspectos importantes del sistema para que se pueda mostrar al cliente y así éste pueda evaluar y dar retroalimentación que ayude a refinar el requerimiento.

En la Figura 2.7 la primera versión de producción (rombo color verde, V1) es la iteración en la que se realiza la puesta en operación del producto en el área de producción. Al ser la primera versión en producción se debe tomar en cuenta los errores y problemas que puedan surgir, de tal manera que estos errores no se repitan para las siguientes puestas en operación. Como consecuencia esta primera versión de producción llega a tomar más tiempo en liberar que las versiones subsecuentes como: V2, V3, V4, etc. V1 puede tardar hasta doce meses para entregar, de ahí la segunda versión de producción puede llegar a tardar nueve meses y las siguientes versiones seis meses.

Disciplinas	
Modelo (*Model*)	La meta de ésta disciplina se enfoca en entender el negocio de la organización, el dominio del problema que aborda el proyecto e identificar una solución viable para hacer frente al dominio del problema.
Implementación (*Implementation*)	La meta de ésta disciplina es transformar el modelo o modelos en código ejecutable y realizar una prueba a nivel básico, es decir, una prueba unitaria.
Pruebas (*Test*)	La meta de esta disciplina es el desempeño de una evaluación de los objetivos para asegurar la calidad. Esto incluye encontrar defectos, validar que el sistema funcione como fue diseñado y verificar que los requerimientos se cumplan.
Puesta en Operación (*Deployment*)	La meta de ésta disciplina es planificar la entrega del sistema y ejecutar el plan para que el sistema esté disponible para los usuarios finales.
Administración de la Configuración (*Configuration Management*)	La meta de ésta disciplina es administrar el acceso a los artefactos del proyecto. Esto incluye no sólo el seguimiento de versiones de los artefactos en el tiempo, sino que también el control y la administración de los cambios en los mismos.
Administración del Proyecto (*Project Management*)	La meta de ésta disciplina es dirigir las actividades que se llevan a cabo en el proyecto. Esto incluye administración del riesgo, la dirección de personas (asignar tareas, seguimiento de los procesos, etc.), y la coordinación de las personas y los sistemas fuera del alcance del proyecto para asegurarse de que se entregue a tiempo y dentro del presupuesto.
Ambiente (*Environment*)	La meta de ésta disciplina es apoyar el resto de los esfuerzos por asegurar que, el proceso adecuado, la orientación (normas y directrices) y herramientas (hardware, software, etc) estén disponibles para el equipo según sea necesario.

Cuadro 2.3: Disciplinas de AUP.

AUP se basa en los siguientes principios:

- El personal sabe lo que está haciendo
- Simplicidad
- Agilidad
- Se enfoca en actividades de alto valor
- Herramienta de la independencia
- Adaptación de este producto para satisfacer las propias necesidades

Figura 2.7: Iteraciones de AUP[9].

RUP sigue un ciclo de vida iterativo e incremental pero los tiempos de entrega son largos en los ambientes de producción. AUP también tiene la misma característica debido a que la primera versión de producción puede llegar a tomar hasta doce meses, sin embargo, este tiempo va disminuyendo consecuentemente en las demás versiones de producción. En este sentido AUP es más ágil que RUP. Otro aspecto ágil en AUP con respecto a RUP es la retroalimentación que se hace en cada versión de desarrollo, de tal forma que permite adaptabilidad para afrontar posibles cambios en los requerimientos.

Los principios expresados AUP hablan sobre agilidad, simplicidad, actividades de valor, adaptación; entre otros. El *Manifiesto Ágil* comparte estos principios, por lo tanto, la metodología AUP se alinea al Manifiesto.

2.1.7.5. Kanban

En los años 50's surge lo que es *Kanban* (proviene del japonés) y puede traducirse como "letrero" o"tarjeta visual". El primer uso que se da a *Kanban* está relacionado con el Sistema de Producción de Toyota (Toyota System Production). *Kanban* era ocupado por Toyota para manejar las señales en la línea de ensamblaje. Si bien nosotros podemos ocuparlo también, pero para la producción de software.

Entre varias definiciones que podemos encontrar sobre *Kanban*[32] con respecto al desarrollo de software, varios autores comentan lo siguiente:

- Karl Scotland: el significado de *Kanban* es "tarjeta visual", el término es usado por la comunidad de desarrollo de software, y representa mucho más que un estándar de tablero-tareas.
- Torbjörn Gyllebring: Nos ayuda a ganar como equipo.
- Eric Willeke: *Kanban* ayuda a nuestro equipo a la contribución de las actividades mediante el flujo y reduciendo el tiempo a través del Trabajo en Proceso (*Work in Process*, WIP).
- Troy Tuttle: "Pull" valor, WIP limitado y visibilidad pueden crear un ecosistema dónde los equipos tienen oportunidad para improvisar.
- David Anderson: Los principios fundamentales son que un límite fijo WIP proporciona un tiempo de ciclo predecible y una expectativa de un nivel de calidad.

Si bien, cada autor tiene su versión para la definición de *Kanban*, manejaremos la siguiente definición: "*Kanban es un enfoque ajustado para el desarrollo de software*"[1].

La esencia del núcleo de *Kanban* significa:

1. Visualizar el flujo de trabajo:

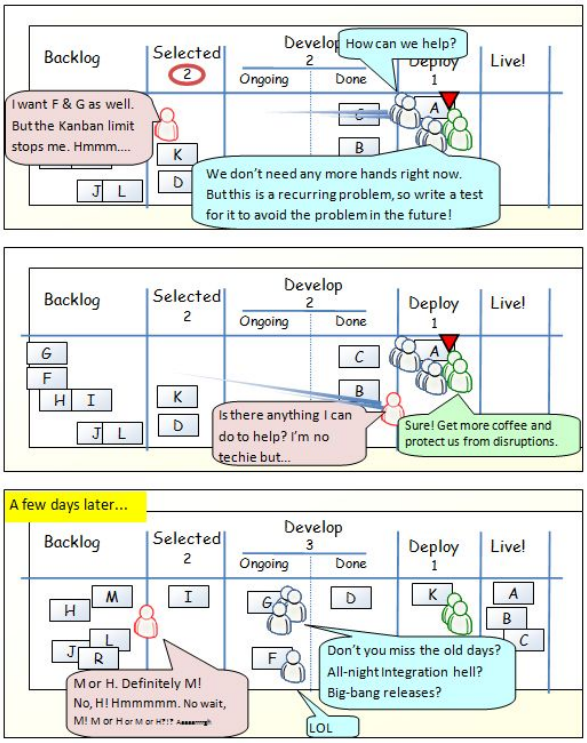

Figura 2.8: Ciclo de vida de *Kanban*[2].

- Dividir el trabajo en piezas o estados, escribir cada ítem en una tarjeta y colocarla en renglones en el muro que indican los flujos de trabajo.
- Usar nombres en las columnas para ilustrar dónde está cada ítem en el flujo de trabajo.

2. WIP limitado: asignar límites explícitos a la forma en que muchos ítems podrían estar en marcha en cada estado de flujo de trabajo.

3. Medir el tiempo de espera: optimizar el proceso para que el tiempo de entrega sea lo más pequeño posible y predecible[1].

En la Figura 2.8 se muestra el ciclo de vida de *Kanban* y se describe a continuación.

Se comienza primero con el *Backlog*, es decir, se genera una lista con los requerimientos funcionales y no funcionales del sistema. También, se establece cuántos requerimientos se pueden elegir para trabajar. Después de que se priorizan y eligen los requerimientos el equipo empieza a trabajar en ellos para realizar la puesta en operación. Si todo sale bien, el flujo de trabajo continua sin problemas, pero siempre teniendo en cuenta cuántos requerimientos se pueden estar implementando.

Kanban se enfoca a ser lo más abierto posible. A esto nos referimos que no es tan prescriptivo como podrían ser otras metodologías. En ese sentido *Kanban* está enfocado a ser más adaptativo. Si bien no hay valores y principios explícitos en *Kanban* el hecho de ser adaptativo agrega agilidad en los desarrollos debido a que permite realizar el desarrollo siguiendo el flujo de trabajo pero requiriendo lo que se vaya a necesitar sin ser tan formales. Por ejemplo, si se desea obtener requerimientos, la documentación que se genere no debe ser tan detallada como la especificación de requerimientos en RUP. El *Manifiesto Ágil* expresa que se debe valorar

individuos e interacciones y *Kanban* no es la excepción ya que tanto cliente como desarrolladores deben estar interactuando constantemente. Por otro lado, la forma de trabajar en *Kanban* se enfoca más hacia el software funcionando sobre la documentación. Por lo tanto *Kanban* se alinea al *Manifiesto Ágil*.

2.1.7.6. Cristal

Cristal es el nombre de una familia de metodologías. En cada metodología Cristal tiene diferentes colores y cierta dureza, que corresponden al tamaño del proyecto y su condición crítica. Algunos de los colores que se emplean en Cristal son: *Claro, Amarillo, Naranja, Naranja Web, Rojo, Marrón, Magenta* y *Azúl*. La comunicación en Cristal se centra en la gente y no en los procesos y artefactos, además, se ajusta para adaptarse a su entorno particular.

El núcleo de la filosofía Cristal es el desarrollo de software visto de una manera útil como un juego cooperativo de invención y comunicación.

Las dos reglas comunes en la familia Cristal son:

- El proyecto debe desarrollarse de manera incremental, con incrementos de no más de cuatro meses.
- El equipo debe mantener un pre- y pos- incremento en el taller de reflexión.

En la Figura 2.9 se muestra el marco de trabajo de las metodologías Cristal. El autor se enfoca en tres factores: carga de comunicaciones (el número de personas, eje "X"), la condición crítica del desarrollo del proyecto (criticidad, eje "Y"), y las prioridades del proyecto[24].

En la Figura 2.9 se muestran diferentes metodologías Cristal. Observamos que en la metodología Cristal Claro el número de personas involucradas está entre 1-6. Sin embargo, la siguiente metodología, Cristal Amarillo, el número de personas involucradas ha aumentado entre 7-20. Las siguientes metodologías también irán aumentando en el personal. Por lo tanto, la comunicación cara a cara empieza a ser distante entre el color de una metodología y otro. Es decir, juntar un equipo de cuatro personas en un mismo cuarto no tiene consecuencias graves en la comunicación porque la comunicación es más fluida y de manera directa. Sin embargo, si tenemos un equipo de ochenta personas en un mismo cuarto se requiere de más comunicación, practicas y herramientas de interacción. Cristal revisa primero los aspectos que se centran en las personas y su interacción, y después aquellos aspectos que se enfocan en el proceso, producción de trabajo, ceremonias y otros. Por otro lado, en la misma Figura 2.9 tenemos que en el eje "Y" se encuentra la condición crítica del desarrollo del proyecto: pérdida de bienestar y comodidad, pérdida "discrecional" de dinero, pérdida "esencial" de dinero (por ejemplo, la quiebra) o la muerte. [24].

Se debe aclarar que de todos los colores de la familia de metodologías Cristal la que se utiliza para los desarrollos ágiles es la de color Claro. Esto se debe a que que está diseñada para pequeños proyectos, con un número de desarrolladores bajo (seis personas y con modificaciones podría ser de ocho a diez) y el área de trabajo debe ser en un mismo espacio.

Cristal Claro se alinea al *Manifiesto Ágil* teniendo entregas frecuentes, comunicación, colaboración con el cliente y respuesta ante los cambios. Las demás metodologías Cristal, a medida que cambia el color, empiezan a mostrar cierta dureza. Es decir, se establecen más actividades, prácticas, métodos, formas de comunicación, herramientas, etc. Esto empieza a generar más carga en la metodología de manera gradual.

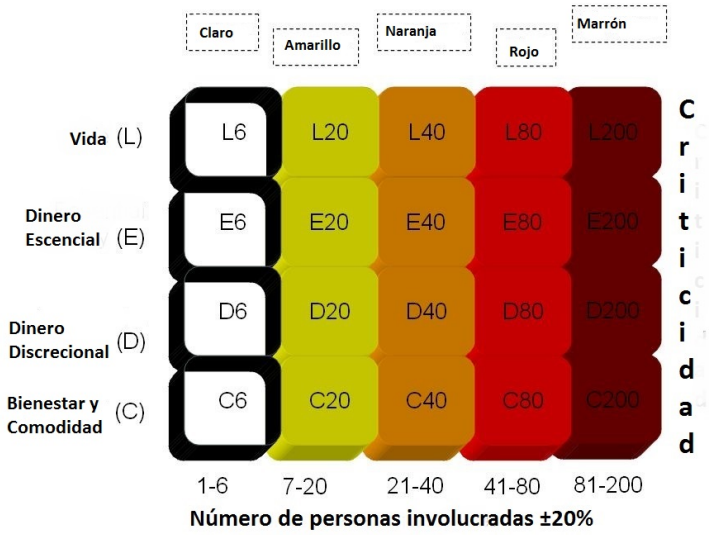

Figura 2.9: Familia de metodologías Cristal por nombres y colores.

2.1.7.7. Método de Desarrollo de Sistemas Dinámicos (Dynamic Systems Development Method, DSDM)[51]

DSDM define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD (Rapid Application Development) unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos.

DSDM se basa en nueve principios[34] que concuerdan con los principios del *Manifiesto Ágil*:

1. La participación activa de los usuarios es imprescindible
2. Los equipos en DSDM deben tener la facultad de tomar decisiones
3. Se enfoca en la frecuente entrega de productos
4. La aptitud para el propósito de negocios es el criterio fundamental para la aceptación de los entregables
5. El desarrollo iterativo e incremental es necesario para converger en una solución de negocio
6. Todos los cambios durante el desarrollo son reversibles
7. Los requisitos son la línea base en un alto nivel
8. Las pruebas se integran a lo largo del ciclo de vida
9. El enfoque colaborativo y cooperativo entre los involucrados es esencial

Observando los principios anteriores notamos que son muy cercanos a los del *Manifiesto Ágil*. En ese sentido esta metodología se considera ágil por la cercanía que hay en la filosofía de sus principios con los del manifiesto.

DSDM consiste de tres fases:

1. Pre-proyecto. Se identifican los candidatos a proyectos, se realiza el financiamiento del proyecto y se asegura el compromiso por parte de los equipos, usuarios y clientes. Básicamente, el objetivo de esta fase es evitar los malos entendidos que puedan generar problemas en las siguientes fases.

2. Ciclo de vida del proyecto. Consiste en cinco etapas:

   - Estudio de viabilidad: se examinan los requerimientos previos y se realizan las preguntas siguientes, ¿este proyecto satisface la demanda del negocio?, ¿este proyecto puede ajustarse a DSDM? y ¿cuáles son los riesgos más importantes que intervienen?
   - Estudio de negocio: se identifica que el proyecto sea viable y se utilizan técnicas que faciliten y aseguren un proyecto de calidad.
   - Iteración del modelo funcional: se realiza un modelo y un prototipo funcional para representar las funcionalidades que se definieron en la iteración que se está desarrollando.
   - Diseño y construcción de la iteración del sistema: se integra los componentes funcionales de la fase anterior y se entrega un diseño de prototipo que los usuarios ponen a prueba
   - Implementación: se entrega el sistema probado y la documentación.

3. Post-proyecto. Consiste en asegurar que el sistema opere de manera eficiente. Este se realiza a través del mantenimiento y correcciones de acuerdo con los principios de DSDM.

2.1.8. Discusión

Las metodologías ágiles son procesos para desarrollar software de manera rápida. Éstas metodologías comparten características comunes entre ellas y a su vez están alineadas a lo que se expresa en el *Manifiesto Ágil*. Con respecto a los métodos tradicionales para el desarrollo de software nos queda claro que las metodologías ágiles no vienen a reemplazar esos métodos, en cambio están pensadas para poder realizar desarrollos de software de manera ágil y tener más ventaja competitiva al respecto. Para esto toman muy en cuenta sus valores y principios que es dónde se concentra la mayor parte de la filosofía para poder agregar agilidad a la metodología sin descuidar la flexibilidad y adaptabilidad que tanto se menciona.

En el Cuadro 2.4 se muestra, de manera resumida, una comparación de las características de tres metodologías: *Scrum*, XP y Cristal Claro, en el uso de los elementos empleados en el desarrollo de software. Los datos que se utilizaron en el Cuadro 2.4 fueron tomados de las siguientes referencias: [48], [15], [23], [18], [50] y [5]. Los números en el Cuadro 2.4 representan el nivel de importancia que se propuso de acuerdo a la información que se obtuvo de las referencias anteriores, de tal forma que el 0 representa sin importancia, el 1 representa muy baja importancia, el 2 representa baja importancia, el 3 representa media importancia, el 4 representa alta importancia y el 5 representa muy alta importancia. A continuación se describe el Cuadro:

- En la columna "Roles", observamos que XP y Cristal claro se asigna un valor 5 a comparación de *Scrum* que tiene 4. Esto porque los roles que emplea *Scrum* son menos que en XP y Cristal claro.

- En la columna “Habilidades”, todas las metodologías se asigna un valor 5 ya que sus practicantes deben contar con cierta experiencia en el desarrollo de software.
- En la columna “Reutilización”, los valores van desde 0 a 2 porque en las metodologías ágiles, explícitamente, no se menciona la reutilización pero de acuerdo al nivel de habilidades de los desarrolladores es posible que implícitamente se esté aplicando la reutilización.
- En la columna “Fases”, *Scrum* y XP se asigna un valor 5 debido a que, como se observa en la sección 2.1.7, el número de fases es mayor que en Cristal claro.
- En la columna “Actividades”, las metodologías tienen un valor 5 dado que en ellas se realizan actividades que aportan mucho valor al método ágil.
- En las columnas “Impacto en proyectos grandes y medianos”, se asigna un valor de 0 a 2 lo cual significa que las metodologías ágiles son débiles en ese elemento.
- En la columna “Impacto en proyectos pequeños”, se asigna el valor de 5 a las metodologías porque son empleadas para ir entregando avances de manera rápida y constante en este tipo de proyectos.
- En la columna “Documentación”, las metodologías tienen un valor entre 2 y 3 debido a que casi no se documenta porque se valora más el software funcionando sobre una documentación extensiva.
- La columna “Tiempo de respuesta ante cambios”, se asigna un valor entre 4 y 5 y aquí es dónde se puede apreciar la la fortaleza de las metodologías, ya que siempre están dispuestas a recibir nuevos cambios en los requerimientos.
- La columna “Más usadas”, se asigna el valor de 5 en *Scrum* y XP como resultado de la encuesta reportada en la Figura 2.2.
- La columna “Colaboración con el cliente”, se asigna el valor de 5 porque en general se mantiene una constante comunicación e interacción con el cliente.

El *Manifiesto Ágil* establece los valores y principios que deben seguir las metodologías ágiles. Por este motivo, si se desarrolla una nueva metodología no debe contraponerse a lo que establece el Manifiesto. Es decir, esta nueva metodología debe respetar los valores y principios del *Manifiesto Ágil* y es por eso que cualquier introducción de elementos nuevos que puedan ayudar a perfeccionar a las metodologías deben revisarse para ver si los elementos a incorporar no se contraponen, con lo expresado en el manifiesto. En varias metodologías puede que los valores y principios no estén expresados de manera explícita como en el Manifiesto debido a que cada autor de cada metodología puede tener sus propias ideas pero lo importante es que deben coincidir con lo expresado en el Manifiesto. Sin embargo, la forma de trabajar utilizando la metodología puede abrir más el panorama para comprender la agilidad en la metodología.

Un punto importante con respecto a las metodologías ágiles, en general, es el uso de buenas prácticas de programación. Por ejemplo, en la metodología ágil XP existe la práctica: “Programación en Parejas”. Esta práctica puede considerarse de mucha utilidad por las siguientes razones: cuando se programa en parejas, por lo general, el código que se va escribiendo se compila hasta que se concluya la codificación; mientras tanto, la persona que está sentada junto al que escribe el código va revisando que no se tengan errores de: sintaxis, inicialización,

Características / Metodologías	Roles	Habilidades	Reutilización	Fases y ciclos	Actividades	Impacto en proyectos grandes	Impacto en proyectos medianos	Impacto en proyectos pequeños	Documentación	Tiempo de respuesta ante cambios	Más usadas	Colaboración con el cliente
Scrum	4	5	0	5	5	0	2	5	2	5	5	5
XP	5	5	1	5	5	0	2	5	2	5	5	5
Cristal claro	5	5	2	3	5	0	2	5	3	4	3	5

Cuadro 2.4: Características de las metodologías ágiles.

declaración, funciones, etc. De esta manera, se puede ahorrar tiempo codificando, aunque en la actualidad los "Entornos de Desarrollo Integrado" (*Integrated Development Environment*, IDE) ayudan mucho. Sin embargo, cuando se tiene un integrante que no cuenta con la experiencia necesaria, esta práctica, puede enriquecerlo bastante para comprender diversos aspectos del proyecto.

Por otro lado, para que los desarrollos ágiles sean exitosos, las metodologías ágiles parten del supuesto que se cuenta con gente muy experimentada, es decir, se debe tener expertos que puedan resolver desde planteamientos sencillos hasta complejos, teniendo en cuenta que el equipo debe estar siempre motivado y sin descuidar la interacción y comunicación constante con el cliente.

Finalmente, en esta sección se observó que la perspectiva de las metodologías ágiles hacia proyectos con ciertas características como: cambios en los requerimientos constantemente, tiempos de entregas cortos, entre otras; tienen un impacto muy positivo en el desarrollo de software y no deben menos preciar el alcance que pueden tener. Además, todas las metodologías ágiles tienen elementos y características comunes que se alinean al *Manifiesto Ágil* lo que está garantizando: agilidad, comunicación, interacción, adaptabilidad, respuestas inmediatas, trabajo constante y calidad.

2.2. Arquitectura de Software

La arquitectura de software empieza a realizarse desde mucho antes de entrar a una fase especial para su diseño, un ejemplo son los casos de uso. Estos son importantes porque nos muestran un camino a través de los requerimientos, análisis, diseño, implementación y pruebas del sistema. Es decir, capturan la forma en que se comportará el sistema sin especificar cómo se implementará ese comportamiento y validan la arquitectura. Sin embargo, hay más cosas en el proceso de desarrollo de software que tener que ir a ciegas a través de los flujos de trabajo (disciplinas) guiados solo por casos de uso[38]. Para el diseño de la arquitectura de software los casos de uso, a pesar de que son muy importantes, no son suficientes. Se necesita algo "más" y ese algo "más" es la arquitectura. La arquitecta de un sistema es una visión común

entre todos los desarrolladores e involucrados. La arquitectura nos da una perspectiva clara de todo el sistema, lo cual es necesario para controlar su desarrollo[38].

La arquitectura de software es una de las partes más importantes en la ingeniería de software, se puede definir como se menciona a continuación:

> "*La arquitectura de software es la estructura o estructuras del sistema, la cual comprende elementos de software, las propiedades externamente visibles de aquellos elementos y las relaciones entre ellos*"[14].

Los atributos de calidad[9] en el software juegan un papel muy importante y se pueden ver reflejados en una buena arquitectura, como por ejemplo, el que un sistema sea mantenible, seguro, escalable, interoperable, etc.

La arquitectura de software es un tema muy importante en los procesos de desarrollo de software. Sin embargo, en algunas metodologías, como las ágiles, no se contemplan fases para el diseño de la arquitectura de software[3].

2.2.1. ¿De dónde provienen las arquitecturas?

En la sección 2.2 hacemos referencia a una definición de arquitectura de software. Sin embargo, debemos de profundizar más y cuestionarnos, ¿de dónde provienen las arquitecturas?.

A continuación citamos lo siguiente[14]:

> "*Una arquitectura es el resultado de una serie de decisiones de negocio y decisiones técnicas. Hay muchas influencias de trabajo en el diseño y la realización de esas influencias cambiarán dependiendo del ambiente en el que la arquitectura es requerida para el desempeño*".

El párrafo anterior nos da una clara idea de dónde proviene la arquitectura. Implícitamente considera los requerimientos no funcionales del sistema y la influencia que hay de los involucrados sobre el arquitecto.

2.2.2. Importancia de la Arquitectura de Software

Necesitamos una arquitectura que describa los elementos del modelo que son más importantes. ¿Cómo vamos a determinar cuáles elementos son importantes? La importancia se encuentra en el hecho de que la arquitectura es una guía para el desarrollo del sistema[38].

En este contexto hay fundamentalmente tres razones[14] para resaltar la importancia de las arquitecturas de software:

1. **Comunicación entre los involucrados.** La arquitectura de software representa una abstracción común de un sistema, que los involucrados en el desarrollo del sistema pueden usar como base para el entendimiento mutuo, negociación, consenso y comunicación.

2. **Decisiones tempranas de diseño.** La arquitectura de software manifiesta las decisiones tempranas de diseño acerca del desarrollo de un sistema, y esos enlaces tempranos acarrean un peso fuera de proporción con respecto al desarrollo restante, su despliegue y

[9] Un atributo de calidad es una propiedad de un producto o de un bien, por la cual la calidad del mismo será juzgada por los involucrados en el desarrollo del sistema. Atributos de calidad tal como desempeño, seguridad, modificabilidad, confiabilidad y usabilidad, entre otros, tienen una influencia significativa en la arquitectura del sistema de software[49].

su mantenimiento. Eso es también el primer momento en el cual las decisiones de diseño que rigen al sistema a ser construido, pueda ser analizado.

3. **Abstracción transferible de un sistema.** La arquitectura de software constituye un modelo, de cómo un sistema es estructurado y cómo sus elementos trabajan juntos y este modelo es transferible a otros desarrollos. En particular se puede aplicar a otros sistemas que exhiben atributos de calidad similares, requerimientos funcionales y pueden promover la reutilización en sistemas a gran escala.

En [14] se menciona que el propósito para el diseño de la arquitectura del sistema es esbozar el diseño y modelos de despliegue, mediante la identificación de lo siguiente:

- Nodos y su configuración de red.
- Subsistemas y sus interfaces.
- Diseño de clases significativas para la arquitectura.
- Mecanismos genéricos de diseño.

La arquitectura, por lo tanto, es fundamental para entender el sistema, organizar el desarrollo, fomentar la reutilización y evolución del sistema con base a un conjunto de vistas[38]. De ahí la importancia para tener una buena arquitectura de software.

2.2.3. Métodos del Instituto de Ingeniería de Software (SEI)

El Instituto de Ingeniería de Software (*Software Engineering Institute*, SEI) ha desarrollado varios métodos para la creación, evaluación y documentación de la arquitectura de software. La arquitectura debe tener atributos de calidad tomando en cuenta las directrices arquitectónicas[10] por las cuales se obtienen los atributos de calidad. Los métodos básicos del SEI nos ofrecen una buena opción para poder cubrir esos atributos de calidad y estos son: QAW (*Quality Attribute Workshop*)[12], ADD (*Attribute-Driven Design*)[52], ATAM (*Architecture Tradeoff Analysis Method*)[44], CBAM (*Cost Benefits Analysis Method*)[40] y ARID (*Active Reviews for Intermediate Designs*)[22].

Varios de los métodos que el SEI ha promovido van enfocados al diseño y análisis de la arquitectura de software. Esos métodos han sido probados en procesos de desarrollo de software tradicionales[45].

Las propuestas del SEI son planteadas para desarrollar arquitecturas de software, incluyendo documentación, y para realizar mantenimiento. Dependiendo de las necesidades que se tengan en el desarrollo de arquitecturas, se deben escoger los métodos particulares. Para propósitos de este trabajo, los métodos considerados se plantean en las siguientes secciones.

2.2.3.1. Taller de Atributos de Calidad (*Quality Attribute Workshop*, QAW)[12]

QAW es un método facilitador que incorpora a todos los involucrados en el desarrollo de un sistema de software de manera temprana en el ciclo de vida, para descubrir los atributos de calidad en la arquitectura. QAW depende de la participación de los involucrados del sistema, individualmente en los que van a tener un impacto significativo, tales como, usuarios, administradores, arquitectos, etc.

[10] Es la combinación de requerimientos funcionales, de atributos de calidad de negocio que conforma la arquitectura del elemento particular bajo consideración.

QAW define una serie de pasos a seguir:

1. Presentación del QAW e introducciones.
2. Presentación del negocio.
3. Presentación del plan arquitectónico.
4. Identificación de las directrices arquitectónicas.
5. Lluvia de ideas para los escenarios.
6. Consolidación de los escenarios.
7. Priorización de los escenarios.
8. Refinamiento de los escenarios.

2.2.3.2. Diseño Guiado por Atributos (*Attribute-Driven Design*, ADD)[52]

ADD es un método que se enfoca en definir arquitecturas de software teniendo como base el proceso de diseño en los requerimientos de atributos de calidad para la arquitectura. Sigue un proceso de descomposición recursivo donde, en cada etapa en la descomposición, tácticas y patrones arquitectónicos son elegidos para satisfacer un conjunto de escenarios de atributos de calidad[11].

ADD define una serie de pasos a seguir:

1. Confirmar que hay suficiente información sobre los requerimientos.
2. Elegir un elemento del sistema a descomponer.
3. Identificar candidatos a directrices arquitectónicas.
4. Elegir un concepto arquitectónico que satisfaga la directriz arquitectónica[12].
5. Ejemplificar (instanciar) elementos arquitectónicos y asignar responsabilidades.
6. Definir interfaces para los elementos instanciados.
7. Verificar y refinar requerimientos y convertirlos en restricciones para los elementos instanciados.
8. Repetir de los pasos 2 al 7 para el siguiente elemento del sistema que se va a descomponer.

[11]

[12]Es cualquier requerimiento funcional, restricción de diseño, requerimiento de atributo de calidad o requerimiento de negocio que tiene un impacto significativo en la arquitectura de software[14].

2.2.3.3. Método de Análisis/Evaluación de Arquitectura (*Architecture Tradeoff Analysis Method*, ATAM)[44]

ATAM es un método para mitigar los riesgos arquitectónicos. Es usado de manera temprana en el ciclo de vida del desarrollo de software. ATAM fue desarrollado por el SEI.

El propósito de ATAM es evaluar las consecuencias de las decisiones arquitectónicas en puntos vista de requerimientos de atributos de calidad.

Los pasos de ATAM son:

1. Presentar ATAM.
2. Presentar las directrices de negocio.
3. Presentar la Arquitectura.
4. Identificar los enfoques arquitectónicos.
5. Generar un árbol de utilidad de atributos de calidad.
6. Analizar enfoques arquitectónicos.
7. Lluvia de ideas y escenarios priorizados.
8. Analizar enfoques arquitectónicos.
9. Presentar resultados.

2.2.3.4. Vistas y más allá (*Views and Beyond*, V&B)[20]

V&B es una propuesta realizada en el SEI para documentar la arquitectura software de un sistema. V&B propone la definición de una serie de vistas relevantes de la arquitectura de software del sistema, documentando cada una de ellas, así como las características que afecten a más de una o a todas en general.

Las prácticas modernas de arquitectura de software están muy unidas a las vistas arquitectónicas. Una vista es una representación de un conjunto de elementos del sistema y las relaciones asociadas con ellas[19]. Las vistas son representaciones de las estructuras de muchos sistemas presentados de manera simultánea en los sistemas de software. Los sistemas modernos son muy complejos como para ser entendidos una sola vez. En cambio, si se restringe la atención de un componente a otro, las estructuras del sistema son representadas como vistas[19].

Hay una forma inmensa de elegir las vistas que se presentarán. Sin embargo, es útil pensar acerca de vistas en grupos, también en vistas que estén de acuerdo con el tipo de información que expresan o transmiten:

- Vistas de módulos, describen cómo el sistema será estructurado como un conjunto de unidades de código.
- Vistas de componentes y conectores, describen cómo el sistema será estructurado como un conjunto de elementos que van a interactuar en tiempo de ejecución.
- Vistas de asignación, describen cómo el sistema se relaciona con estructuras fuera del sistema en su entorno.

En V&B para elegir las vistas se sigue un procedimiento de tres pasos:

1. Produce una lista de vistas candidatas.- Se construye una tabla de involucrados/vistas para nuestros proyectos. Se enumeran los involucrados para los proyectos y se ponen en las filas. Para las columnas, se enumeran las vistas que se aplicarán al sistema. Algunas vistas se aplican a cada sistema, otras vistas sólo se aplican a los sistemas diseñados de acuerdo a los estilos correspondientes.

   Una vez que se tienen las filas y columnas definidas, se completa cada celda con la información de los involucrados sea requerida para una vista: ninguna, solo resumen o muy detallado.

2. Combinando vistas.- Se busca vistas de la tabla que sólo requieren visión a profundidad o que sirven a muy pocos involucrados.

   Para las vistas, se identifican buenos candidatos que pueden llegar a ser vistos que se puedan combinar. Una vista combinada muestra información nativa de dos o más vistas separadas. Una regla importante es que si hay una fuerte correspondencia entre elementos en dos vistas, son buenos candidatos a ser combinadas.

3. Priorizar.- Después del paso 2, debemos de tener un mínimo conjunto de vistas que sirvan para la comunidad de involucrados. En este punto, necesitamos decidir que vamos a hacer primero. Por ejemplo, se podría ver los intereses de los involucrados para ver qué se necesita primero.

2.2.4. Discusión

La arquitectura de software se define como la estructura o estructuras del sistema que va a comprender los elementos de software, las propiedades visibles externas de aquellos elementos y las relaciones entre esos elementos. Comprender esta definición es importante porque se puede observar que la arquitectura proporciona una visión de alto nivel para el desarrollo de software. De este modo la arquitectura debe ser vista como una estructura general con propiedades visibles, así como las posible relaciones con sus elementos.

La arquitectura de software es útil en el desarrollo y debe tomarse en cuenta desde un inicio. De no hacerlo, se puede tener el riesgo de no tener una idea clara de los elementos que se deben desarrollar así como las interacciones y relaciones que puedan tener estos elementos. Es por esto que la arquitectura puede considerarse como una guía para los desarrolladores[38]. Así mismo el producto se ve beneficiado porque la arquitectura considera varios atributos de calidad necesarios para el producto como pueden ser: seguridad, usabilidad, disponibilidad, portabilidad, etc. Por ejemplo en mantenibilidad, se estaría ganando tiempo en el momento de realizar modificaciones en el código y así no cambiar de manera constante gran parte del código para poder implementar cada nuevo cambio que surja. Por lo que el mantenimiento sería relativamente flexible y no tan complejo de realizarlo. Por lo tanto, tener en cuenta que si se emplea el desarrollo de la arquitectura se podría dar valor al producto en desarrollo. Por otro lado, emplear la arquitectura desde el inicio es útil para tomar decisiones importantes de diseño y no tener problemas a la mitad o al final del desarrollo. Por último, la arquitectura estaría promoviendo la reutilización porque la experiencia ganada en un desarrollo puede ser utilizada en el desarrollo de un nuevo producto que requiera atributos de calidad semejantes.

Para el desarrollo de la arquitectura de software el SEI plantea varios métodos. Los métodos considerados en esta tesis son: QAW, ADD, ATAM y V&B. QAW es un método que se recomienda antes de diseñar la arquitectura, para conocer los requerimientos de atributos de

calidad del sistema en desarrollo. QAW se centra en el sistema, se incluye a todos los involucrados y se basa en escenarios[12]. Además, hace uso de elementos como: árboles de utilidad, lluvia de ideas y plantillas. Como resultado se obtienen escenarios perfectamente priorizados y refinados en dónde los involucrados (entre ellos el cliente) tienen que interactuar para lograr resultados apropiados. ADD es un método recursivo que se utiliza para el diseño de la arquitectura, el cual requiere como entrada los resultados obtenidos con el QAW. De igual manera que QAW, ADD recomienda el uso de árboles de utilidad y se necesita una persona con experiencia y conocimiento para utilizar las tácticas y estrategias que cumplirán los atributos de calidad. ATAM es un método que se usa para evaluar las consecuencias de las decisiones arquitectónicas en puntos de vista de los requerimientos de atributos de calidad. Para utilizar ATAM no es necesario tener la arquitectura en el sistema ya implementado. Es decir, si aún no se ha desarrollado el sistema y se tiene documentada la arquitectura (tal vez no totalmente), se puede aplicar ATAM. Los beneficios que se obtienen al emplear ATAM se ven reflejados en la documentación generada que será de gran utilidad a lo largo del desarrollo. El método sugiere el uso de elementos que también emplea QAW y ADD. Un aspecto que puede resultar negativo si se emplea ATAM se debe a que este método puede no ser tan accesible para las PyMEs debido a que se debe contratar a un grupo externo a la organización, diferente al grupo de desarrollo interno, quienes realizan el análisis y evaluación de la arquitectura. Lo que si se puede rescatar de ATAM son aquellos elementos que pueden apoyar de manera significativa la arquitectura de software. V&B es un método que se utiliza para documentar la arquitectura de software a través de una serie de vistas relevantes de la arquitectura y sus características. Por lo tanto, estos métodos del SEI tienen como objetivos el análisis, diseño, evaluación y documentación de la arquitectura los cuales vamos a analizar para su posible utilización en este trabajo de tesis.

En el Cuadro 2.5 se presenta una comparación de elementos relacionados al desarrollo de arquitectura de software, contra los métodos propuestos del SEI. El objetivo de esta comparación los elementos que podrían ser aplicables en metodologías ágiles. Los elementos considerados fueron extraídos de las referencias [12], [52], [44] y [20]. Así mismo, la información en las celdas se utiliza para indicar si el elemento se puede aplicar directamente, se asigna el color verde (1), o parcialmente, se asigna el color amarillo (2), en las metodologías ágiles. A continuación, se describe a detalle el Cuadro 2.5:

- En la columna "Pasos", observamos que los métodos: QAW, ADD y ATAM siguen una serie de pasos que se pueden aplicar parcialmente en las metodologías ágiles para aspectos arquitectónicos y por eso se asignó el color amarillo (2). Es decir, por lo general los primeros pasos tanto en QAW y ATAM son una introducción del método y/o presentación de los involucrados, y tal vez con una sola vez que se lleven a cabo para conocer más a fondo es más que suficiente, pero para los siguientes desarrollos ya no serían tan necesarios. En el trabajo mencionado en la sección 2.3.3 hacen un mapeo de la metodología ágil Cristal[23] sobre ATAM y muestran que los pasos 1, 2 y 9 pueden eliminarse. Siguiendo esa misma idea que plantean los autores se podría eliminar algunos pasos para QAW y ADD de tal manera que se puedan agilizar para introducirlos en las metodologías ágiles.

- En la columna "Roles", observamos que los métodos QAW, ADD y ATAM, tienen como característica en común actividades que son llevadas a cabo por personas que cubren cierto papel en el equipo de desarrollo, mejor conocido como rol. De igual manera, las metodologías ágiles (ver Cuadro 2.4) hacen uso de roles entre sus integrantes. Por este

motivo, introducir algún rol que tome en cuenta aspectos de diseño de arquitectura de software beneficiaría en gran medida para poder plantear el diseño de la arquitectura de software en los desarrollos ágiles. Sin dejar de considerar que los desarrollos que tengan experiencia pueden estar desarrollando de manera implícita la arquitectura. Por lo tanto, se considera que a los roles se les asigna verde (1).

- La columna "Documentación" describe:
  - Escenarios, plantillas, árboles de utilidad y lluvia de ideas: son elementos importantes en QAW, ADD y ATAM y se aplican parcialmente a los métodos ágiles. Por lo tanto, se asigna color amarillo (2).
  - Diseño: elementos de diseño que se plantean en V&B se pueden aplicar en metodologías ágiles y se asigna el color verde (1), porque el trabajo *Documenting Software Architectures in an Agile World*[21], hace mención de que se puede ocupar V&B con metodologías ágiles.
- Las columnas: "Requerimientos", "Diseño" y "Análisis" se aplican parcialmente en metodologías ágiles y se les asigna el color amarillo (2) porque se pueden introducir en metodologías ágiles como se menciona en los trabajos [47] y [44].
- La columna "Habilidades" considera "Tácticas" y/o "Estrategias" y se considera que se pueden aplicar a metodologías ágiles:
  - Tácticas: se asigna color verde (1) porque es esencial emplear este elemento en métodos ágiles.
  - Estrategias: se asigna color verde (1) porque las metodologías ágiles consideran que el equipo de desarrollo debe tener experiencia para resolver planteamientos generales y/o específicos.

Métodos \ Elementos	Pasos	Roles	Documentación					Requerimientos	Diseño	Análisis	Habilidades	
			Escenarios	Plantillas	Árboles de utilidad	Lluvia de ideas	Diseño				Tácticas	Estrategias
QAW	(2)	(1)	(2)	(2)	(2)	(2)		(2)				
ADD	(2)	(1)	(2)	(2)	(2)	(2)			(2)		(1)	(1)
ATAM	(2)	(1)	(2)	(2)	(2)	(2)				(2)		(1)
V&B				(2)			(1)					

Cuadro 2.5: Elementos de los métodos de arquitectura de software.

Dada las características de los métodos del SEI no es tan sencillo poder introducirlos en alguna metodología aunque tampoco imposible. Se debe tomar en cuenta que no todos los

elementos podrían encajar directamente. Para esto, ajustar estos elementos de tal forma, para que no contrapongan al *Manifiesto Ágil* puede ser la mejor opción.

El uso de los métodos para el diseño de la arquitectura de software en metodologías ágiles no es una práctica común. Como se menciona en el Cuadro 2.1, la arquitectura de software no se toma en cuenta en las metodologías ágiles porque se asume que los desarrolladores son personas expertas y con la experiencia necesaria para ser ágiles en el desarrollo y, emplear explícitamente métodos para el diseño de la arquitectura de software, puede quitar agilidad debido a la cantidad de documentación que se genera al utilizar los métodos para el diseño, obstaculizando la agilidad en las metodologías ágiles, como se expresa en el *Manifiesto Ágil* en el sentido de que se debe valorar más el software funcionando sobre la documentación.

Puede ser complicado introducir explícitamente la arquitectura de software en las metodologías ágiles. Sin embargo, como se expuso en esta sección los beneficios de emplear la arquitectura se verán reflejados en una arquitectura robusta del producto. A su vez esto podría beneficiar a las metodologías ágiles porque en el *Manifiesto Ágil* se menciona que deben estar abiertas a recibir cambios a lo largo del desarrollo, en dónde puede ocurrir que un requerimiento nuevo impacte a la arquitectura y se tenga que hacer cambios importante en el diseño. Por lo tanto, si se cuenta con una arquitectura robusta difícilmente se tendría que elaborar una arquitectura completamente nueva.

Los métodos del SEI mencionados en este trabajo no se podrían aplicar directamente en las metodologías ágiles por lo siguiente:

- Al aplicar las tareas establecidas en cada uno de los métodos se deben obtener los productos necesarios para ejecutar las siguientes tareas o métodos
- Algunos pasos no son tan explícitos y se deben adaptar a alguna metodología
- En algunos casos se debe contratar a personal externo
- Se generan artefactos en los que se tiene que generar documentación de manera formal

Por estos motivos si no se modifican los métodos para el diseño de arquitecturas de software podrían obstaculizar la agilidad en las metodologías ágiles y lo que se busca es que estos métodos sean fáciles de adaptar para no perder agilidad en el desarrollo.

2.3. Estado del arte en arquitecturas de software y métodos ágiles

En esta sección se muestra una recopilación de trabajos que han realizado diversos autores, con planteamientos teóricos y/o empíricos, en los cuales exponen propuestas para incorporar la arquitectura de software en metodologías ágiles específicas.

2.3.1. Arquitectura de software y la metodología ágil Scrum

En el trabajo titulado *Agile Architecture IS Possible You First Have to Believe!*[37], para incorporar la arquitectura de software en Scrum, el autor plantea llevar a cabo una serie de estrategias en base a su experiencia. Lo que propone se menciona a continuación[37]:

- Al hacer una refactorización de la arquitectura se propone: por un lado, plantear la nueva arquitectura de manera invisible y en paralelo con una versión anterior de la

arquitectura; por otro lado, hacer una prueba apoyándose en la nueva arquitectura en conjunto con la ya existente. Si esto no funciona entonces se regresa de la nueva arquitectura a la anterior.

- Al dar una idea de cómo se desea el diseño final, no siempre es necesario que todos los componente sean refactorizados al mismo tiempo. Por otro lado, elementos constitutivos del diseño se dividen en iteraciones, usando plataformas temporales cuando sea apropiado o necesario: proxys, interfaces, etc. Lo anterior, para vincular los componentes viejos a la nueva arquitectura. Cuando todos los sistemas viejos hayan sido convertidos, se eliminan todos los componentes en las plataformas anteriores y se continua con la nueva.

Este trabajo es importante porque el autor está introduciendo el diseño arquitectónico en base a su experiencia con una serie de estrategias y tácticas. Como precondición en las metodologías ágiles se debe contar con personas experimentadas, es decir, deben tener cierto grado experiencia desarrollando software. Por lo tanto, se ven favorecidos en el sentido de poder resolver desde problemas técnicos, hasta planteamientos de aspectos abstractos y complejos.

Las ventajas que presenta este trabajo teniendo como referencia el *Manifiesto Ágil*, son:

- Realizar diseños lo más simple posible sin descuidar que más adelante pueden presentarse problemas que deben resolverse con este diseño.
- Emplear tácticas y estrategias.
- Ser flexible y ser adaptable a los cambios que puedan ir surgiendo a lo largo del desarrollo.

2.3.2. Arquitectura de software y la metodología ágil XP

En la literatura de la metodología ágil XP no hay una especificación de tareas para el desarrollo de arquitectura de software. Para considerar la arquitectura en la metodología XP, los autores del trabajo titulado *Integrating Software-Architecture-Centric Methods into Extremme Programming (XP)*[47] proponen utilizar los siguientes métodos del SEI: Taller de Atributos de Calidad (*Quality Attribute Workshop*, QAW)[12], Diseño Guiado por Atributos (*Attribute-Driven Desing*, ADD)[52], Método de Análisis de Compromisos Arquitecturales (*Architecture Tradeoff Analysis Method*, ATAM)[44], Método de Análisis Costo-Beneficio (*Cost Benefits Analysis Method*, CBAM)[40] y Diseños Intermediarios para Revisiones Activas (*Active Reviews for Intermediate Designs*, ARID)[22]. En el Cuadro 2.6 se presenta un resumen de cómo específicamente los métodos enfocados en la arquitectura propuestos por el SEI, pueden mejorar las actividades de XP. En la primera y segunda columnas, se mencionan y se describen, respectivamente, los métodos del SEI y en la tercera, la función primaria de cada método dentro de XP. Así mismo, en este trabajo se describe cómo cada método del SEI complementa las prácticas de XP. Por ejemplo, QAW incorpora a todos los involucrados de manera temprana en el ciclo de vida para descubrir los atributos de calidad pero emplear este método tiene un impacto en las Prácticas XP: Juego de Planificación, Cliente *in situ*, Desarrollo Guiado por Pruebas, Historias de Usuarios. Por lo tanto, esas prácticas estarían siendo beneficiadas con el método de QAW. Por otro lado el método ADD tendría la función de definir la arquitectura de software pero también tendría un impacto en las Prácticas XP: Juego de Planificación, Metáfora del Sistema, Refactorización, Diseño Simple y Pico Arquitectónico. Y de la misma manera los demás métodos pero con sus prácticas respectivas.

Método	**Tarea primaria**	**Prácticas XP y artefactos que afecta**
QAW	Ayuda a entender las preocupaciones de los interesados en los requerimientos de atributos de calidad	Juego de Planificación, Cliente *in situ*, Desarrollo Guiado por Pruebas, Historias de Usuarios
ADD	Para definir una arquitectura granular	Juego de Planificación, Metáfora del Sistema, Refactorización, Diseño Simple, Pico Arquitectónico
ATAM/CBAM	Para evaluar la arquitectura	Juego de Planificación, Cliente *in situ*, Refactorización, Plan de Liberación
ARID	Evaluar una porción de la arquitectura	Integración Continua, Liberaciones Cortas, Desarrollo Guiado por Pruebas

Cuadro 2.6: Métodos enfocados en la arquitectura de software y actividades de XP[47]

Retomando el *Manifiesto Ágil*, ¿cuál es el conflicto que puede haber con este trabajo? Los autores argumentan que utilizar los métodos del Cuadro 2.6 añaden valor a XP. Es decir, un valor de XP es la comunicación y ese valor se ve beneficiado con la comunicación e interacción que se genera cuando se emplea, en este caso, el método QAW. De manera similar con los demás valores de XP como simplicidad, coraje y retroalimentación que son influenciados con los otros métodos del SEI.

Este trabajo muestra que se pueden ocupar métodos que se enfoquen en la arquitectura de software. Sin embargo, haciendo una revisión de los métodos del SEI se observa que no se pueden aplicar directamente. Un punto en contra, que se encontró en este trabajo, es que los autores no describen claramente la forma de acoplar los elementos de los métodos del SEI y no se percibe como éste acoplamiento afecta a XP y si genera algún conflicto en la congruencia de XP con el *Manifiesto Ágil*.

2.3.3. Arquitectura de software y Cristal

Para abordar la arquitectura de software en la metodología Cristal los autores del trabajo titulado *Adding Agility to Architecture Tradeoff Analysis Method for Mapping on Crystal*[30] proponen modificar el método ATAM, de tal manera que se pueda mapear para poder adaptarlo en Cristal. Este mapeo es crucial para asegurar la calidad de los productos usando metodologías ágiles.

En la Figura 2.10 se muestra cómo se hace el mapeo. Del lado izquierdo tenemos elementos de la metodología ágil Cristal y del lado derecho las fases y pasos del método ATAM. Las flechas indican hacia dónde son mapeados los elementos de Cristal. Una vez realizado el mapeo los autores realizan varias encuestas a diversos desarrolladores de software. Para eliminar elementos de ATAM se realiza una encuesta en la que hacen mención sobre qué fases y pasos, los practicantes de este método, creen que consume más tiempo. Los cuadros de color rojo son las fases/pasos del método que se eliminan para poderlo adaptar.

El enfoque que dan los autores al método ATAM en este trabajo, no es recomendable

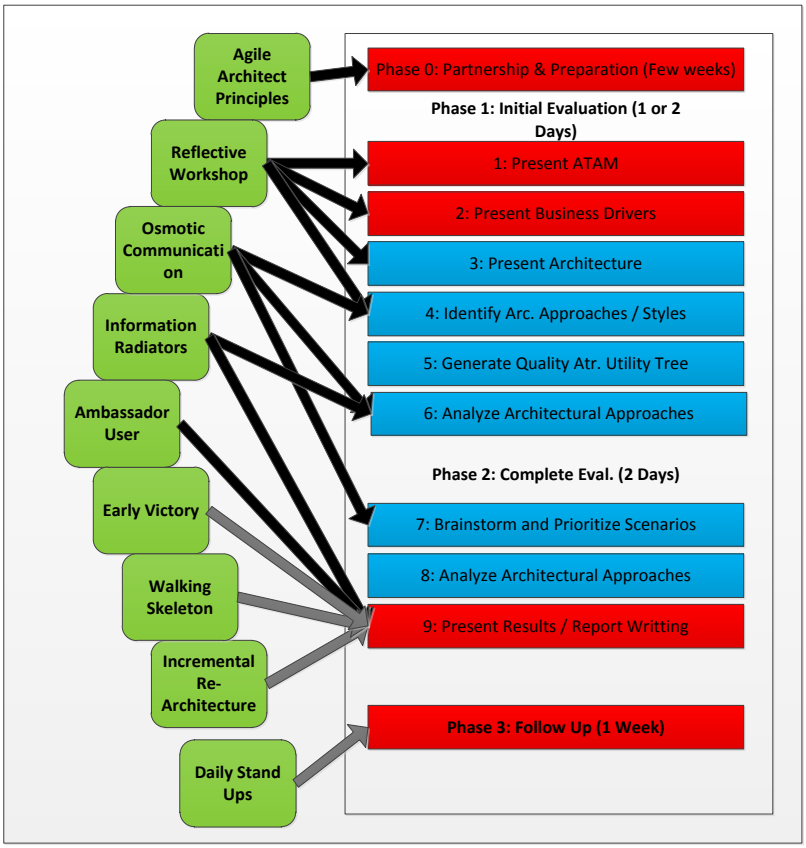

Figura 2.10: ATAM modificado para la metodología ágil Cristal[30]

para el diseño de la arquitectura de software porque ATAM está más enfocado en verificar la arquitectura. Para el desarrollo de este trabajo de tesis este enfoque que utilizan no es adecuado. Sin embargo, la forma de mapear un método arquitectónico hacia una metodología ágil muestra que se debe de analizar ambas partes y se puede integrar partes del método en la metodología.

2.3.4. Documentando arquitectura de software en metodologías ágiles

Los autores del reporte técnico *Documenting Software Architectures in an Agile World*[21] tienen como propósito un enfoque para documentar información acerca de la arquitectura de software de manera que sea consistente con metodologías ágiles. La documentación es una parte fundamental en los procesos de desarrollo de software. Por lo general, la documentación es formal y específica en las metodologías tradicionales, a diferencia de la que se genera en metodologías ágiles. En el Cuadro 2.1 se menciona que en las metodologías ágiles se documenta lo indispensable considerando que uno de los valores del *Manifiesto Ágil* dice que se debe valorar el software funcionando sobre documentación extensiva. Por lo tanto, se debe tener en cuenta que la documentación en una metodología ágil estará presente pero esta debe ser flexible y solo se debe documentar lo indispensable. En ese sentido, introducir la arquitectura de software implica generar documentación extra y en ese aspecto los practicantes de las metodologías ágiles no están totalmente de acuerdo. Este planteamiento pretende capturar información de la arquitectura, utilizando el método V&B, sin entrar en conflicto con el *Ma-*

nifiesto Ágil. Por este motivo, como se plantea en el trabajo, las propuestas de V&B pueden ser un punto de partida para documentar el desarrollo de arquitecturas en métodos ágiles. Sin embargo, se descarta utilizar esta proposición de los autores porque en sus conclusiones mencionan que hasta el momento no se ha puesto en práctica esta propuesta.

2.3.5. Discusión

Desde hace algunos años se han realizado propuestas para introducir el diseño de la arquitectura de software en metodologías ágiles. Los trabajos van desde la propia experiencia del desarrollador hasta la utilización y adaptación de métodos conocidos. En algunos trabajos es importante retomar las ideas que plantearon porque se pueden utilizar para introducir la arquitectura de software en metodologías ágiles en general.

Los trabajos mencionados anteriormente muestran que es posible ocupar elementos de arquitectura de software para emplearlos en metodologías ágiles específicas. Sin embargo, no mencionan cómo se podrían aplicar, de manera explícita, sus propuestas hacia otras metodologías. Es decir, si quisiéramos utilizar metodologías ágiles como las que se mencionan en la sección 2.1.7, los trabajos no son nada específicos en cómo poder aplicar sus propuestas en otras metodologías ágiles. Como podemos observar en la Figura 2.11 cada metodología ágil aplica directamente desde su enfoque hacia elementos de arquitectura de software. Por otro lado, las metodologías comparten características similares entre ellas, pero los autores no generalizan su propuesta hacia otros métodos ágiles. Además, en la Figura 2.11 se puede observar que hay una relación entre la documentación y la arquitectura (ver sección 2.2.3). Sin embargo, el trabajo [21] (ver sección 2.3.4), se puede utilizar para documentar la arquitectura de software a la par de las metodologías y sin especificar alguna metodología en particular.

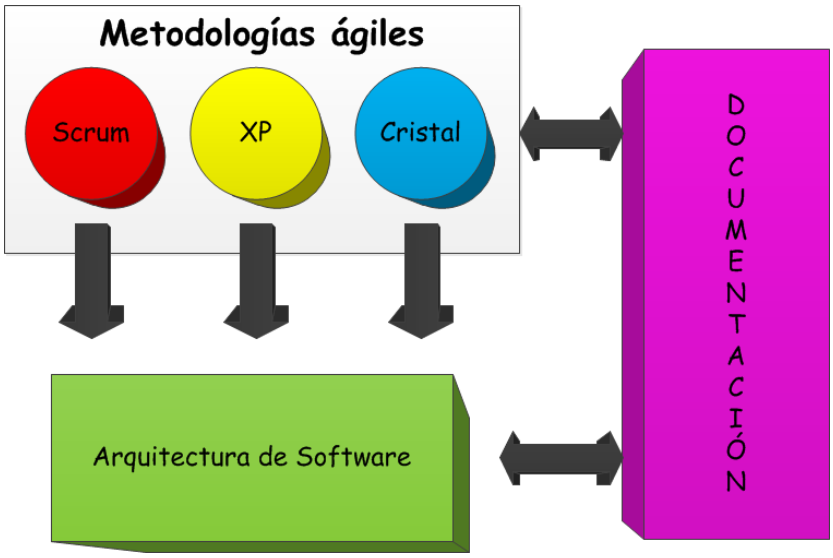

Figura 2.11: Trabajos con relación de metodologías ágiles con arquitectura de software y la documentación.

El Cuadro 2.7 refleja el valor de los trabajos revisados en esta sección. Los datos para elaborar el Cuadro 2.7 fueron extraídos de [37], [47] y [44]. Se asignan los números: 0 (ninguno), 1 (medio) y 2 (alto) que corresponden al valor que se le decidió agregar tomando en cuenta si el trabajo aportaba ideas que puedan ser retomadas más adelante. A continuación se describe el Cuadro 2.7:

- En la fila de "*Scrum*", se asigna el valor de 2 (alto) en la columna "ADD" porque el autor, a pesar de que no utiliza el método ADD en si, se pone en un rol de arquitecto y emplea estrategias/tácticas para realizar el diseño de la arquitectura de software con la metodología ágil *Scrum*. En las demás columnas se asigna el valor 0 (ninguno) dado que la forma en abordar el problema va más enfocado en el diseño de la arquitectura sobre los demás.
- En la fila de "XP" se asigna el valor de 1 (medio) en todas las columnas porque los autores mencionan, sin detallar, que se pueden emplear los métodos del SEI en XP.
- En la fila "Cristal" se asigna el valor de 1 (medio) en la columna "ATAM", porque el planteamiento que hacen los autores es para introducir el método en el sentido de evaluar la arquitectura de software. Sin embargo, como se mencionó en la sección 2.2.4 emplear el método para la evaluación de la arquitectura en metodologías ágiles no tan accesible debido a que se debe contratar a un grupo externo a la organización, diferente al grupo de desarrollo interno, quienes realizan el análisis y evaluación.

Métodos / **Metodologías**	**QAW**	**ADD**	**ATAM**	**CBAM**	**ARID**
Scrum	0	2	0	0	0
XP	1	1	1	1	1
Cristal	0	0	1	0	0

Cuadro 2.7: Métodos arquitectónicos aplicados en metodologías ágiles.

En general, se observa que los trabajos revisados no se comprometen a sugerir la forma de introducir elementos de diseño de arquitecturas de software en metodologías distintas a la que ellos se enfocan. Sin embargo, se pueden retomar elementos planteados en [37] (tácticas y estrategias) y en [21], para el desarrollo de esta tesis.

2.4. Procesos

A lo largo de la historia el ser humano siempre ha necesitado expandir sus capacidades para el desarrollo y creación de nuevas herramientas de trabajo. Desde la aparición de sistemas numéricos la vida del ser humano cambió para siempre. En la actualidad tenemos una enorme cantidad de máquinas las cuales usan símbolos que nos ayudan a contar además de hacer cálculos numéricos muy extensos y programas que facilitan el trabajo cotidiano de las personas en general.

Los dispositivos mecánicos a través de la historia han sido diversos; basta mencionar que su finalidad era computar, pero hay un punto en la historia en que las máquinas empiezan a "evolucionar" tecnológicamente de manera muy significante. En esa evolución encontramos las generaciones de las computadoras: Válvulas de vacío (1946-1957), Transistores (1958-1964), Circuitos integrados (1965-1970) y Computadoras personales (1971-). De estas generaciones resalta que se empiezan a introducir y ocupar lenguajes de programación, como por ejemplo, ALGOL, COBOL y FORTRAN entre otros. A partir de los lenguajes de programación el rumbo del software empieza a cobrar mucho auge. A temprana edad se introduce el término

"Crisis del software" (F. L. Bauer) que más adelante fue usado por Edsger Dijkstra's y que comenta lo siguiente:

> "*La mayor causa es ... Que las máquinas se han convertido en varias órdenes de magnitud más poderosas! Para decirlo bastante claro: cuando no habían máquinas, la programación no era problema en absoluto, cuando nosotros tuvimos pocas computadoras débiles, la programación se convirtió en un problema ligero y ahora que tenemos computadoras gigantescas, la programación se ha convertido en un problema igualmente gigantesco...*"[26].

A partir de esta idea se empezó a introducir lo que se conoce actualmente como la "Ingeniería de Software" y se enfoca en:

- Mejores formas de desarrollar software de calidad (procesos)
- Herramientas (apoyo al proceso)
- Organización del grupo de desarrollo (administración de proyectos)
- Integración de la organización completa (modelos de referencia)

Dada la importancia que tienen los procesos para este trabajo se presentan definiciones y algunos ejemplos de ellos.

2.4.1. Concepto de Proceso

Definiciones de proceso:

1. *Conjunto de fases sucesivas de un fenómeno natural o de una operación artificial*[29].
2. *Conjunto de prácticas relacionadas entre si, llevadas a cabo a través de roles y por elementos automatizados, que utilizando recursos y a partir de insumos producen un satisfactor de negocio para el cliente*[46].
3. *Un proceso es un conjunto de actividades que se ejecutan para lograr un propósito*[11].

Un proceso como tal debe formalizarse a través de una descripción precisa, para que se pueda documentar y comunicar. En su descripción se deben detallar los componentes principales del proceso. Esto incluye el propósito del proceso, los roles que se emplearán, las actividades y tareas que se llevarán a cabo, los objetivos, las entradas y las salidas; entre otros elementos[49].

Una vez definido y documentado el proceso se recomienda realizar una instancia de éste para llevar a cabo las actividades y tareas tomando en cuenta todos los elementos que se describieron.

En las secciones siguientes se revisan y describen dos procesos para retomar elementos que puedan ser útiles en esta tesis.

2.4.2. OpenUp[28]

OpenUp es un proceso iterativo e incremental. Se centra en el trabajo colaborativo para el desarrollo de software. Partes de sus principios comparten la filosofía de las metodologías ágiles.

2.4.2.1. Descripción

OpenUP es un proceso mínimo y suficiente. Es decir, tiene incluido lo esencial y fundamental para llevar a cabo el proceso. No provee lineamientos para todos los elementos que se podrían manejar en un proyecto. Sin embargo, tiene los componentes básicos que pueden servir de base a tareas específicas. La mayoría de los elementos de OpenUP están declarados para fomentar el intercambio de información entre el equipo de desarrollo y así mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

OpenUP, como se observa en la Figura 2.12, se organiza en tres capas: micro-incremento, ciclo de vida de iteración y ciclo de vida del proyecto. La capa micro-incremento, representa unidades cortas de trabajo que se producen a un ritmo constante del progreso de éste (normalmente medido en horas o pocos días). El proceso se aplica con una intensiva colaboración para desarrollar un incremento de un sistema por un equipo auto organizado. Estos micro-incrementos proporcionan una retroalimentación muy corta que impulsa las decisiones de adaptación dentro de cada iteración. La capa ciclo de vida de iteración, se miden normalmente por semanas. El plan de iteración define qué debe ser entregado en cada iteración, el resultado puede ser un demo o un incremento entregable de la aplicación. Finalmente, la capa ciclo de vida del proyecto se mide por meses y se da en cuatro fases: inicio, elaboración, construcción y transición. En cada fase se realizan una serie de tareas y éste tiene que ser visible para que los involucrados y miembros del equipo puedan supervisar y tomar decisiones en el momento oportuno. El Plan de Proyecto define el ciclo de vida y el resultado final es una versión de la aplicación.

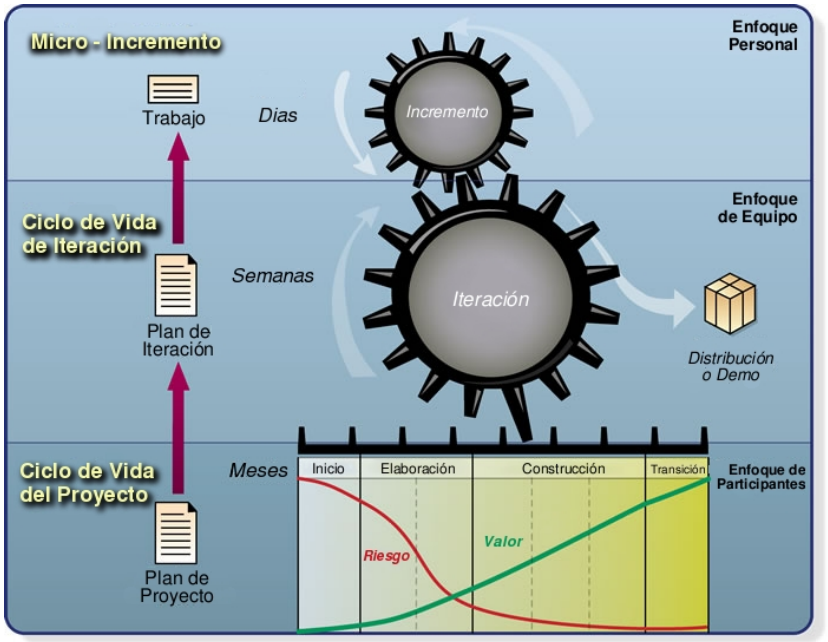

Figura 2.12: Capas de OpenUP: micro-incremento, ciclo de vida de iteración y ciclo de vida del proyecto.

2.4.2.2. Principios de OpenUP

OpenUP se basa en 4 principios:

1. Colaborar para sincronizar intereses y compartir conocimiento:
    - Promueve prácticas que impulsan un ambiente de equipo "saludable".
    - Facilita la colaboración.
    - Desarrollo de conocimiento compartido del proyecto.
2. Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto:
    - Promueve prácticas que permiten desarrollar una solución que maximice los beneficios y que cumpla con los requisitos y restricciones del proyecto.
3. Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
4. Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo:
    - Promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continua de los participantes del proyecto.
    - Permite mostrar incrementos progresivos en la funcionalidad.

2.4.2.3. Elementos básicos

Los elementos básicos de OpenUP son los siguientes:

1. *Producto producido*:
    - Documentos (visión, plan de proyecto).
    - Un modelo (casos de uso, diseño).
    - Bases de datos, hojas de cálculo y otros repositorios de información.
    - El código fuente y los ejecutables.
    - Los productos de trabajo se pueden clasificar como "artefactos" si son cosas concretas, "resultados" si no se concreta, y "entregables" si son un paquete de los artefactos.
2. *Tareas* (¿Cómo realizar el trabajo?):
    - Realizada por un rol.
    - Desarrollar una visión (global, alcance, frontera, etc).
3. *Rol* (¿Quién realiza el trabajo?):
    - Responsabilidades
4. *Proceso*:
    - Utilizado para definir la división y el flujo de trabajo.
    - Procesos de *pull* junto con tareas, producto producido, etc.
5. *Orientación (dirección)*:
    - Plantillas, Listas de verificación, Ejemplos, Directrices, Conceptos, etc.

El propósito de revisar OpenUP es identificar los elementos que lo componen y que puedan ser útiles para la definición y documentación del modelo propuesto en este trabajo.

2.4.3. Proceso Personal de Software (*Personal Software Process*, PSP)[35]

Este proceso siempre se ha caracterizado por tener una estructura sencilla y clara para utilizarlo. Lo que se busca aquí es entender cómo se estructura los guiones para poder emplearlo en la instancia de el modelo genérico a desarrollar hacia una metodología ágil.

2.4.3.1. Descripción

PSP es un proceso personal para desarrollar software o para hacer cualquier otra actividad definida. Incluye una serie de fases, pasos definidos, formas y estándares.

PSP proporciona un marco de trabajo de medición y análisis para caracterizar y administrar trabajo personal. Es un procedimiento definido que ayuda a mejorar el desempeño personal.

En la Figura 2.13 podemos observar el flujo del proceso de PSP. Por un lado, tenemos los guiones de psp. Estos guiones tienen definidos los pasos y actividades que vamos a seguir para realizar la planeación, diseño, codificación, compilación, pruebas y *post mortem*. Por otro lado, tenemos las bitácoras en donde iremos registrando el tiempo que nos tardamos en cada paso. Por último, el resumen del proyecto que tiene los datos del proyecto y del proceso que se fueron elaborando a lo largo del desarrollo.

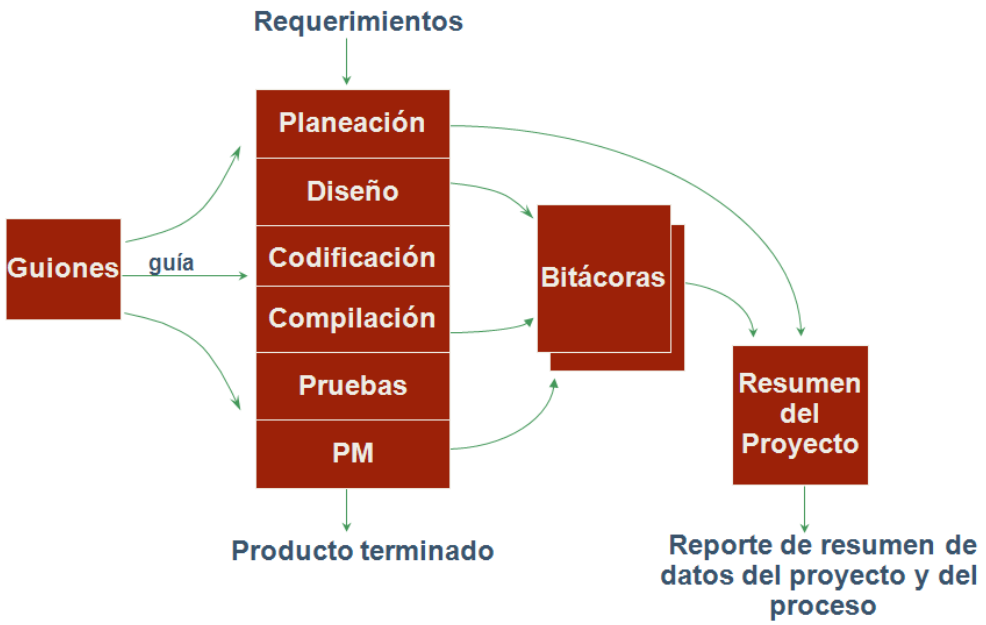

Figura 2.13: Flujo de proceso de PSP.

2.4.3.2. Guiones de Proceso

PSP hace uso de guiones los cuales sirven de guía a través del proceso. Es decir, proporcionan una guía de "nivel-experto" de cómo usar el proceso. Los guiones son de una o dos páginas de extensión y describen: Propósito, Criterio de Entrada, Guías Generales, Pasos y Criterio de Salida.

Para seguir los guiones de PSP se debe:

- Revisar el criterio de entrada antes de iniciar la fase

- Registrar la hora de inicio de la fase
- Efectuar los pasos e instrucciones de la fase
- Registrar los defectos conforme son encontrados y corregidos
- Revisar el criterio de salida antes de concluir la fase
- Registrar la hora de fin de la fase
- Ir a la siguiente fase

Un ejemplo de un guión en PSP se puede observar en la Figura 2.14. Observemos que se define un criterio de entrada y lo que se debe realizar para poder ir paso por paso a través de una serie de actividades. Finalmente, tenemos una salida que consisten en código ya probado y la documentación necesaria.

Propósito	Guiar el desarrollo de programas de nivel módulo
Criterio de Entrada	- Descripción del Problema - Forma de Resumen de Plan de Proyecto PSP0 - Bitácoras de Registro de Tiempo y Defectos - Estándar de Tipos de Defectos - Cronómetro (opcional)

Paso	**Actividades**	**Descripción**
1	Planeación	- Producir u obtener una especificación de requerimientos. - Estimar el tiempo de desarrollo requerido. - Registrar el dato de planeación en la forma de Resumen del Plan de Proyecto. - Completar la bitácora de Registro de Tiempo.
2	Desarrollo	- Diseñar el programa. - Implementar el diseño. - Compilar el programa, corregir y registrar todos los defectos encontrados. - Probar el programa, corregir y registrar todos los defectos encontrados. - Completar la bitácora de Registro de Tiempo.
3	Post mortem	Completar la forma de Resumen del Plan de Proyecto con los datos de tiempo, defectos y tamaño reales.

Criterio de Salida	- Un programa probado ampliamente - Forma del Resumen del Plan de Proyecto completa con los datos estimados y reales - Bitácoras de Registro de Tiempo y Defectos completas

Figura 2.14: Guión de PSP0.

2.4.4. Modelo de Procesos para la Industria del Software (MoProSoft)[46]

Para comprender lo que es MoProSoft se debe entender lo que es un modelo de procesos. Primero se describe la definición de modelo:

- *Un modelo es una representación abstracta que ilustra los componentes o relaciones de una aplicación especifica o componente*[49].
- *Un modelo es un arquetipo o punto de referencia para imitarlo o reproducirlo*[29].

Las dos definiciones son complementarias, por un lado se habla de una representación general de componente o relaciones y por otro, se menciona que es un punto de referencia para imitarlo.

Por otro lado, la definición de modelo de procesos es: *Un conjunto estructurado de elementos que describen las características de procesos efectivos y de calidad, a su vez se debe indicar qué es lo que se hace*[46].

2.4.4.1. Descripción

MoProSoft surge del Programa para el Desarrollo de la Industria del Software (PROSOFT)[25] que tuvo su origen de una iniciativa del gobierno de México. Desde el año 2002 se propuso crear una norma que tome en cuenta un modelo de procesos, modelo de capacidades de procesos y método de evaluación. En base a esto surge MoProSoft, para responder a las necesidades de la industria mexicana para el desarrollo de software de alta calidad y considera un modelo de procesos que integra un modelo de capacidades en la versión coloreada. Para las organizaciones que formalmente no aplican un modelo de procesos, MoProSoft puede ser la opción que permita elevar la capacidad de las organización para ofrecer el desarrollo de productos con calidad y alcanzar niveles internacionales de competitividad[46].

2.4.4.2. Estructura

MoProSoft está constituido por nueve procesos distribuidos en tres categorías:

1. Alta Dirección: Aborda las prácticas relacionadas con la gestión del negocio. Proporciona lineamientos a los procesos de la Categoría de Gerencia y se retroalimenta con la formación generada por ellos. El proceso que incluye es:

    - Gestión de Negocio: Su propósito es establecer la razón de ser de la organización, sus objetivos y las condiciones para lograrlos.

2. Gerencia: Aborda las prácticas de gestión de procesos, proyectos y recursos en función de los lineamientos establecidos en la Categoría de Alta Dirección. Proporciona los elementos para el funcionamiento de los procesos de la Categoría de Operación, recibe y evalúa la información por éstos y comunica los resultados a la Categoría de Alta Dirección. Los procesos que incluye son:

    - Gestión de Procesos: Establece los procesos de la organización, así como definir, planificar e implantar las actividades para mejorar los mismos.
    - Gestión de Proyectos: Asegura que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la organización.
    - Gestión de Recursos: Asegura que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la organización. El proceso de Gestión de Recursos se compone de tres subprocesos:
        - Recursos Humanos y Ambiente de Trabajo: Proporciona los recursos humanos y adecuados para cumplir las responsabilidades asignadas a los roles dentro de la organización, así como la evaluación del ambiente de trabajo.
        - Bienes, Servicios e Infraestructura: Proporciona proveedores de bienes, servicios e infraestructura que satisfagan los requisitos de adquisición de los procesos y proyectos de la organización.
        - Conocimiento de la Organización: Mantener disponible y administrar la base de conocimiento que contiene la información y los productos generados por la organización.

3. Operación: Aborda las prácticas de los proyectos de desarrollo y mantenimiento de software. Además, realiza las actividades de acuerdo a los elementos proporcionados por la Categoría de Gerencia y entrega a ésta la información y productos generados. El procesos de Operación tiene dos subprocesos:

- Administración de Proyectos Específicos: Establece y lleva a cabo sistemáticamente las actividades que permiten cumplir con los objetivos de un proyecto en tiempo y costo esperados.
- Desarrollo y Mantenimiento de Software: Realiza de manera sistemática las actividades de análisis, diseños, construcción, integración y pruebas de productos de software nuevos o modificados cumpliendo con los requerimientos especificados.

2.4.4.3. Niveles de capacidad

La versión 1.3 coloreada de MoProSoft utiliza colores para indicar el nivel de capacidad. Esto se puede ver en el Cuadro 2.8. Los colores indican el orden de la implementación de las prácticas de los procesos de MoProSoft, desde las prácticas bases que corresponde al nivel 1 hasta las que corresponden a niveles más avanzados. También se alcanza a observar la correspondencia entre los niveles de capacidades de procesos y los colores.

Nivel	Capacidad de proceso	Color
1	Realizado	amarillo
2	Gestionado	azul
3	Establecido	verde
4	Predecible	rosa
5	Optimizado	ninguno

Cuadro 2.8: Niveles de capacidades de MoProSoft[46].

2.4.4.4. Patrón de procesos

Todos los procesos definidos en MoProSoft siguen una misma estructura y como elemento adicional dentro de MoProSoft se define un patrón que puede ser utilizado para definir cualquier proceso que sea requerido en una organización.

El patrón de procesos es un esquema de elementos que sirve para la documentación de los procesos de MoProSoft o de algún proceso de una organización y está constituido por tres partes:

- **Definición general del proceso:** Se identifica el nombre, la categoría a la que pertenece, el propósito, la descripción general de sus actividades, los objetivos, los indicadores, las metas cuantitativas, la responsabilidad y autoridad, los subprocesos en caso de tenerlos, los procesos relacionados, las entradas, las salidas, los productos internos y las referencias bibliográficas.

- **Prácticas:** Se identifican los roles involucrados en el proceso y la capacitación requerida, se describen las actividades en detalle, asociándolas a los objetivos del proceso, se presenta un diagrama de flujo de trabajo, se describen las verificaciones y validaciones requeridas, se listan los productos que se incorporan a la base de conocimiento, se identifican los recursos de infraestructura necesarios para apoyar las actividades, se establecen las mediciones del proceso, así como las prácticas para la capacitación, manejo de situaciones excepcionales y uso de lecciones aprendidas.

- **Guías de ajuste:** Se sugieren modificaciones al proceso que no deben afectar los objetivos del mismo.

2.4.5. Discusión

Los procesos son un conjuntos de prácticas relacionadas entre si para lograr un propósito. Son indispensables en muchas áreas de trabajo pero en especial en el desarrollo de software. Los procesos que se consideraron en esta tesis son: OpenUP y PSP. Como se menciona en la sección 2.4.2, OpenUp es un proceso iterativo e incremental, mínimo y suficiente. Parte de sus principios se ven reflejados en las metodologías ágiles. Es interesante que OpenUP mencione en sus principios que se debe considerar la arquitectura de software desde el inicio del desarrollo. Esta idea se puede retomar cuando se defina el modelo de esta tesis. Como se menciona en 2.4.3, PSP es un proceso personal de software. PSP emplea guiones que guían a través del proceso. Éstos son fáciles y sencillos de seguir porque establecen los criterios de entrada y salida, los pasos y actividades que se deben seguir para el desarrollo. Estos guiones pueden resultar provechosos para realizar el desarrollo del modelo relacionado en este trabajo de tesis.

Los procesos de MoProSoft fueron definidos utilizando el patrón de procesos. Éste patrón de procesos, se caracteriza por tener un esqueleto práctico y simple dividido en tres partes que facilitan la definición de los procesos. El patrón de procesos se puede adecuar a las necesidades que se requieran para el modelo relacionado a este trabajo lo que permitirá definir el modelo para integrar, de manera explícita, el diseño de arquitectura de software en metodologías ágiles.

A continuación el Cuadro 2.9 se presentan los:

- Principios de OpenUP.
- Elementos de las guías de PSP.
- Elementos de la plantilla de MoProSoft.

Principios de OpenUP	**Guías PSP**	**Plantilla MoProSoft**
▪ Colaborar para sincronizar intereses y compartir conocimiento ▪ Equilibrar las prioridades para maximizar el beneficio por los interesados en el proyecto ▪ Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo ▪ Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo	▪ Propósito ▪ Criterio de entrada ▪ Actividades ▪ Criterio de salida	▪ Nombre del proceso ▪ Propósito ▪ Descripción ▪ Objetivos ▪ Indicadores ▪ Entradas ▪ Salidas ▪ Referencias bibliográficas

Cuadro 2.9: Elementos considerados útiles para el propósito de éste trabajo de tesis.

Como podemos observar en el Cuadro 2.9 se muestran los principios de OpenUP. El principio que se puede retomar es el que menciona que se debe considerar la arquitectura de software desde un inicio. Para esta tesis se considerará cuando se defina el modelo referido a este trabajo. En la columna guías PSP, tenemos los elementos de las guías. Estos elementos pueden ser convenientes para definir pasos y actividades a realizar. Finalmente, en la columna plantilla MoProSoft se tienen los elementos del patrón de procesos. Estos elementos se podrían utilizar para definir el modelo relacionado de este trabajo.

Los conceptos revisados en la sección 2.4 pueden acelerar la definición del modelo referido en este trabajo y por eso no se descarta la utilización de los elementos presentados en el Cuadro 2.9.

2.5. Planteamiento del problema

Las metodologías ágiles, con base a su manifiesto, expresan que se debe valorar a los individuos y sus interacciones sobre los procesos y herramientas, al software funcionando sobre documentación extensa, la colaboración del cliente sobre la negociación del contrato y la respuesta al cambio sobre el seguimiento de un plan. Además, de un conjunto de 12 principios[17] que describen el enfoque que debe regir el desarrollo de software bajo una metodología ágil.

Hay una gran variedad de metodologías ágiles y cada una tiene sus características particulares (ver sección 2.1). Además, en cada metodología se parte del supuesto de que se cuenta con expertos técnicos e individuos muy motivados. Es decir, cada integrante debe tener ciertas habilidades para resolver desde problemas técnicos, hasta planteamientos de aspectos abstractos y complejos, por lo que la experiencia debe ser fundamental.

Por otro lado, las metodologías definen ciclos, que en algunos casos incluyen varias fases. En las metodologías ágiles, también se genera documentación, sin embargo, el proceso de documentación es menos rígido, porque por lo general se documenta únicamente lo indispensable[7, 30]. De los aspectos que más llaman la atención de las metodologías ágiles es el hecho que están abiertas a recibir cambios en los requerimientos, sin importar, si estos cambios tienen un impacto negativo en la arquitectura del sistema. Sin embargo, se ha afirmado que en las metodologías ágiles el diseño de la arquitectura de software no es una actividad importante[10]. Es decir, existe la tendencia a no tomarlo en cuenta el diseño arquitectónico. Explícitamente, en las metodologías ágiles no hay una fase para el diseño de la arquitectura como en el caso de las metodologías tradicionales. Esto implica que desde el momento en que se empieza a escribir código, la arquitectura se va creando en la medida en que se va desarrollando el proyecto.

Cómo se mencionó anteriormente, los cambios en los requerimientos a lo largo del proyecto es algo natural e inevitable, por lo que, dependiendo del cambio se puede tener un gran impacto en la arquitectura. En este sentido, pueden surgir nuevos requerimientos que podrían necesitar de una nueva arquitectura o la realización de cambios significativos, de tal manera que los atributos de calidad que se pudiesen tener en la arquitectura original, se pierdan.

Anteriormente, se ha mencionado que las metodologías ágiles comparten entre ellas ciertas características. Por tal motivo, la introducción de un modelo que pueda definir elementos genéricos para establecer las directrices necesarias e incorporar explícitamente arquitectura de software tendría un impacto bastante positivo en las metodologías ágiles, considerando que el modelo que se proponga en esta tesis no debe contraponerse en gran medida con los valores y principios expuestos en el *Manifiesto Ágil*[16].

Los trabajos que se han realizado para atacar el problema de la arquitectura de software en las metodologías ágiles es escaso. El valor que aportan estos trabajos consiste en proponer estrategias para el diseño de una buena arquitectura. Por otro lado, también hacen la propuesta de usar métodos establecidos por el SEI para integrar la arquitectura de software con las metodologías ágiles. Además, de una descripción muy breve de: QAW, ADD, ATAM y V&B para rescatar elementos que puedan servir para el modelo a desarrollar.

2.6. Objetivos

Con base a lo anterior, se plantean los objetivos de este trabajo:

- **Objetivo general:**
  - Desarrollar un modelo que permita establecer las directrices necesarias para incorporar, explícitamente y en forma genérica, la arquitectura de software en metodologías ágiles.
- **Objetivos específicos:**
  - Entender las bases comunes de los métodos ágiles para conocer todos sus elementos.
  - Revisar métodos de desarrollo de la arquitectura de software para encontrar puntos clave y su posible adaptación a metodologías ágiles.
  - Analizar diferentes procesos para poder rescatar elementos valiosos que permitan documentar los resultados a obtener en este proyecto.
  - Desarrollar y probar con una instancia, el modelo genérico a elaborar en este proyecto, para incorporar el desarrollo de arquitecturas de software en métodos ágiles.

Capítulo 3

Metodología y herramientas

3.1. Metodología

Las revisiones y discusiones en el capítulo 2 marcaron el punto de partida para obtener las bases y fundamentos para el desarrollo de este trabajo. A continuación se hace un recuento para rescatar los puntos importantes:

- Por un lado, las metodologías ágiles comparten características similares porque están alineadas a los valores y principios del *Manifiesto Ágil*. Por otro lado, no hay una fase específica para el diseño de la arquitectura de software por lo que esta práctica es poco común entre sus practicantes.

- No se puede aplicar directamente los métodos arquitectónicos en las metodologías ágiles. Para que esto pueda hacerse, antes se deben analizar sus elementos para adaptarlos.

- Los trabajo revisados en el estado del arte no son claros en cómo redirigirlos hacia otras metodologías ágiles pero muestran que si se pueden introducir elementos arquitectónicos en metodologías ágiles.

- La plantilla de MoProSoft proporciona un esquema para definir un proceso y las guías de PSP son convenientes para definir pasos y actividades a realizar.

De acuerdo a los objetivos planteados en esta tesis se debe proponer una estrategia para poder lograr dichos objetivos. Además, documentar la metodología permite definir, de manera formal, los pasos a seguir que al aplicarlos darán los mismos resultados. En la metodología se definen una serie de actividades, tareas y productos a lograr, como se muestra en la Figura 3.1 y que se describe a continuación. En la Figura 3.1, las actividades se representan con las elipses en color amarillo y los cuadros representan los productos. Los cuadros en color verde representan los productos más importantes en este trabajo de tesis. En la Figura 3.1 se observa que de las actividades se generan productos que sirven de entrada a actividades posteriores. Las tareas que se describen a continuación corresponden a las actividades de la Figura 3.1.

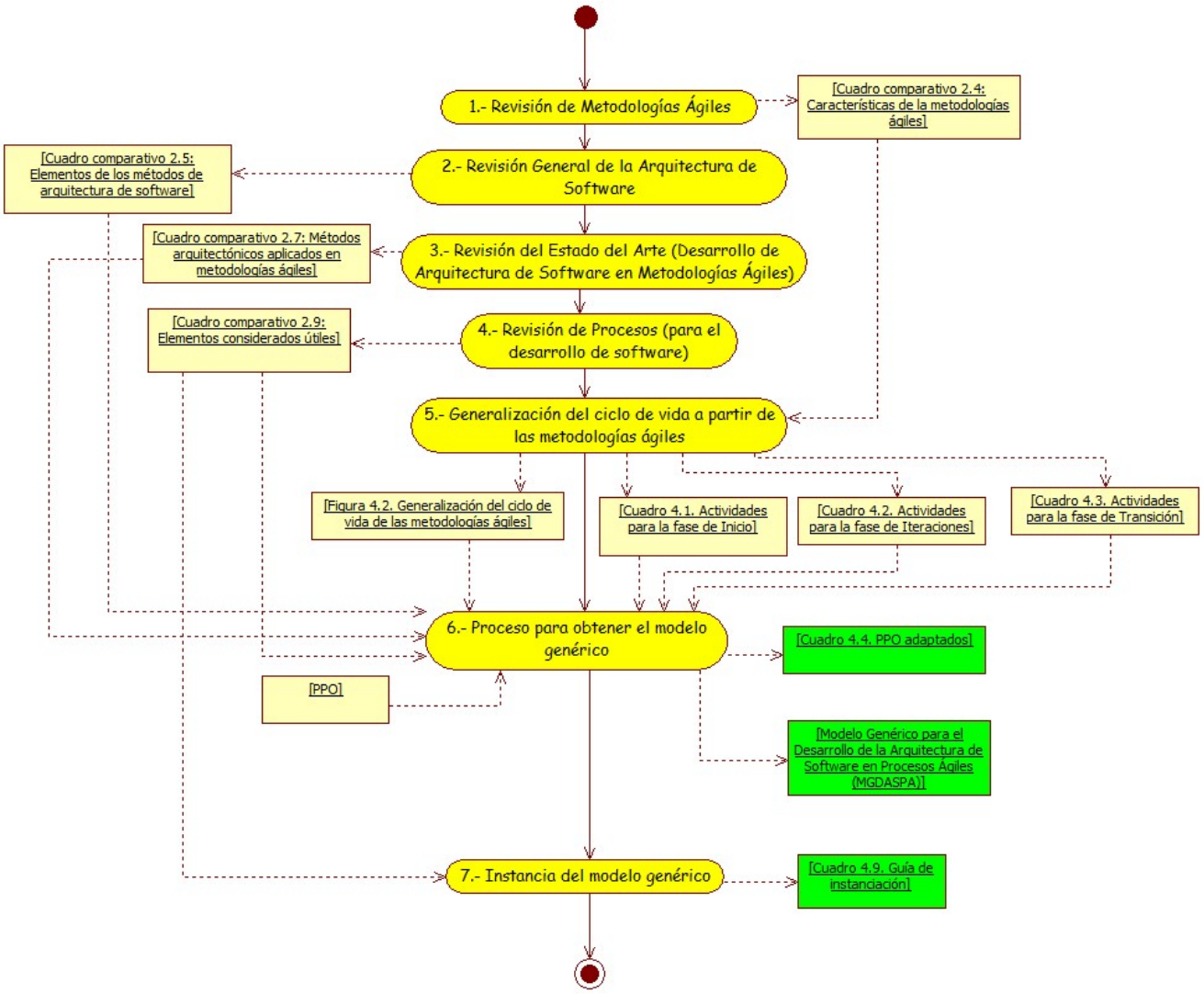

Figura 3.1: Diagrama de actividad para representar la metodología propuesta para el desarrollo del modelo.

3.1.1. Actividad uno: Revisión de Metodologías Ágiles

En esta actividad se debe realizar una revisión de las metodologías ágiles siguiendo estas tareas:

1. Comprender la definición y el enfoque de las metodologías ágiles.
2. Revisar las metodologías ágiles más importantes, que se están utilizando actualmente.
3. Comprender el ciclo de vida de las metodologías ágiles revisadas.
4. Identificar los elementos y características de las metodologías ágiles revisadas.
5. Comparar las metodologías ágiles con respecto a sus características.

Como se observa en la Figura 3.1, al concluir esta actividad se obtiene como producto "Características de las metodologías ágiles" (ver Cuadro 2.4).

3.1.2. Actividad dos: Revisión General de la Arquitectura de Software

En esta actividad se debe realizar una revisión de conceptos, métodos y características de arquitectura de software, considerando las siguientes tareas:

1. Comprender los conceptos de arquitectura de software.
2. Revisar métodos para el desarrollo de arquitectura de software, que se estén utilizando actualmente.
3. Revisar y comprender los métodos para el desarrollo de la arquitectura de software.
4. Identificar los elementos y características de los métodos para el desarrollo de la arquitectura de software.
5. Clasificar los elementos de los métodos para la arquitectura de software.
6. Comparar los elementos en el desarrollo de la arquitectura de software que puedan fortalecer las metodologías ágiles.

Como se observa en la Figura 3.1, al concluir esta actividad se obtiene como producto "Elementos de los métodos de arquitectura de software" (ver Cuadro 2.5).

3.1.3. Actividad tres: Revisión del Estado del Arte (Desarrollo de Arquitectura de Software en Metodologías Ágiles)

En esta actividad se debe realizar una revisión del estado del arte de acuerdo a las siguientes tareas:

1. Buscar en bases de datos trabajos sobre el tema "Desarrollo de arquitectura de software en metodologías ágiles".
2. Comprender el problema que plantean los autores y el enfoque que dan para resolverlo.
3. Identificar qué tan aplicable es el resultado de esos trabajos para problemas con contextos similares.
4. Identificar elementos que se puedan reutilizar en el proyecto de tesis y aspectos que no se abordan dentro del desarrollo de arquitectura de software.

Como se observa en la Figura 3.1, al concluir esta actividad se obtiene como producto "Métodos arquitectónicos aplicados en metodologías ágiles" (ver Cuadro 2.7).

3.1.4. Actividad cuatro: Revisión de Procesos (para el desarrollo de software)

En esta actividad se debe realizar una revisión de procesos de acuerdo a las siguientes tareas:

1. Comprender el concepto de procesos para el desarrollo de software.

2. Revisar y comprender procesos, con elementos mínimos, para el desarrollo de software.

3. Identificar elementos útiles para definir el modelo genérico.

Como se observa en la Figura 3.1, al concluir esta actividad se obtiene como producto "Elementos considerados útiles" (ver Cuadro 2.9).

3.1.5. Actividad cinco: Generalización del ciclo de vida a partir de metodologías ágiles en uso

En esta actividad se debe realizar la generalización del ciclo de vida de varias de las metodologías ágiles en uso y se divide en dos tareas:

1. Correlación y clasificación de las fases de las metodologías ágiles para generalizar el ciclo de vida:

   a) Realizar una selección de las metodologías ágiles a relacionar y asignar un identificador a cada una de ellas.

   b) Analizar e identificar las actividades en las fases del ciclo de vida de las metodologías ágiles seleccionadas.

   c) Asignar un nombre general o representativo a las actividades analizadas e identificadas, en el punto anterior, que tengan la misma finalidad, el mismo objetivo o la misma tarea.

   d) Verificar que existe correlación entre las fases de los métodos ágiles seleccionados, comparando las fases de una metodología con las fases de otra.

   e) Clasificar las fases correlacionadas del punto anterior tomando en cuenta lo siguiente:

      - Fases iniciales correlacionadas donde se realizan tareas como: obtención de requerimientos, planificación, misión, alcance, etc. Después, asignar un nombre representativo para esa fase.
      - Fases intermedias correlacionadas donde se realizan tareas relacionadas al diseño, iteraciones de desarrollo, implementación, etc. Después, asignar un nombre representativo para esa fase.
      - Fases finales correlacionadas donde se realizan entrega de producto, puesta en operación, etc. Después, asignar un nombre representativo para esa fase.

   f) Describir el ciclo de vida generalizado utilizando las fases clasificadas del punto anterior.

   Como se observa en la Figura 3.1, al concluir esta tarea se obtendrá el producto "Generalización del ciclo de vida de las metodologías ágiles".

2. Actividades de la generalización:

   a) Retomar la información y producto:

      - Nombres generales o representativos.
      - Clasificación de las fases correlacionadas.
      - Descripción del ciclo de vida generalizado.

b) Generar cuadros de actividades para cada fase de la generalización con lo retomado en el punto anterior.

Como se observa en la Figura 3.1, al concluir esta tarea se obtendrán los productos "Actividades para la fase de Inicio" (ver Cuadro 4.1), "Actividades para la fase de Iteraciones" (ver Cuadro 4.2) y "Actividades para la fase de Transición" (ver Cuadro 4.3).

3.1.6. Actividad seis: Proceso para obtener el modelo genérico

En esta actividad se debe diseñar el modelo genérico. Para describir el proceso para obtener el modelo genérico se utilizan dos elementos:

1. Plantilla de MoProSoft (patrón de procesos, ver sección 2.4.4). Esta plantilla propone un esquema para documentar los procesos de MoProSoft. Sin embargo, para describir el proceso general se adaptaron los elementos de la plantilla considerando que lo que se describe en este trabajo es un modelo y no un proceso. En la Figura 3.2, se muestran en color verde los elementos que se emplean para la descripción del proceso y en color rojo los elementos que se descartan por no ser indispensables para la descripción.

2. Paquetes de Puesta en Operación (PPO, ver sección 3.2.2) adaptados. Esta herramienta se utiliza para introducir actividades para el desarrollo de la arquitectura de software en el modelo genérico.

Figura 3.2: Plantilla de MoProSoft adaptada.

3.1.6.1. Definición general del proceso

Como se observa en la Figura 3.2, los elementos elegidos que componen la definición general del modelo son:

- **Proceso:** este elemento se utiliza para definir el nombre que lleva el proceso genérico.
- **Propósito:** este elemento se utiliza para establecer el objetivo general y el resultado esperado de la implantación del modelo.
- **Descripción:** este elemento se utiliza para describir las actividades y productos que componen el flujo de trabajo del modelo.
- **Objetivos:** este elemento se utiliza para definir los objetivos específicos del modelo para asegurar el cumplimiento del propósito. Los objetivos se identifican como 01, 02, etc.
- **Indicadores:** este elemento se utiliza para definir los indicadores que evalúan el cumplimiento de los objetivos del modelo. Los indicadores se identifican como I1, I2, etc.
- **Entradas:** este elemento se utiliza para establecer el nombre del producto o recurso que se necesita como entrada para el modelo genérico.
- **Salidas:** este elemento se utiliza para establecer los productos que se obtienen.
- **Referencias bibliográficas:** este elemento se utiliza para la bibliografía.

Cabe aclarar que se renombra el elemento "Proceso" con el de "Modelo" porque lo que se describe en este trabajo es un modelo.

3.1.6.2. Proceso

El nombre del proceso con su respectivo acrónimo es:

- **PROGMGDASMA. Proceso para Generar el Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles**

3.1.6.3. Propósito

El propósito de PROGMGDASMA es definir elementos generales que permitan establecer las directrices necesarias para incorporar explícitamente la arquitectura de software en metodologías ágiles, sin contraponerse con los valores y principios expresados en el *Manifiesto Ágil*.

Introducir en forma explícita actividades de desarrollo de arquitectura de software a metodologías ágiles.

3.1.6.4. Descripción

PROGMGDASMA se compone de las siguientes etapas a partir de la generalización del ciclo de vida de las metodologías ágiles: Inicio, Iteraciones y Transición. Mismas etapas que se definieron en la Figura 4.2. A continuación, se describen las actividades generales que deben realizarse en cada etapa de PROGMGDASMA enfocándonos en aquellas (en negritas) que promueven el desarrollo de la arquitectura de software:

1. Inicio:
    - Misión
    - **Visión**

- Definir el alcance del proyecto
- Realizar estimaciones acerca del proyecto
- **Requerimientos**
- Realizar una planificación
- Identificar riesgos del proyecto
- Establecer los roles de los integrantes del equipo
- Objetivos del proyecto
- Priorizar los requerimientos con el cliente
- **Arquitectura inicial**
- Establecer acuerdos con el cliente

2. Iteraciones:

   - Mantener participación constante con los involucrados
   - Realizar reuniones con el equipo
   - Construir las iteraciones
   - Realizar la planificación
   - Realizar retroalimentación con el equipo de desarrollo
   - Codificar
   - Realizar los cambios
   - Realizar las pruebas
   - **Arquitectura**
   - **Selección de requerimientos**

3. Transición:

   - Generar la documentación del sistema
   - Realizar la puesta en operación del sistema
   - Reuniones con los involucrados
   - Documentar manuales del sistema
   - Identificar defectos y mejoras para el sistema.

3.1.6.5. Objetivos

O1 Identificar actividades y etapas de la generalización del ciclo de vida de las metodologías ágiles donde pueda incluirse actividades de desarrollo de arquitectura de software.

O2 Definir actividades para asignar y adaptar las tareas de los PPO (Cuadro 4.4), en su respectiva etapa de la generalización del ciclo de vida de las metodologías ágiles.

O3 Verificar que las actividades asignadas y tareas adaptadas de los PPO, definidas en las etapas de la generalización del ciclo de vida de las metodologías ágiles, en su conjunto, no se contrapongan con el *Manifiesto Ágil*.

3.1.6.6. Indicadores

I1 (O1) Lista de actividades para el desarrollo de la arquitectura de software y su correspondiente etapa de la generalización del ciclo de vida de las metodologías ágiles.

I2 (O2) Lista de etapas, de la generalización del ciclo de vida de las metodologías ágiles, afines a las tareas de los PPO.

I3 (O2) Los PPO tienen asignado una etapa de la generalización del ciclo de vida de las metodologías ágiles.

I4 (O2) Lista de actividades definidas para el desarrollo de la arquitectura de software, de la generalización del ciclo de vida de las metodologías ágiles, con tareas adaptadas de los PPO.

I5 (O3) Las actividades y las tareas adaptadas respetan los valores y principios del *Manifiesto Ágil*.

3.1.6.7. Entradas

Nombre	**Fuentes**
Valores y Principios	*Manifiesto Ágil*
Actividades de Arquitectura de Software	Paquetes de Puesta en Operación (PPO) para la norma ISO/IEC 29110
Ciclo de vida de las metodologías ágiles	Actividades de la generalización del ciclo de vida de las metodologías ágiles

Cuadro 3.1: Entradas de PROGMGDASMA.

3.1.6.8. Salidas

Nombre	**Descripción**	**Destino**
MGDASMA	Contiene una generalización del ciclo de vida de las metodologías ágiles mejorada con las actividades necesarias para incorporar el desarrollo de la arquitectura de software sin contraponerse con el *Manifiesto Ágil*.	Guía de instanciación.

Cuadro 3.2: Salidas de PROGMGDASMA.

3.1.6.9. Referencias bibliográficas

- *Manifesto for Agile Software Development*[16].
- Erick Andrey Serratos Álvarez. Paquetes de Puesta en Operación (PPO) para la norma ISO/IEC 29110.
- Patrón de procesos de MoProSoft [46].

3.1.6.10. Prácticas

Como se observa en la Figura 3.2, los elementos elegidos que componen las prácticas del modelo son:

- **Roles involucrados y capacitación:** este elemento se utiliza para identificar los roles involucrados y capacitación requerida.
- **Actividades:** este elemento se utiliza para describir las tareas y los roles responsables.
- **Diagramas de flujo de trabajo:** este elemento se utiliza para describir mediante diagramas de actividades de UML las actividades del flujo de trabajo.
- **Verificaciones y validaciones:** este elemento se utiliza para definir las verificaciones y validaciones asociadas a los productos generados en las actividades.
- **Mediciones:** este elemento se utiliza para evaluar los indicadores del modelo.

3.1.6.11. Roles involucrados y capacitación

Rol	**Abreviatura**	**Capacitación**
Responsable de PROGMGDAS-MA	RM	Conocimiento de las actividades necesarias para definir e instanciar exitosamente el modelo hacia una metodología ágil.
Verificador del PROGMGDAS-MA	VM	Conocimiento de los PPO, actividades y etapas de la generalización del ciclo de vida de las metodologías ágiles enfocadas al desarrollo de la arquitectura de software.

Cuadro 3.3: Roles de PROGMGDASMA.

3.1.6.12. Actividades

Las actividades de PROGMGDASMA se presentan en el Cuadro 3.4.

A1. Identificación de actividades y etapas de la generalización (O1).	
Roles	**Descripción**
VM	A1.1. Revisar las actividades y etapas de la generalización del ciclo de vida de las metodologías ágiles.
VM	A1.2. Elaborar una lista de actividades, de la generalización del ciclo de vida de las metodologías ágiles, que promuevan el desarrollo de la arquitectura de software.
VM	A1.3. Verificar la lista de actividades que promueven el desarrollo de la arquitectura de software (**Ver1**).
RM	A1.4. Validar la lista de actividades que promueven el desarrollo de la arquitectura de software (**Val1**).
VM	A1.5. Elaborar una lista de las etapas, de la generalización del ciclo de vida de las metodologías ágiles, para el desarrollo de la arquitectura de software.

VM	A1.6. Verificar la lista de etapas para el desarrollo de la arquitectura de software (**Ver2**).
RM	A1.7. Validar la lista de etapas para el desarrollo de la arquitectura de software(**Val2**).
A2. Adaptación de las tareas de las PPO en la generalización (O2).	
Roles	**Descripción**
VM	A2.1. Revisar la lista de actividades que promueven el desarrollo de la arquitectura de software.
VM	A2.2. Revisar la lista de etapas para el desarrollo de la arquitectura de software.
VM	A2.3. Revisar la lista de tareas de los PPO analizados con el *Manifiesto Ágil* y sus correspondientes etapas en la generalización.
VM	A2.4. Definir los nombres de las actividades que tendrán las tareas de los PPO.
VM	A2.5. Asignar e integrar las tareas de los PPO a las actividades de la generalización del ciclo de vida de las metodologías ágiles.
VM	A2.6. Adaptar las tareas de los PPO tomando en cuenta lo expresado en el *Manifiesto Ágil*.
VM	A2.7. Verificar las actividades con las tareas adaptadas de los PPO en las etapas de la generalización del ciclo de vida de las metodologías ágiles (**Ver3**).
RM	A2.8. Validar las actividades definidas con las tareas adaptadas de los PPO en las etapas de la generalización del ciclo de vida de las metodologías ágiles (**Val3**).
A3. Verificación de actividades con tareas adaptadas de los PPO (O3).	
Roles	**Descripción**
VM	A3.1. Realizar una lista de verificación para revisar si las actividades definidas con tareas de los PPO, en su conjunto, no se contraponen con el *Manifiesto Ágil*.
VM	A3.2. Verificar la lista de verificación (**Ver4**).
RM	A3.3. Aprobar la lista de verificación (**Val4**).

Cuadro 3.4: Actividades para el desarrollo de la arquitectura de software de PROGMGDASMA.

3.1.6.13. Diagrama de flujo de trabajo

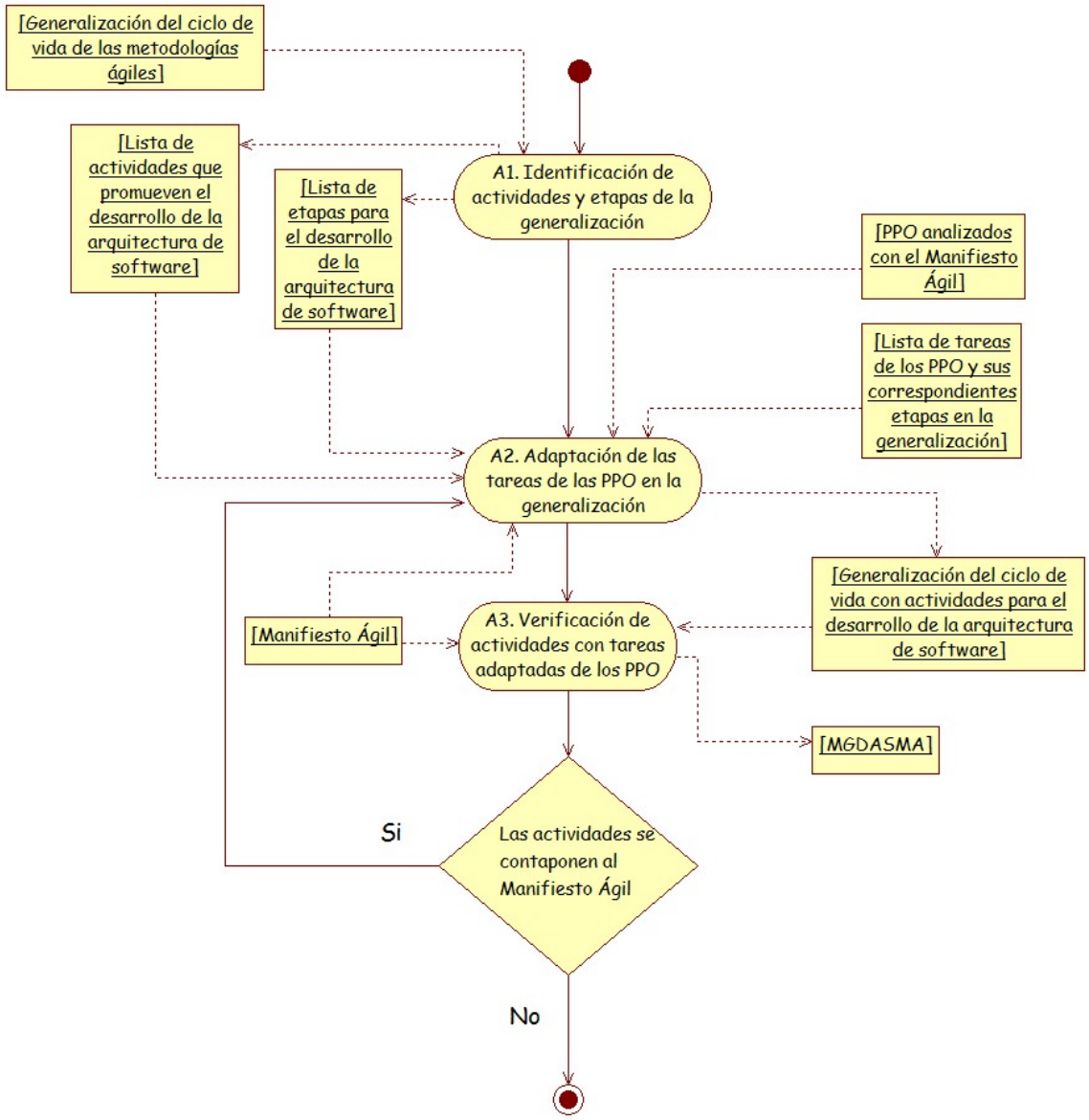

Figura 3.3: Diagrama de flujo de trabajo de PROGMGDASMA.

3.1.6.14. Verificaciones y validaciones

El Cuadro 3.5, muestra las verificaciones o validaciones asociadas a los productos generados generados en las actividades.

Verificación o validación	**Actividad**	**Producto**	**Rol**	**Descripción**
Ver1	A1.3	Lista de actividades que promueven el desarrollo de la arquitectura de software.	VM	Verificar que se identifiquen las actividades para el desarrollo de la arquitectura de software en la generalización.
Val1	A1.4	Lista de actividades que promueven el desarrollo de la arquitectura de software.	RM	Validar que las actividades identificadas sean las necesarias para el desarrollo de la arquitectura de software en la generalización.
Ver2	A1.6	Lista de etapas para el desarrollo de la arquitectura de software.	VM	Verificar que se identifiquen las etapas para el desarrollo de la arquitectura de software en la generalización.
Val2	A1.7	Lista de etapas para el desarrollo de la arquitectura de software.	RM	Validar que las etapas identificadas sean las correctas para el desarrollo de la arquitectura de software en la generalización.
Ver3	A2.7	Actividades definidas con las tareas adaptadas de los PPO.	VM	Verificar que las actividades definidas para el desarrollo de la arquitectura de software tengan asignadas las tareas adaptadas de los PPO.
Val3	A2.8	Actividades definidas con las tareas adaptadas de los PPO.	RM	Validar que las actividades definidas para el desarrollo de la arquitectura de software tengan asignadas las tareas adaptadas de los PPO.
Ver4	A3.2	Lista de verificación	VM	Verificar que las actividades definidas con tareas adaptadas de los PPO no se contrapongan con el *Manifiesto Ágil*.
Val4	A3.3	Lista de verificación	RM	Validar que las actividades definidas con tareas adaptadas de los PPO no se contrapongan con el *Manifiesto Ágil*.

Cuadro 3.5: Entradas de PROGMGDASMA.

3.1.6.15. Mediciones

M1 (I1) Revisar y comprobar la lista de actividades para el desarrollo de la arquitectura de software de la generalización.

M2 (I2, I3 y I4) Revisar y comprobar que estén disponibles:

- Lista de etapas afines a las tareas de los PPO.
- PPO tienen asignado una etapa de la generalización.
- Listas de actividades definidas con tareas adaptadas de los PPO.

M5 (I5) Revisar y comprobar que las actividades no se contrapongan al *Manifiesto Ágil*.

Como se observa en la Figura 3.1, al concluir esta actividad se obtienen los productos en color verde "PPO adaptados" y "Modelo Genérico para el Desarrollo de la Arquitectura de Software en Procesos Ágiles (MGDASMA)".

3.1.7. Actividad siete: Guía de instanciación

En esta actividad se plantea una guía para poder realizar una instancia de MGDASMA a una metodología ágil. En esta guía se describen los pasos y actividades que se tienen que llevar a cabo para fortalecer una metodología ágil con actividades para la desarrollo de la arquitectura de software.

En la actualidad hay una gran variedad de metodologías ágiles: Scrum, XP, Kaban, ASD, AUP, DSDM, Cristal Claro, FDD, etc. Cada metodología ágil tienen características que comparten entre ellas (ver Cuadro 2.4). Esto se debe a que las metodologías ágiles están alineadas al *Manifiesto Ágil*, el cual establece valores y principios que se deben seguir para desarrollar software de manera ágil.

MGDASMA define elementos generales que permiten establecer directrices para incorporar explícitamente la arquitectura de software en metodologías ágiles, sin contraponerse con los valores y principios expresados en el *Manifiesto Ágil*.

Para plantear la guía de instanciación se retoma los elementos considerados útiles, específicamente, las guías PSP (ver Cuadro 2.9), porque dicho elementos nos permiten definir los pasos y las actividades necesarias para la instancia de MGDASMA a una metodología ágil.

Los elementos de las guías PSP que se emplearán son:

- **Propósito:** se utiliza para definir el propósito de la guía de instanciación.
- **Criterio de entrada:** se utiliza para describir los elementos que se necesita para empezar.
- **Pasos:** se utiliza para definir los pasos a seguir.
- **Actividades:** se utiliza para especificar las actividades a realizar en cada paso.
- **Descripción:** se utiliza para describir las tareas de cada actividad.
- **Criterio de salida:** se utiliza para describir los resultados que se obtienen después de aplicar la guía.

En el Cuadro 3.6, se presenta la guía de instanciación:

Guía de instanciación		
Propósito		Introducir actividades para la elaboración de arquitectura de software en metodologías ágiles
Criterio de entrada		▪ Documentación de la metodología ágil específica ▪ MGDASMA
Paso	**Actividades**	**Descripción**
1	Conocer MGDASMA	Es necesario tener conocimiento de la estructura de MGDASMA, principalmente sus actividades en las etapas de Inicio e Iteraciones.
2	Conocer la metodología ágil	Es necesario conocer, en la metodología ágil, lo siguiente: ▪ Ciclo de vida de la metodología ágil ▪ Valores y/o principios de la metodología ágil ▪ Elementos y características de la metodología ágil ▪ Roles de la metodología ágil ▪ Actividades, tareas, y/o prácticas de la metodología ágil Es importante comprender las actividades, tareas y/o prácticas que componen a la metodología ágil por lo que es recomendable tener experiencia utilizando la metodología ágil que se elija para realizar la instancia.
3	Identificar fases y actividades	Identificar las fases y actividades para el desarrollo de la arquitectura de software de la metodología elegida con las etapas y actividades definidas de la generalización. Este paso es casi inmediato dado que ya se tiene identificado las fases de XP. Esto se debe por la sección 2.1.7.2 y la actividad número cinco de la metodología (ver sección 3.1.5).

4	Introducir actividades, sus equivalencias y sus relaciones	Incluir las actividades de MGDASMA: ▪ Acuerdos con el cliente ▪ Arquitectura Inicial ▪ Iniciar taller ▪ Obtener escenarios de atributos de calidad. ▪ Diseño arquitectónico. en sus respectiva fase de la metodologías ágil elegida. También se debe especificar cuál actividad de la metodología ágil es la equivalente en las actividades de MGDASMA mencionadas anteriores. Cabe aclarar que se debe ser cuidadoso con las actividades de la generalización del ciclo de vida a las que se les agregó tareas de los PPO como: Acuerdos con el cliente y Arquitectura Inicial. Después de eso se debe establecer las relaciones entre esas actividades.
Criterio de salida		▪ Metodología ágil robustecida con actividades para desarrollo de la arquitectura de software con atributos de calidad

Cuadro 3.6: Guía para realizar la instancia de MGDASMA a una metodología ágil específica.

Esta del Cuadro 3.6, tiene la finalidad de mostrar cómo realizar la instancia de MGDASMA a una metodología ágil de una forma sencilla y fácil.

Como se observa en la Figura 3.1, al concluir esta actividad se obtendrá el producto "Guía de instanciación".

3.2. Herramientas

Para la elaboración y administración de este trabajo se emplearon varias herramientas que se muestran en las siguientes secciones.

3.2.1. Herramientas de Software

En esta sección se presentan las herramientas que se utilizó para la elaboración de la planificación y los diagramas de este trabajo.

3.2.1.1. OpenProj

OpenProj[1] es proyecto de código abierto (*open source*) de administración de proyectos, similar a Microsoft Project[2]. OpenProj se ejecuta con la máquina virtual de Java, lo que permite ejecutarlo en diferentes sistema operativos.

OpenProj se utilizó para documentar la planificación, visualización y entrega de este trabajo. Esta herramienta fue de mucha utilidad para poder definir el tiempo que se asignó para cada actividad y/o tarea en esta tesis. Se procedió a realizar una lista de tareas y entregables los cuales se dividieron en fases para saber por dónde empezar y en dónde terminar cada parte de la investigación. Conforme se avanzó en el proyecto se iba registrando el avance de las tareas.

3.2.1.2. StarUML

StarUML[3] es un proyecto de código abierto que permite generar diagramas de UML (*Unified Modeling Language*).

Esta herramienta se empleó para generar vistas significativas para describir la metodología y el modelo a desarrollar.

3.2.2. Paquetes de Puesta en Operación como herramienta para el diseño arquitectónico

Los **Paquetes de Puesta en Operación (PPO)**[**53**] son dos de guías propuestas para facilitar la puesta en operación de las tareas propuestas para el desarrollo de la arquitectura de software en la norma ISO/IEC 29110. Los PPO utilizadas en este trabajo de tesis son los siguientes:

1. **Obtención de Requerimientos de Atributos de Calidad (RAC):** Guía que define actividades para la obtención de requerimientos de atributos de calidad ofreciendo como resultado un conjunto de escenarios de atributos de calidad, necesarios para el diseño de la arquitectura de software.

2. **Diseño, Documentación y Evaluación Arquitectónica basados en Atributos de Calidad (AAC):** Guía que propone actividades que se deben de aplicar de manera iterativa para obtener el diseño y la documentación de la arquitectura de software y que concluye hasta que se logra una arquitectura que satisface los principales requerimientos de atributos de calidad.

Los roles que se emplean en los PPO RAC y ACC son:

1. **Facilitador (F)**: es el conductor y moderador del Taller.

2. **Arquitecto (AQ)**: Encargado del diseño y documentación de la arquitectura de software del proyecto en cuestión.

3. **Equipo de Desarrollo (ED)**: Administradores de proyecto, líderes de proyecto, desarrolladores, probadores, arquitectos, etc.

[1] http://sourceforge.net/projects/openproj/
[2] www.microsoft.com/project
[3] staruml.sourceforge.net/

4. **Clientes (CL)**: conjunto de personas interesadas en la adquisición del producto de software para su organización.

5. **Representante de los clientes (RC)**: participante que representa a los clientes para exponer las necesidades del negocio

6. **Participante (PT)**: Son las personas que asisten al taller. Estos pueden ser los clientes, usuarios, el equipo de desarrollo.

Esta herramienta se empleó para agregar elementos arquitectónicos a las metodologías ágiles a partir de la generalización del ciclo de vida. Con el PPO número uno, se obtienen escenarios de atributos de calidad que sirven de entrada para el PPO número dos. El PPO número dos, se centra en el diseño, documentación y evaluación arquitectónica a partir de los escenarios de atributos de calidad del PPO número uno. Se resalta que para la elaboración de las PPO el autor adaptó métodos del SEI como: QAW, ADD, ARID y V&B .

3.2.3. Herramientas para el diseño del modelo

3.2.3.1. Patrón de procesos

El **patrón de procesos** de MoProSoft es una plantilla que se utilizó para definir los procesos de MoProSoft. MoProSoft, define un esquema de elementos está constituido por tres partes: definición general del proceso, prácticas y guías de ajuste. Esta herramienta se utilizó para definir y documentar el modelo propuesto en este trabajo.

3.2.3.2. Guiones PSP

PSP utiliza **guiones** que sirven para guiar al desarrollador a través del proceso. Pero no solo se le puede dar esta utilidad. Es decir, no solo se centra al desarrollo de software sino que también puede emplearse para describir los pasos a realizar en otra actividad.

En la guía de instanciación de este trabajo se utilizó los guiones de PSP porque son simples y fáciles de utilizar, y definen un propósito, criterios de entrada y salida, una serie de pasos y guías generales.

3.2.4. Discusión

En este capítulo se presentó la metodología y las herramientas para la elaboración y administración de esta tesis.

Por un lado, la metodología que se propuso tuvo que desarrollarse minuciosamente. Esto permitió que se lograrán los objetivos de esta tesis. La forma en cómo se elaboró esta metodología permite que se pueda emplear y se obtengan los mismos resultados. Por este motivo, dicha metodología tiene gran impacto en el capítulo 4, dónde se presentan los resultados de este trabajo de tesis. Observemos que en la Figura 3.1, desde las primeras actividades se fueron obteniendo resultados que serían ocupados en las últimas actividades. Por ejemplo:

- Al finalizar la actividad uno se obtiene un producto que será utilizado en la actividad cinco.

- Al finalizar las actividades dos, tres y cuatro se obtienen productos que serán utilizados en la actividad seis.

Esto permitió tener mayor claridad en lo que se desarrolló en este trabajo.

Por otro lado, se utilizaron varias herramientas las cuales tuvieron un grado de importancia diferente. Por ejemplo:

- *Openproj*: sin esta herramienta no se hubiera podido visualizar el alcance de este trabajo de tesis. Tampoco, se habría visualizado los tiempos y productos que se generarían. Dicha herramienta permitió administrar el proyecto de la mejor manera posible.
- StarUML: esta herramienta permitió elaborar los diagramas de las Figuras 3.1 y 3.3. De no emplear la herramienta hubiera sido más complicado describir las secciones de las Figuras.

Las herramientas anteriores tuvieron un impacto medio en este trabajo. Sin embargo, las siguientes herramientas tuvieron un impacto grande:

- PPO: esta herramienta es clave para este trabajo de tesis. Permitió ahorrar tiempo y de no utilizarse se habría tenido que buscar otras alternativas para lograr lo que plantean los PPO.
- Plantilla de MoProSoft: esta plantilla es la que permitió elaborar el proceso para obtener el modelo genérico de este trabajo de tesis.
- Guiones PSP: estos guiones fueron indispensables para la elaboración de la guía de instanciación del modelo genérico.

Capítulo 4

Desarrollo del Modelo

Al ejecutar los primeros pasos de la metodología para este trabajo de tesis (ver sección 3.1) se obtuvieron resultados útiles para los siguientes pasos de la metodología:

1. Cuadro 2.4: Características de las metodologías ágiles.
2. Cuadro 2.5: Elementos de los métodos de arquitectura de software.
3. Cuadro 2.7: Métodos arquitectónicos aplicados en metodologías ágiles.
4. Cuadro 2.9: Elementos de procesos considerados útiles para el propósito de este trabajo de tesis.

En las siguientes secciones se presentan los resultados obtenidos al ejecutar la metodología a partir del paso cinco.

4.1. Generalización del ciclo de vida a partir de metodologías ágiles en uso

En esta sección, se presentan los elementos básicos de la generalización a partir del análisis de varias metodologías ágiles en uso, los resultados son :

- Correlación y clasificación de las fases de las metodologías ágiles para generalizar el ciclo de vida.
- Ciclo de vida correlacionado de las metodologías ágiles elegidas. Este resultado fue utilizado como referencia para la introducción de actividades de desarrollo de arquitectura de software.

Los resultados obtenidos al ejecutar el paso cinco, de la metodología (ver sección 3.1), son considerados los más importantes para el desarrollo del modelo de este trabajo de tesis y continuación se presentan.

4.1.1. Correlación y clasificación de las fases de las metodologías ágiles para generalizar el ciclo de vida

El resultado generado representa una generalización del ciclo de vida de las metodologías ágiles y a través de él se pueden identificar, de manera general, las fases que componen a las metodologías ágiles y sus actividades.

En la Figura 4.1 se presenta la correlación y la clasificación de las fases del ciclo de vida de las metodologías ágiles:

- Por un lado, en la columna "Metodologías Ágiles" se presentan cinco metodologías: *Scrum*[8], XP[4], ASD[41], AUP[9] y *Kanban*[2]. Por otro lado, en la columna "Fases del ciclo de vida" se presentan las fases de esas metodologías.

- Algunas metodologías ágiles definen fases agrupadas en otras más generales. Por ejemplo: en XP hay tres fases agrupadas en "Mantenimiento", en ASD hay dos fases agrupadas en "Especulación" y dos más en "Aprender" y en *Kanban* hay dos fases agrupadas en "Desarrollo". Estas agrupaciones son casos particulares del ciclo de vida de esas metodologías ágiles.

- Se observa en la Figura 4.1 que cada metodología ágil usa nombres particulares para designar sus fases. Sin embargo, eso no significa que en las fases de cada metodología ágil se realicen actividades diferentes a las de otras. Por ejemplo, se identificó que la fase "Incepción" de AUP contiene actividades que son equivalentes a las fases "Iniciación" y "Planificación" de ASD. Pero también, esas actividades de AUP se especifican en *Scrum* en las fases "Iteración -1" e "Iteración 0".

- En la Figura 4.1, las flechas punteadas de color negro, bidireccionales, indican que se pueden correlacionar las fases de una metodología ágil con las de otras. Esto es porque las fases comparten en gran medida actividades similares. Por ejemplo, en la fase "Incepción" de AUP se revisa qué tan viable es el proyecto. También, se obtiene la financiación inicial del proyecto; entre otras cosas. En la fase "Iteración -1" de *Scrum*, de manera similar, se revisa qué tan viable es el proyecto y se obtiene una financiación inicial al hacer una priorización de los proyectos que existen, de los cuales se elegirá uno para desarrollar. Por lo tanto, si una fase está correlacionada con otra significa que esas fases comparten actividades similares.

- En la Figura 4.1, las líneas punteadas de color rojo agrupan fases semejantes en cada metodología ágil. En ese sentido se obtiene una clasificación de tres etapas: "Inicio", "Iteraciones" y "Transición", las cuales se describen a continuación. Cabe aclarar que una etapa incluye varias fases correlacionadas de las metodologías ágiles.

4.1.2. Ciclo de vida correlacionado de las metodologías ágiles elegidas

En la Figura 4.1, se observan tres etapas que están divididas por las líneas rojas punteadas. Los etapas corresponden a:

- **Etapa de Inicio:**
  - Es la primer etapa del ciclo vida correlacionado que comprende fases y sus actividades relacionadas a: la visión, alcance del proyecto, motivación, entre otras actividades.

- **Etapa de Iteraciones:**
  - Es la etapa intermedia del ciclo de vida correlacionado que comprende fases y sus actividades relacionadas al: análisis/diseño, implementación y pruebas.

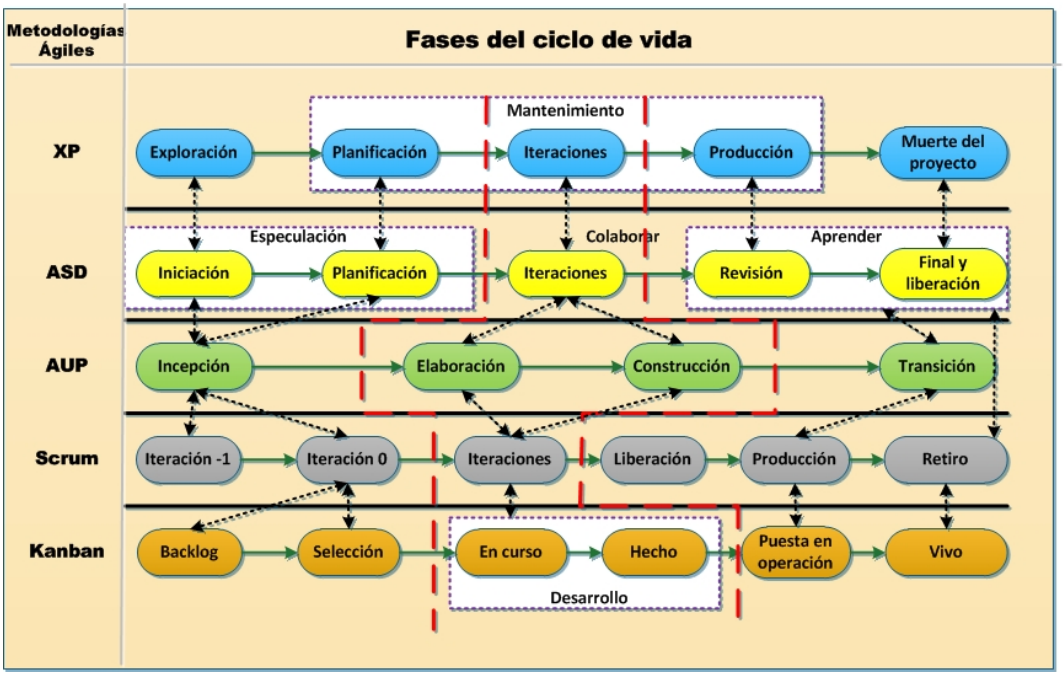

Figura 4.1: Correlación y clasificación de las fases de las metodologías ágiles.

- **Etapa de Transición:**
  - Es la etapa final del ciclo de vida correlacionado que comprende fases y sus actividades relacionadas: liberación, validación y entrega del software, entre otras actividades.

Las etapas descritas anteriormente se pueden observar en la Figura 4.2. Esta Figura representa la generalización del ciclo de vida de las metodologías ágiles partiendo de la correlación y clasificación de las fases de las metodologías ágiles presentada en la Figura 4.1.

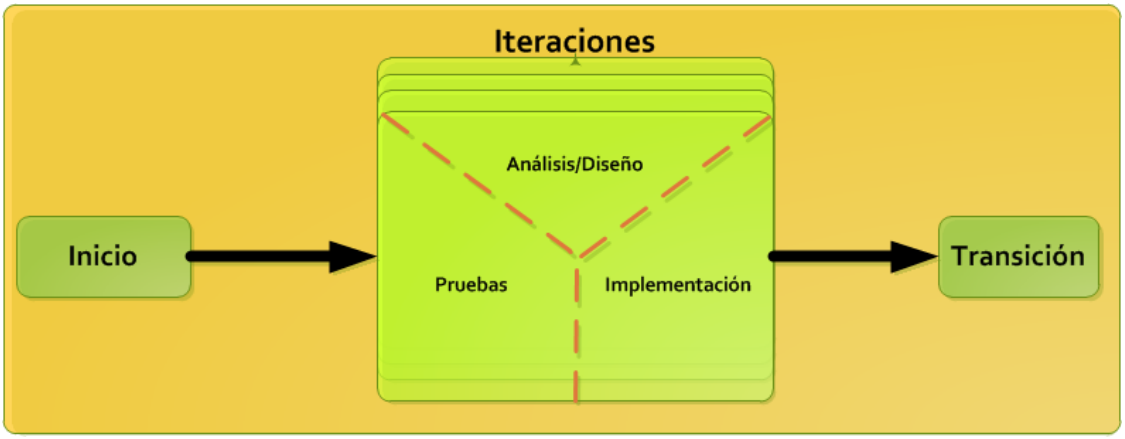

Figura 4.2: Generalización del ciclo de vida de las metodologías ágiles.

Para simplificar el ciclo de vida correlacionado se ha planteado que una etapa contiene varias fases correlacionadas con las metodologías ágiles en estudio. Por lo tanto, se ha consi-

derado que cada etapa rescate las actividades de las fases correlacionadas sin considerar esas fases. De esta forma las siguientes secciones se plantean bajo esta consideración.

4.1.3. Etapas y actividades a partir de la generalización

Las etapas y actividades de la generalización se han obtenido como consecuencia de la correlación aplicada anteriormente y se presentan en los Cuadros 4.1, 4.2 y 4.3. Cada Cuadro representa la etapa correspondiente a partir de la generalización y las actividades que se realizan en cada metodología ágil con base a la correlación. Como se mencionó anteriormente, cada metodología ágil emplea nombres específicos para sus fases y actividades. Para presentar las actividades de las etapas en los cuadros se han usado nombres más generales. Los colores asignados en las celdas surgen en consecuencia a la revisión de las metodologías ágiles dónde:

- Color verde (1): significa que esta actividad es explícita para la metodología ágil correspondiente.
- Color amarillo (2): significa que esta actividad se menciona pero no es explícita en la especificación de la metodología ágil.
- Color rojo (3): significa que esta actividad no se menciona en la especificación de la metodología ágil, pero puede ser que esté implícita.

El detalle de la relación entre las actividades de cada etapa se observan en los Cuadros 4.1, 4.2 y 4.3 respectivamente.

Etapa de Inicio	Scrum	XP	ASD	AUP	Kanban
Misión	(3)	(3)	(1)	(1)	(3)
Visión	(1)	(2)	(1)	(1)	(1)
Alcance	(1)	(1)	(1)	(1)	(1)
Estimación	(1)	(1)	(2)	(2)	(2)
Requerimientos	(1)	(1)	(1)	(1)	(1)
Planificación	(1)	(1)	(1)	(1)	(1)
Riesgos	(3)	(3)	(1)	(2)	(3)
Roles	(1)	(1)	(2)	(1)	(1)
Objetivos	(3)	(3)	(1)	(1)	(3)
Priorización	(1)	(1)	(1)	(2)	(1)
Restricciones	(3)	(3)	(1)	(1)	(2)
Arquitectura Inicial	(1)	(1)	(1)	(1)	(1)
Acuerdos con el cliente	(1)	(1)	(1)	(1)	(1)
Financiamiento inicial	(1)	(3)	(3)	(1)	(3)

Cuadro 4.1: Actividades para la etapa de Inicio.

Cabe aclara que en la etapa de Iteraciones representada en el Cuadro 4.2, las actividad de arquitectura no son explícitas en cuatro de las metodologías ágiles a excepción de AUP. Aquí podemos resaltar de manera inmediata que éste trabajo de tesis tiene como objetivo principal hacer explícitas las actividades de desarrollo de la arquitectura de software a través de un modelo genérico.

Etapa de Iteraciones	Scrum	XP	ASD	AUP	Kanban
Participación con los involucrados	(1)	(1)	(1)	(1)	(1)
Reuniones	(1)	(2)	(1)	(2)	(2)
Construcción de las iteraciones	(1)		(1)	(2)	(1)
Planificación	(1)	(1)	(2)	(2)	(1)
Retroalimentación	(1)	(1)	(1)	(2)	(1)
Codificación	(1)	(1)	(1)	(1)	(1)
Cambios	(2)	(1)	(1)	(2)	(2)
Revisiones	(1)	(2)	(1)	(3)	(1)
Inspecciones	(1)	(2)	(1)	(3)	(1)
Pruebas	(1)	(1)	(2)	(2)	(1)
Arquitectura	(3)	(3)	(3)	(2)	(3)
Selección de requerimientos	(1)	(1)	(3)	(3)	(1)

Cuadro 4.2: Actividades para la etapa de Iteraciones.

Etapa de Transición	Scrum	XP	ASD	AUP	Kanban
Reuniones	(1)	(3)	(1)	(3)	(1)
Documentación	(2)	(2)	(2)	(2)	(2)
Validación	(3)	(1)	(3)	(1)	(2)
Puesta en operación	(1)	(1)	(2)	(1)	(1)
Identificar defectos y mejoras	(1)	(1)	(3)	(2)	(3)

Cuadro 4.3: Actividades para la etapa de Transición.

4.2. Paquetes de Puesta en Operación analizados con el Manifiesto Ágil

Para describir las actividades del modelo genérico de este trabajo primero es necesario adaptar los PPO tomando como referencia al *Manifiesto Ágil*. El Cuadro 4.4, muestra los dos PPO adaptados:

- En la columna "Identificador de tarea", se presenta el identificador que se asignó a la tarea.
- En la columna "Descripción y justificación de la tarea", se presenta la tarea y la justificación para su adaptación.
- En la columna "Color de adaptación", se asigna un color que indica:
  - Color verde: la tarea es muy afín al *Manifiesto Ágil* y se puede utilizar directamente sin ninguna adaptación. Sin embargo, no se descarta que el equipo considere necesario realizar cambios.
  - Color amarillo: sólo parte de la tarea es afín al *Manifiesto Ágil* y se puede utilizar parcialmente, y en caso de ser necesario se deben adaptar pasos de la tarea o simplemente quitarlos.

- Color rojo: la tarea no es afín al *Manifiesto Ágil* y no se debe utilizar.

PPO: Obtención de Requerimientos de Atributos de Calidad[53]		
Identificador de tarea	**Descripción y justificación de la tarea**	**Color de adaptación**
RAC1	Elaboración de una guía de participante: ▪ Omitir esta tarea. Considerando que la comunicación con el cliente es importante esta tarea se debe omitir. Sin embargo, si se desea realizar se debe dar una breve plática al cliente de los conceptos básicos que se manejarán en el proyecto.	**Rojo**
RAC2	Preparación de entradas, documentación y plan arquitectónico inicial: ▪ Utilizar la tarea directamente.	**Verde**
RAC3	Preparación del Taller de Obtención de Atributos de Calidad: ▪ Adaptar la tarea: • El paso 1, se puede omitir o mencionar muy brevemente a los participantes. • El paso 3, se debe realizar de manera muy breve y lo más rápido posible. • El paso 5, se puede omitir en caso de no haber realizado la guía del participante (RAC1).	**Amarillo**
RAC4	Presentación e introducción del Taller: ▪ Utilizar la tarea directamente.	**Verde**
RAC5	Presentación del negocio: ▪ Utilizar la tarea directamente.	**Verde**
RAC6	Presentación de elementos arquitectónicos iniciales: ▪ Utilizar la tarea directamente.	**Verde**
RAC7	Presentación de directrices arquitectónicas: ▪ Utilizar la tarea directamente.	**Verde**
RAC8	Lluvia de escenarios de atributos de calidad: ▪ Utilizar la tarea directamente.	**Verde**
RAC9	Consolidación de escenarios de atributos de calidad: ▪ Utilizar la tarea directamente.	**Verde**
RAC10	Establecimiento de prioridades de escenarios de atributos de calidad: ▪ Utilizar la tarea directamente.	**Verde**
RAC11	Cierre del Taller.	**Verde**

RAC12	Refinamiento de escenarios de atributos de calidad: ▪ Adaptar la tarea: • En el paso 1, lo más recomendable es dejar la plantilla solamente con lo siguiente: Estímulo, Respuesta y Ambiente. • Si el equipo considera que es necesario agregar más elementos de la plantilla puede hacerlo.	**Amarillo**
PPO: Diseño, Documentación y Evaluación Arquitectónica basados en Atributos de Calidad[53]		
Tarea	**Desarrollo**	**Color de adaptación**
AAC1	Confirmar que hay suficiente información sobre los requerimientos: ▪ Utilizar la tarea directamente.	**Verde**
AAC2	Elegir un elemento del sistema a descomponer: ▪ Utilizar la tarea directamente.	**Verde**
AAC3	Identificar las directrices arquitectónicas del elemento en descomposición: ▪ Utilizar la tarea directamente.	**Verde**
AAC4	Elegir un concepto de diseño que satisfaga las directrices arquitectónicas: ▪ Utilizar la tarea directamente.	**Verde**
AAC5	Descomponer el elemento en elementos más pequeños: ▪ Utilizar la tarea directamente.	**Verde**
AAC6	Hacer una instancia de los elementos de software y asignarles funcionalidad: ▪ El paso 4, describe que se debe analizar y documentar las decisiones de diseño. Se pueden utilizar vistas arquitectónicas y no en detalle.	**Verde**

AAC7	Definir interfaces para los elementos arquitectónicos: ▪ Adaptar la tarea: • El paso 1 de esta tarea describe que se debe realizar un caso de prueba de los requerimientos funcionales relacionados con la tarea AAC6 para después en el paso 2 observar y registrar información producida por un elemento y consumida por otro. Lo recomendable para estos pasos es no entrar en una documentación detalla dado que en este caso lo que se pretende es definir los servicios provistos y requeridos por cada elemento de software. • El paso 3 describe que se debe documentar las observaciones y se recomienda que la documentación sea detallada. Este paso entra en conflicto con el *Manifiesto Ágil*. Por lo tanto, se debe documentar lo indispensable para lograr esta tarea. • Las metodologías ágiles parten del supuesto que se cuenta con gente con experiencia. En esta tarea se puede llegar a generar documentación extensiva. En vez de eso, el arquitecto debe enfocarse en el objetivo de la tarea y debe expresarlo con los integrantes del equipo de la mejor manera posible.	**Amarillo**
AAC8	Verificar que los requerimientos son satisfechos por un elemento-hijo: ▪ Utilizar la tarea directamente.	**Verde**
AAC9	Planear los Casos y Procedimientos de Prueba para las pruebas de integración: ▪ Utilizar la tarea directamente.	**Verde**
AAC10	Revisar y decidir si se satisfacen los requerimientos del sistema: ▪ Utilizar la tarea directamente.	**Verde**

Cuadro 4.4: Tareas de los PPO factibles y no factibles a ser introducidas en MGDASMA.

Dada las características de las tareas de los PPO, se puede apreciar que éstos tienen una correspondencia en las etapas de la generalización del ciclo de vida de las metodologías ágiles. La Figura 4.3, muestra la correspondencia uno a uno de los PPO y etapas de la generalización del ciclo de vida.

4.3. Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles (MGDASMA)

Para generar MGDASMA se deben seguir proceso para obtener el modelo genérico sintetizado en la Figura 3.3.

En la actividad A1 de la Figura 3.3, se verifica y valida la lista de actividades y etapas, de la generalización del ciclo de vida de las metodologías ágiles, que promueven el desarrollo de la arquitectura de software y que se muestran en el Cuadro 4.5.

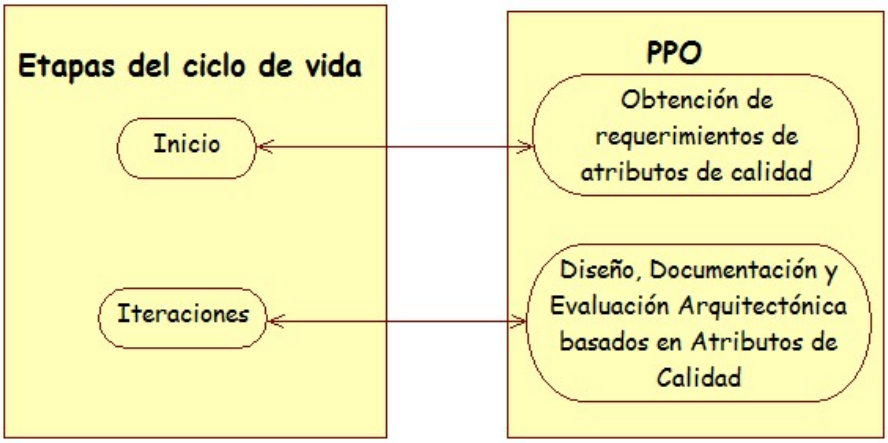

Figura 4.3: Correspondencia de los PPO con las etapas de la generalización del ciclo de vida de las metodologías ágiles.

Lista de etapas y actividades de la generalización	
Etapas	**Actividades**
▪ Inicio	▪ Visión ▪ Requerimientos ▪ Arquitectura Inicial
▪ Iteraciones	▪ Arquitectura ▪ Selección de requerimientos

Cuadro 4.5: Etapas y actividades para el desarrollo de la arquitectura de software en la generalización del ciclo de vida de las metodologías ágiles.

En la actividad A2 de la Figura 3.3, se utilizan como entrada los siguientes resultados:

- Tareas de los PPO factibles y no factibles a ser introducidas en MGDASMA (ver Cuadro 4.4).
- Correspondencia de los PPO con las etapas de la generalización del ciclo de vida de las metodologías ágiles (ver Figura 4.3).
- Etapas y actividades para el desarrollo de la arquitectura de software en la generalización del ciclo de vida de las metodologías ágiles (ver Cuadro 4.5).

Antes de definir los nombres para las actividades que tendrán las tareas de los PPO se debe analizar los siguientes puntos:

1. Definir dos actividades con sus respectivas tareas de los PPO.

2. Introducir tareas de los PPO a las actividades de la generalización del ciclo de vida y definir actividades para las tareas restantes no consideradas.

3. Reemplazar actividades de la generalización del ciclo de vida por otras.

El punto número uno, es el más sencillo de realizar porque, simplemente, se toman las tareas de los PPO analizados con el *Manifiesto Ágil* y el conjunto de tareas de un PPO se introduce a una actividad nueva en la generalización del ciclo de vida. De igual manera se realiza el mismo procedimiento para el otro PPO. Esto puede generar un inconveniente cuando se realicen las otras actividades de la generalización del ciclo de vida. Supongamos que en una actividad se obtiene un resultado que también se puede obtener en la actividad nueva. Esto significa que se estaría realizando dos veces el mismo trabajo. Por este motivo, se tuvo cuidado en no considerar este punto.

En el punto número dos, se analiza qué tareas de los PPO se pueden realizar en las actividades de la generalización del ciclo de vida para posteriormente introducirlas a dichas actividades. También se analiza, que para las tareas que fueron no fueron consideradas lo recomendable es definir una actividad representativa e introducir las tareas restantes. Cabe aclarar que se tendría que especificar el orden de las actividades con las tareas restantes. El refinamiento de las actividades descritas anteriormente permitió evitar la duplicidad de algunos resultados y, también, permitió un intercalación natural de las tareas de los PPO con las actividades de la generalización del ciclo de vida. Por ejemplo, se puede tener un actividad de la generalización del ciclo de vida que se complemente con tareas que provienen las tareas de los PPO.

En el punto tres, se analiza si las actividades de la generalización del ciclo de vida debe ser reemplazada por otras actividades nuevas. Por ejemplo, puede ocurrir que actividades de la generalización del ciclo de vida sean implícitas. Si se reemplaza esta actividad con una nueva, que sea explícita, fortalecería a la generalización del ciclo de vida.

En el Cuadro 4.6, se muestran las tareas de los PPO que se introducen a las actividades de la generalización del ciclo de vida. En el Cuadro 4.7, se muestran las actividades nuevas de la generalización del ciclo de vida en dónde se incluyen las tareas de los PPO.

Actividades	**Tareas**
Acuerdos con el cliente	▪ RAC1 ▪ RAC3
Arquitectura inicial	▪ RAC2

Cuadro 4.6: Actividades con tareas de los PPO en la etapa de Inicio de la generalización del ciclo de vida.

Actividades nuevas	**Tareas**
Iniciar taller	▪ RAC4 ▪ RAC5 ▪ RAC6 ▪ RAC7
Obtener escenarios de atributos de calidad	▪ RAC8 ▪ RAC9 ▪ RAC10 ▪ RAC11 ▪ RAC12

Cuadro 4.7: Actividades nuevas con tareas de los PPO en la etapa Inicio de la generalización del ciclo de vida.

La lista de actividades con tareas adaptadas de los PPO en la etapa Inicio se muestran en el Cuadro 4.8. Cabe aclarar que, para la para la etapa Inicio de la generalización del ciclo de vida se debe seguir cierto orden como se plantea a continuación:

1. Entre una actividad y otra pueden realizarse otras actividades de la generalización del ciclo de vida, pero el orden las actividades con tareas de los PPO deben ser realizadas como se muestra en el Cuadro 4.8.

2. El resultado obtenido de la actividad Visión, de la generalización del ciclo de vida (ver Cuadro 4.1), facilita la realización de las tareas: RAC2, RAC5 y RAC6, que están distribuidas en las actividades: Arquitectura inicial y Iniciar taller.

Actividades en la etapa de Inicio
Acuerdos con el cliente
Visión
Requerimientos
Arquitectura inicial
Iniciar taller
Obtener escenarios de atributos de calidad

Cuadro 4.8: Actividades con tareas adaptadas de los PPO de la etapa de Inicio.

El Cuadro 4.9 muestra la actividad nueva Diseño arquitectónico con tareas del PPO Diseño, Documentación y Evaluación Arquitectónica basados en Atributos de Calidad en la etapa de Iteraciones. Cabe aclarar que las tareas de esta actividad pueden realizarse de manera conjunta en una misma actividad a diferencia de las tareas del PPO de Obtención de Requerimientos de Atributos de Calidad.

Actividad nueva	**Tareas**
Diseño arquitectónico	▪ AAC1 ▪ AAC2 ▪ AAC3 ▪ AAC4 ▪ AAC5 ▪ AAC6 ▪ AAC7 ▪ AAC8 ▪ AAC9 ▪ AAC10

Cuadro 4.9: Actividad nueva con tareas de los PPO en la etapa de Iteraciones de la generalización del ciclo de vida.

La actividad Diseño arquitectónico no debe realizarse por partes. Al introducir las tareas de los PPO en esta actividad nueva, lo más recomendable es que se realice la actividad en un solo momento. Esto obliga a la actividad a tener más prioridad sobre otras actividades de la generalización del ciclo de vida. Es decir, para las actividades Planificación y Construcción de las iteraciones se necesita la salida de la actividad Diseño arquitectónico, lo cual permitirá poder planificar y construir de la mejor manera posible las iteraciones. No se debe olvidar que los requerimientos del sistema que se obtienen en la actividad Requerimientos en la etapa Inicio de la generalización del ciclo de vida, servirán como entrada a la actividad Diseño arquitectónico. Es por eso que se debe tener cuidado en elegir aquellos requerimientos que puedan ser directrices arquitectónicas en potencia para que finalmente la primera iteración sea donde se empiece a construir la arquitectura.

La lista de actividades con tareas adaptadas de los PPO en la etapa de Iteraciones se muestra en el Cuadro 4.10. Cabe aclarar que entre una actividad y otra pueden realizarse otras actividades de la generalización del ciclo de vida pero el orden debe ser como se muestra en el Cuadro 4.9.

Actividades en la etapa de Iteraciones
Selección de requerimientos
Diseño arquitectónico
Planificación
Construcción de las iteraciones

Cuadro 4.10: Actividades con tareas adaptadas de los PPO de la etapa de Iteraciones.

Finalmente, se verifican y validan, las actividades con las tareas adaptadas de los PPO en las etapas de Inicio e Iteraciones, de la generalización del ciclo de vida de las metodologías ágiles, y que se muestran en los Cuadros 4.8 y 4.10. Cabe aclarar que las tareas de los PPO, anteriormente, fueron analizadas con el *Manifiesto Ágil* .

En el Cuadro 4.11, se presenta la Generalización del Ciclo de Vida con Actividades para el Desarrollo de la Arquitectura de Software. Las actividades nuevas con tareas de los PPO se encuentran en negritas y las actividad originales en la generalización del ciclo de vida a las que se les introdujeron tareas de los PPO tienen la marca †. La etapa Transición no tiene actividades para el desarrollo de la arquitectura de software y se omite en el Cuadro.

Inicio	**Iteraciones**
▪ Acuerdos con el cliente † ▪ Visión ▪ Requerimientos ▪ Arquitectura inicial † ▪ **Iniciar taller** ▪ **Obtener escenarios de atributos de calidad**	▪ Selección de requerimientos ▪ **Diseño arquitectónico** ▪ Planificación ▪ Construcción de las iteraciones

Cuadro 4.11: Generalización del ciclo de vida con actividades para el desarrollo de la arquitectura de software.

Para que MGDASMA no se contraponga al *Manifiesto Ágil* se debe realizar la actividad A3 de la Figura 3.3. Por lo tanto, se verifican y validan las actividades nuevas que se definieron y, además, las actividades a las que se les introdujeron tareas. Anteriormente, se analizaron las tareas con el *Manifiesto Ágil* de manera individual para que la verificación de las actividades de manera conjunta sea adecuada.

En el Cuadro 4.12, se presenta la lista de verificación de las actividades nuevas y las actividades a las que se les introdujeron tareas. Se debe mencionar que, para poder realizar esta lista de verificación, las preguntas que se plantean en el Cuadro 4.12 son para saber si las actividades están alineadas a los valores y principios del *Manifiesto Ágil*. Cabe aclarar que las actividades del Cuadro 4.12, no cubren en su totalidad a dicho *Manifiesto*. Esto se debe a que una actividad por si sola no puede satisfacer a todos los valores y principios. Sin embargo,

un conjunto de actividades tienen más alcance para cubrir a los valores y principios. Cabe aclarar que los valores y principios que falten por ser cubiertos estarán siendo afectados por las actividades restantes de la generalización del ciclo de vida.

Actividades / **Manifiesto**	**Acuerdos con el cliente**	**Arquitectura inicial**	**Iniciar taller**	**Obtener escenarios de atributos de calidad**	**Diseño arquitectónico**
¿Se promueve la interacción del equipo?			✓	✓	✓
¿Se promueve la colaboración con el cliente?	✓	✓	✓	✓	
¿Se promueve la comunicación cara a cara?	✓	✓	✓	✓	✓
¿Se promueve trabajo en equipo?		✓	✓	✓	✓
¿Se promueve flexibilidad para documentar?			✓	✓	✓
¿Se promueve en buen diseño?	✓	✓	✓	✓	✓

Cuadro 4.12: Lista de verificación de las actividades.

Finalmente, MGDASMA se muestra en el Cuadro 4.13. Las actividades nuevas están en negritas y las actividad existentes en la generalización del ciclo de vida a las que se les introdujeron tareas de los PPO tienen la marca †. Cabe aclara que la actividad Arquitectura de la etapa Iteraciones (ver Cuadro 4.2), ha sido cambiada por la actividad Diseño arquitectónico.

Inicio	Iteraciones	Transición
▪ Misión ▪ Acuerdos con el cliente † ▪ Alcance ▪ Visión ▪ Planificación ▪ Riesgos ▪ Roles ▪ Objetivos ▪ Requerimientos ▪ Arquitectura inicial † ▪ Priorización ▪ Estimaciones ▪ **Iniciar taller** ▪ **Obtener escenarios de atributos de calidad**	▪ Participación constante con los involucrados ▪ Reuniones ▪ Selección de requerimientos ▪ **Diseño arquitectónico** ▪ Planificación ▪ Construir las iteraciones ▪ Retroalimentación ▪ Codificación ▪ Cambios ▪ Pruebas	▪ Generar la documentación del sistema ▪ Realizar la puesta en operación del sistema ▪ Reuniones con los involucrados ▪ Documentar manuales del sistema ▪ Identificar defectos y mejoras para el sistema.

Cuadro 4.13: Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles (MGDASMA).

4.4. Instancia de MGDASMA a una metodología ágil

En esta sección se presenta la instancia de MGDASMA a una metodología ágil específica. Para realizar dicha instancia se utilizará la guía de instanciación del Cuadro 3.6. Para empezar la instancia se debe de tener en cuenta el criterio de entrada que especifica:

- Documentación de la metodologías ágil específica
- MGDASMA (Cuadro 4.13)

Para la documentación de la metodología ágil específica, en la sección 2.1.7 se presentó la Figura 2.2, dónde se muestra que la metodología ágil XP es la más adoptada para desarrollos ágiles, está se eligió para realizar la instancia. Esta metodología consiste, a grandes rasgos, en utilizar buenas prácticas de programación sin meterse mucho a cuestiones administrativas. Cabe aclarar que las prácticas de programación equivalen a las actividades propuestas de MGDASMA. A continuación se irán desarrollando los pasos de la guía de instanciación (ver Cuadro 3.6):

1. **Conocer MGDASMA:**
   - Revisar la sección 4.3.
2. **Conocer la metodología ágil:**

- Se eligió la metodología ágil XP.
- El ciclo de vida, valores y/o principios se pueden revisar en la sección 2.1.7.2.
- Las prácticas y roles de XP se pueden revisar en la sección A.1.

3. **Identificar fases y actividades:**

Fases y actividades de XP	**Etapas y actividades de la generalización**
Exploración y Planificación de la Entrega: ▪ Cliente *in situ* ▪ Metáfora ▪ Juego de Planificación	Inicio ▪ Acuerdos con el cliente ▪ Arquitectura inicial ▪ Iniciar taller ▪ Obtener escenarios de atributos de calidad
Iteraciones ▪ Diseño simple	Iteraciones ▪ Diseño arquitectónico

Cuadro 4.14: Fases, etapas y actividades de XP y MGDASMA.

4. **Introducir actividades, sus equivalencias y sus relaciones:**
La equivalencia de las prácticas con las actividades de MGDASMA son:

   - Cliente *in situ* y Juego de Planificación ∼ Acuerdos con el cliente
   - Metáfora ∼ Arquitectura Inicial
   - Diseño simple ∼ Diseño arquitectónico

   La Figura 4.4, muestra las relaciones entre las actividades.

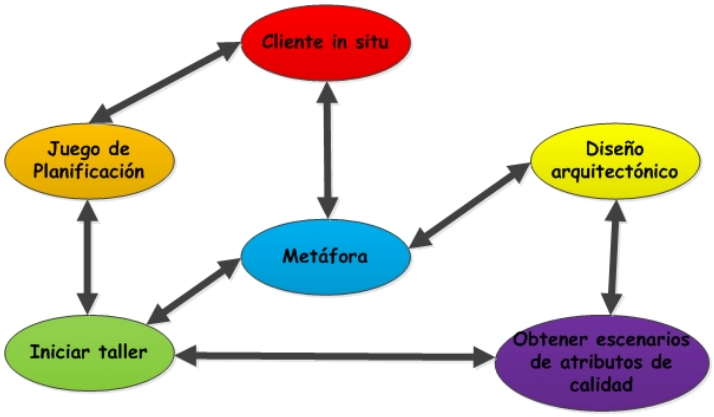

Figura 4.4: Relación de las actividades.

Finalmente, el Cuadro 4.15, muestra las actividades para el desarrollo de la arquitectura de software, en su respectiva fase, introducidas a XP. Cabe aclarar que las actividades con la marca †tienen tareas de los PPO añadidas a las además tareas que se realicen en esa actividad.

Exploración y Planificación de la Entrega
▪ Cliente *in situ* † ▪ Juego de Planificación † ▪ Metáfora † ▪ Iniciar taller ▪ Obtener escenarios de atributos de calidad
Iteraciones
▪ Diseño arquitectónico

Cuadro 4.15: Actividades para el desarrollo de la arquitectura de software introducidas en XP.

Al finalizar el paso cuatro de la guía de instanciación se obtiene una metodología ágil robustecida con actividades para el desarrollo de la arquitectura de software.

4.5. Administración del proyecto

La administración del proyecto se realizó a través de una serie de actividades enfocadas hacia el cumplimiento de los objetivos de la tesis. Para ello, se decidió utilizar la herramienta *OpenProj* para definir tareas, actividades y entregables que nos llevaron a obtener en tiempo los resultados de este trabajo. Una ventaja de emplear la administración del proyecto se refleja en la temprana y oportuna elaboración de documentos entregables de tal manera que al momento de desarrollar la tesis el tiempo de redacción se minimizó en gran medida.

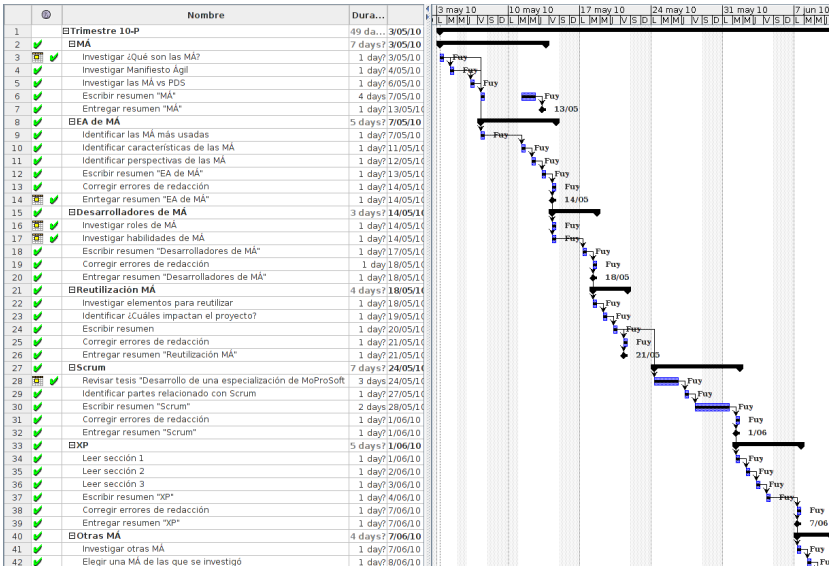

Figura 4.5: Actividades y entregables en la revisión de metodologías ágiles.

En la Figura 4.5, se muestra un ejemplo de las actividades y entregables realizados en la revisión de metodologías ágiles. Los resultados obtenidos se pueden ver reflejados en la sección 2.1. Las actividades que se realizaron fueron de gran utilidad para conocer más a fondo los aspectos más importantes para el desarrollo de la generalización (ver sección 4.1).

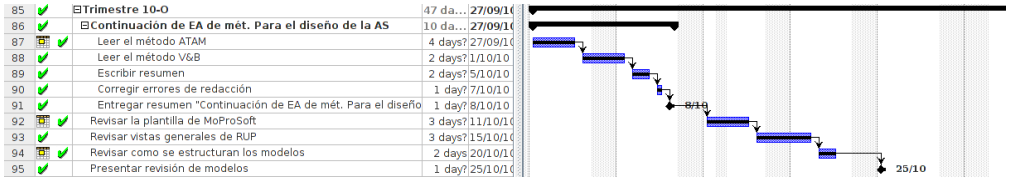

Figura 4.6: Actividades y entregables en la revisión de arquitectura de software y procesos.

En la Figura 4.6, se muestra otro ejemplo de las actividades y entregables realizados en la revisión de métodos para el desarrollo de la arquitectura de software y procesos. Los resultados obtenidos se pueden ver reflejados en la sección 2.2 y 2.4. Las actividades que se realizaron fueron de gran utilidad para conocer diferentes métodos para el desarrollo de la arquitectura de software. También, para conocer como estructurar el modelo genérico de este trabajo de tesis.

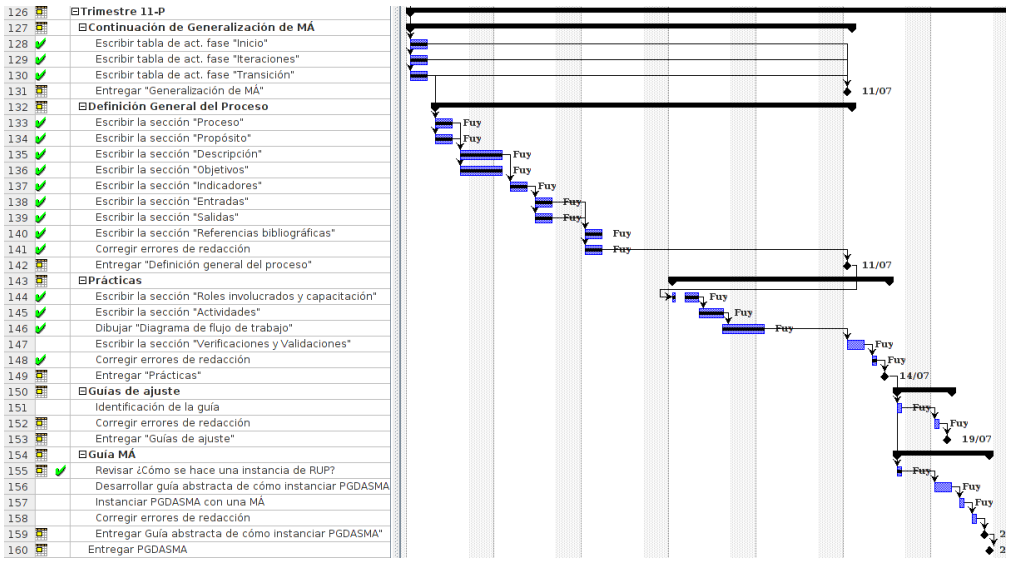

Figura 4.7: Actividades y entregables para la generalización y definición de MGDASM.

En la Figura 4.7, se muestra un último ejemplo de las actividades y entregables realizados en la generalización del ciclo de vida de las metodologías ágiles y la definición de MGDASMA. Los resultados obtenidos se pueden ver reflejados en la sección 4.1 y 4.3. Estas últimas actividades fueron las más importantes en esta administración del proyecto.

Esta administración de proyecto es interesante porque se puede apreciar los recursos, la duración y la secuencia de cada actividad. Esto permitirá que para la continuidad del trabajo futuro la planificación que se realice podrá ser más precisa.

En este capítulo se presentaron los resultados:

- Generalización del ciclo de vida de las metodologías ágiles

- PPO analizados con el *Manifiesto Ágil*
- MGDASMA
- Instancia de MGDASMA a una metodología ágil

Los resultados muestran evidencias de cómo introducir actividades para el desarrollo de la arquitectura de software en metodologías ágiles en forma general con cinco de las metodologías: *Scrum*, XP, AUP, ASD y Kanban. Para que esto fuera posible se tuvo que utilizar los PPO, los cuales ofrecen una serie de tareas para el desarrollo de la arquitectura de software. Estos PPO fueron analizados con el *Manifiesto Ágil* para que al momento de introducir las tareas a las actividades de la generalización del ciclo de vida no se contrapusieran con dicho *Manifiesto*. Finalmente, a partir de la generalización del ciclo de vida, los PPO y utilizando la plantilla de MoProSoft se desarrolla MGDASMA y se prueba realizando el modelo realizando una instancia hacia una metodología ágil específica.

Capítulo 5

Discusión de resultados

En este capítulo se presenta la discusión de los resultados más relevantes de este proyecto:

- Generalización. Los resultados de generalización se discuten en el sentido de la correlación y clasificación de las fases de las metodologías ágiles para generalizar el ciclo de vida. También, son abordados los resultados del ciclo de vida correlacionado de las metodologías ágiles, y sus etapas: Inicio, Iteraciones y Transición, y sus respectivas actividades.
- Paquetes de Puesta en Operación (PPO). Se discute el uso de los PPO y los resultados obtenidos al utilizarlos.
- MGDASMA y sus instancias. En esta sección se discute MGDASMA.

5.1. Generalización

La generalización del ciclo de vida de las metodologías ágiles fue una estrategia que se propuso para esta tesis (ver sección 3.1). Este trabajo consistió en revisar los ciclos de vida y fases de varias metodologías. En esta revisión se lograron identificar actividades que se realizan en cada fase del ciclo de vida de la metodología ágil: Scrum, XP, Kanban, ASD y AUP. Se observó que las fases tenía actividades similares y/o equivalentes con las de otra metodología. Esto permitió establecer una correlación entre una(s) fase(s) de una metodología con la(s) fase(s) de otra. A través de esta acción se descubrió que la(s) fase(s) correlacionadas tienen actividades similares y/o equivalentes. Para realizar la correlación se presentaron dos alternativas:

1. Correlación de la(s) fase(s) de una metodología con la(s) fase(s) de otra metodología.
2. Correlación de la(s) fase(s) de una metodología con todas las fases de las demás metodologías.

La alternativa número uno permitió dividir en partes más concretas el problema. De esta forma la correlación fue más precisa al momento de identificar las actividades de cada fase y su respectiva correlación. La alternativa número dos implicaba más pasos para establecer la correlación. Primero se elegía una metodología y su(s) fase(s). Después, se tenía que establecer una correlación con la(s) fase(s) de otra metodología. Nuevamente se retomaba la metodología elegida y se establecía otra correlación con la(s) fase(s) de otra metodología y

así sucesivamente. Esta alternativa no era clara porque la(s) fase(s) de una metodología podía tener correlación con una o más fases de las metodologías restantes. Por lo tanto, para obtener claridad en la correlación se optó por la alternativa número uno.

La clasificación de las fases de las metodologías ágiles permite agrupar fases correlacionadas de tal forma que los grupos que se conformen contienen actividades con enfoques similares y/o equivalentes. La clasificación que se estableció no fue inmediata. Se podía clasificar de "n" formas, pero se eligió clasificar tomando en cuenta los siguientes aspectos:

- Etapas iniciales con actividades como: obtención de requerimientos, planificación, misión, alcance, etc.
- Etapas intermedias con actividades como: diseño, iteraciones, implementación, etc.
- Etapas finales con actividades como: entrega de producto, puesta en operación, etc.

En base a lo anterior se establecieron tres etapas: Inicio, Iteraciones y Transición. Esta clasificación de etapas permitió evidenciar dos etapas muy importantes para la introducción de actividades para el desarrollo de la arquitectura de software: Inicio e Iteraciones. Dada las características de las actividades en las etapas de Inicio e Iteraciones, se concluyó que estas etapas fueron claves parar introducir actividades para el desarrollo de la arquitectura de software. Por lo tanto, dada la clasificación a través de la correlación, se estableció la generalización del ciclo de vida de las metodologías ágiles consideradas, como se muestra en la Figura 4.2. Cabe aclarar que esta generalización no es única. Esta generalización se da en consecuencia por la forma en la que se realizó la clasificación tomando en cuenta los aspectos mencionados anteriormente. De haber clasificado de otra manera la generalización tendría otra forma por la diversidad de fases y actividades de cada metodología ágil. Lo que se resalta de esta generalización es que es clara y simple parar introducir actividades para el desarrollo de la arquitectura de software en las etapas respectivas.

Las actividades en las etapas de la generalización son puntos clave para poder determinar en dónde son introducidas las nuevas actividades para el desarrollo de la arquitectura de software, pero a su vez muestran actividades que no son explícitas en esas metodologías. Por ejemplo, en el Cuadro 4.2, se puede observar que la actividad "Arquitectura" es una actividad no explícita, en el sentido de que no se detalla dentro de la metodología, en *Scrum*, XP, ASD y Kanban. Se puede advertir que cuando se emplee una de esas metodologías ágiles la gente menos experimentada puede tener dificultades cuando realice esa actividad.

Como sabemos existe una gran variedad de metodologías ágiles y la generalización del ciclo de vida de ellas en este trabajo de tesis, no representa, en su totalidad a todas las metodologías ágiles. Como se plantea en este trabajo, este ciclo de vida generalizado, parte de cinco metodologías ágiles, por lo que las etapas y actividades definidas van en sintonía con las metodologías empleadas.

5.2. Paquetes de Puesta en Operación (PPO)

La introducción de actividades para el desarrollo de la arquitectura de software en metodologías ágiles no es una tarea trivial. Por un lado, en el inicio de este trabajo de tesis, se especificó realizar una revisión de métodos conocidos y que se utilizan en el desarrollo de la arquitectura de software. Al revisarlos se identificó que muchos no eran aplicables directamente a las metodología ágiles. La razón de peso es el *Manifiesto Ágil* ya que si se desean utilizar métodos para el desarrollo arquitectónico, las actividades, pasos y/o tareas no deben

contraponerse con lo expresado en el *Manifiesto Ágil*. De entrada, varios de estos métodos arquitectónicos exigen una documentación detallada, a diferencia de las metodologías ágiles en dónde se valora más al software funcionando sobre una documentación extensiva.

Por otro lado, están los PPO y estos se enfocan hacia la obtención de requerimientos, el diseño, la documentación y evaluación arquitectónica basados en atributos de calidad. Sin embargo, las tareas de los PPO no están alineadas al *Manifiesto Ágil*, pero se identificó que sus tareas tienen congruencia con varios de los valores y principios del *Manifiesto Ágil*. Entre los aspectos congruentes están:

- Interacción constante con el cliente.
- Colaboración y reuniones con el cliente y equipo de desarrollo.
- Motivación entre los miembros del equipo de desarrollo.

El punto desfavorable del *Manifiesto Ágil* hacia los PPO se centra en la documentación. Por lo general, en las tareas, de los PPO, se menciona documentar en diferentes formas e incluso se promueve hacer uso de plantillas que se proporcionan. Esto podría ser un inconveniente, pero en esas tareas se especifica que es opcional generar la documentación detalla. Asumimos que esto permite ser flexible al momento de utilizar los PPO. De tal forma, que el miembro con más experiencia pueda buscar alternativas más adecuadas para realizar la tarea y no contraponerse al *Manifiesto Ágil*.

Los resultados al analizar los PPO, tomando como referencia el *Manifiesto Ágil* (Cuadro 4.4), permitieron minimizan la contraposición de las tareas al manifiesto. Esto tiene un cierto grado de incertidumbre por lo siguiente: el *Manifiesto Ágil* contiene valores y principios a los que se deben alinear las metodologías ágiles. No es sencillo establecer cómo se deben adaptar las tareas con todo lo que se describe en el *Manifiesto Ágil*, en ese sentido la ejecución de una tarea no puede representar la oposición o no oposición al *Manifiesto Ágil*. Sin embargo, un conjunto de tareas si podría hacerlo. Por este motivo, los PPO analizados como se describe en este trabajo de tesis garantizan en forma teórica que las tareas en su conjunto no se contraponen con el *Manifiesto*. Sin embargo, en la práctica no se ha realizado ningún caso de estudio formal para comprobarlo.

5.3. MGDASMA y sus instancias

Como se mencionó anteriormente la generalización del ciclo de vida de las metodologías ágiles se elabora a partir de cinco de ellas: XP, *Scrum*, Kanban, ASD y AUP. Esto significa que MGDASMA, en teoría, se puede instanciar fácilmente a cualquiera de ellas. Sin embargo, no se puede garantizar que la instancia sea adecuada en alguna otra metodología ágil que no sean las anteriores hasta no correlacionar esta metodología. Suponemos que la correlación no debe ser un problema si se emplea la metodología de la sección 3.1.5.

La plantilla de MoProSoft (Patrón de procesos) fue de mucha utilidad para simplificar y estructurar la descripción del proceso para construir MGDASMA. La ventaja de usar los elementos rescatados de la plantilla se debe a que establece un buen nivel de formalidad para describir MGDASMA. En estos elementos se destacan los que tuvieron mayor impacto: objetivos, indicadores, entradas, salidas, actividades, verificaciones, validaciones, flujo de trabajo y mediciones. Tan solo el elemento flujo de trabajo fue de mucha utilidad para poder realizar las actividades para identificar, adaptar y verificar las partes que componen a MGDASMA.

De no emplear la plantilla, la construcción de MGDASMA hubiera sido más sofisticada ya que se habría tenido que analizar, establecer y definir varios criterios para conformarlo.

La guía de instanciación (sección 3.1.7) es un punto clave para llevar a cabo el MGDASMA a una metodología ágil. Cabe aclarar que esta instancia tiene que ser hacia una de las metodologías ágiles utilizadas en la generalización del ciclo de vida. Un modelo, a diferencia de un proceso, debe ser instanciado para utilizarlo. Caso contrario con un proceso, dónde se establece un conjunto de actividades que se ejecutan para lograr un propósito. En este sentido, en este trabajo de tesis se desarrollo un proceso para establecer un modelo. Para que la guía de instanciación sea eficiente, es importante que cuando se realice la instancia ésta esté a cargo de personas que tengan un amplio conocimiento sobre la metodología ágil elegida, en especial en la práctica. De esta manera se podrá establecer de forma específica las relaciones de las actividades de MGDASMA cuando ya estén en la metodología. De no contar con la experiencia necesaria la instancia puede no ser la más adecuada. Este sería un punto débil de la instancia.

Los trabajo revisados anteriormente (sección 2.3), muestran como aplican métodos arquitectónicos en metodologías ágiles por lo que el enfoque que dan los autores siempre es dirigido a metodologías específicas. Si bien el resultado es favorable para esa metodología específica, para otra no lo es porque no es claro como se pueda replicar. Hasta el momento no se encontró en la literatura algún proceso, modelo o metodología capaz de abarcar aspectos de arquitectura de software hacia metodologías ágiles en general. Por este motivo, MGDASMA es un trabajo con una visión mucho más amplia porque trata de no dejar a un lado a las demás metodologías ágiles, ofreciendo una guía general y formal para instanciar el modelo a otras metodologías ágiles. Por lo tanto, el aporte que genera este trabajo tiene que ser analizado y discutido con la comunidad de desarrolles ágiles.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

6.1.1. Aportación de este trabajo

El propósito de este trabajo de tesis fue desarrollar un modelo que permita establecer las directrices necesarias para incorporar, explícitamente y en forma genérica, el desarrollo de la arquitectura de software en metodologías ágiles. Los aportes de este trabajo son:

- Proporcionar una metodología formal (ver sección 3.1) para definir pasos, tareas, actividades y productos a lograr, de tal manera que al reproducir dicha metodología se puedan obtener los mismos resultados.

- Proporcionar una generalización del ciclo de vida de cinco metodologías ágiles (ver sección 4.1): Scrum, XP, AUP, ASD y Kanban, que se compone de tres etapas: Inicio, Iteraciones y Transición. Esta generalización del ciclo de vida es la base del modelo para la introducción de actividades para el desarrollo de la arquitectura de software en metodologías ágiles. Sin embargo, dada la forma en cómo se construyó la generalización del ciclo de vida de las metodologías ágiles no solo se puede introducir actividades arquitectónicas. En el trabajo titulado:"Trying to Link Traceability Elements in a General Agile Model Life Cycle, 12th International Conference on Agile Software Development" (ver sección B.1.3), se utiliza la generalización del ciclo de vida para introducir el elemento trazabilidad en las metodologías ágiles. Por lo que también se podría introducir otros elementos para fortalecer a las metodologías ágiles.

- Presentar el proceso para la elaboración de un Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles (ver sección 4.3). Dicho proceso puede ser replicable con cambios mínimos con otro enfoque.

- Proporcionar una guía para instanciar MGDASMA a una metodología ágil específica. Esta guía también es aplicable para las metodologías ágiles con las que estableció la generalización.

6.1.2. Logro de los objetivos

Los objetivos de este trabajo de tesis se muestran en la sección 2.6. A continuación se expone el objetivo y se describe en qué secciones se lograron dichos objetivos:

1. Entender las bases comunes de los métodos ágiles para conocer todos sus elementos.
   En la sección 2.1, se hace una revisión de varias metodologías ágiles. Se presenta varios aspectos importantes de las metodologías como el *Manifiesto Ágil*. Dicho Manifiesto establece los valores y principios de las metodologías ágiles. Se identifica que las metodologías ágiles utilizadas para este trabajo comparten elementos y características en común que serán útiles para generalizar el ciclo de vida de ellas.

2. Revisar métodos de desarrollo de la arquitectura de software para encontrar puntos clave y su posible adaptación a metodologías ágiles.
   En la sección 2.2, se hace un revisión de métodos arquitectónicos. En esta revisión se exponen los elementos de dichos métodos y se analiza cuáles pueden ser candidatos a ser introducidos en las metodologías ágiles.

3. Analizar diferentes procesos para poder rescatar elementos valiosos que permitan documentar los resultados a obtener en este proyecto.
   En la sección 2.4, se hace una revisión de varios procesos. Esto con el fin de entender cómo definir y estructurar el modelo genérico de este trabajo.

4. Desarrollar y probar con una instancia, el modelo genérico a elaborar en este proyecto, para incorporar actividades para el desarrollo de la arquitectura de software en métodos ágiles.
   En la sección 4.3, se presenta el Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles. Éste Modelo realiza el objetivo general de la tesis. Para poder probar el modelo se necesita instanciarlo hacia una metodología ágil. La metodología elegida es XP y la instancia se puede revisar en la sección 4.4.

6.2. Trabajo futuro

Como trabajo a futuro de esta tesis tenemos que:

1. La generalización expuesta en el capítulo 4.1 es un resultado muy importante que puede utilizarse con diversos enfoques. Emplear la generalización del ciclo de vida de las metodologías ágiles puede ser muy útil para introducir más elementos a las metodologías ágiles. Anteriormente, la generalización se ocupó para introducir el elemento trazabilidad. Como trabajo futuro se debe hacer una revisión de elementos que se puedan integrar para fortalecer, aún más, a las metodologías ágiles.

2. Añadir más metodologías ágiles a la generalización y establecer sus respectivas correlaciones.

3. Afinar o ajustar MGDASPA. La primera versión de un modelo y/o proceso no es el definitivo. Se debe probar MGDASMA para poder afinarlo y/o ajustarlo con respecto a las observaciones y/o retroalimentación después de su uso y así enriquecer y fortalecer a MGDASMA.

4. Realizar la instancia de MGDASMA con las metodologías ágiles de la generalización: *Scrum*, Kanban, ASD y AUP .

5. Utilizar la guía de instanciación para probar MGDASMA en un proyecto real.

6. Publicar a la comunidad de ingeniería de software MGDASMA.

7. Realizar una publicación en revista de MGDASMA ya probado con diferentes casos de estudio en proyectos reales.

Apéndices

Apéndice A

A.1. Prácticas y roles de XP

A.1.1. Prácticas

Prácticas de programación de XP([18] y [43]):

- **Juego de Planificación (*Planning Game*):** Es un espacio frecuente de comunicación entre el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración. Esta práctica se puede ilustrar como un juego, donde existen dos tipos de jugadores: Cliente y Programador. El cliente establece la prioridad de cada historia de usuario, de acuerdo con el valor que aporta para el negocio. Los programadores estiman el esfuerzo asociado a cada historia de usuario. Se ordenan las historias de usuario según prioridad y esfuerzo, y se define el contenido de la entrega y/o iteración, apostando por enfrentar lo de más valor y riesgo cuanto antes. Este juego se realiza durante la planificación de la entrega, en la planificación de cada iteración y cuando sea necesario reconducir el proyecto.

- **Liberaciones Cortas (*Small Release*):** La idea es producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad pretendida para el sistema, pero si que constituyan un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

- **Metáfora del Sistema (*System Metaphor*): En XP no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume evolutiva y los posibles inconvenientes que se generarían por no contar con ella explícitamente en el comienzo del proyecto se solventan con la existencia de una metáfora.** El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que o actúen como vocabulario para hablar sobre el dominio del problema , ayudando a la nomenclatura de clases y métodos del sistema).

- **Diseño Simple (*Simple Desing*):** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. La complejidad innecesaria y el código extra debe ser removido inmediatamente. Kent Beck dice "*que en cualquier momento el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención*

de implementación de los programadores y tiene el menor número posible de clases y métodos".

- **Refactorización (*Refactoring*):** Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo.

- **Pruebas (*Testing*):** La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial.

- **Programación en Parejas (*Pair Programming*):** Toda producción de código se realiza en parejas.

- **Propiedad Colectiva (*Collective ownership*):** Cualquier programador puede cambiar cualquier parte del código en cualquier momento. Esta práctica motiva a todos a contribuir con nuevas ideas en todos los segmentos del sistema, evitando a la vez que algún programador sea imprescindible para realizar cambios en alguna porción de código.

- **Integración Continua (*Continuos Integration*):** Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.

- **40 horas por semana (*40-hour week*):** Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.

- **Cliente *in situ* (*On-Site Customer*):** El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. **La comunicación oral es más efectiva que la escrita.**

- **Estándares de Codificación (*Coding Standards*):** *XP* enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

A.1.2. Roles

Los roles de XP([18] y [43]) son:

- **Encargado de seguimiento (*Tracker*):** El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del

progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes. Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.

- **Cliente (*Customer*):** El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

- **Programador (*Programmer*):** El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

- **Encargado de Pruebas (*Tester*):** El encargado de pruebas ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

- **Entrenador (*Coach*):** Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

- **Consultor:** Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.

- **Mánager (*Big boss*):** Es el vínculo entre los clientes y desarrolladores, asegura que se siga el proceso y ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas.

Apéndice B

B.1. Artículos y póster con resultados de la tesis

B.1.1. Las Metodologías Ágiles y las Arquitecturas de Software

Urquiza-Yllescas J. F., Martínez-Martínez A., Ibargüengoitia-González G. E.; Las Metodologías Ágiles y las Arquitecturas de Software; Coloquio Nacional de Investigación en Ingeniería de Software y Vinculación Academia-Industria (CoNIIS) pp. 55 – 60, León, México, Septiembre 29 – Octubre 1, 2010.

Resumen. Las metodologías ágiles han ganado bastante popularidad desde hace algunos años. Si bien son una muy buena solución para proyectos a corto plazo, en especial, aquellos proyectos en dónde los requerimientos están cambiando constantemente, en proyecto a largo plazo, el aplicar metodologías ágiles no dan tan buenos resultados. El diseño de la arquitectura de software es una práctica muy importante para el desarrollo de software. Tener una buena arquitectura implica que nuestro sistema tiene atributos de calidad que nos van a dar un valor muy importante en el software. Si se definen actividades que fomenten el uso de métodos para el desarrollo de la arquitectura, en un proceso de desarrollo de software, se puede obtener muchos beneficios con respecto al producto que se desarrolla. Sin embargo, para las metodologías ágiles esas actividades no se consideran en forma importante.

En este trabajo se plantea un avance en el intento de incluir actividades de diseño de arquitecturas de software en metodologías ágiles, desde una perspectiva general que, se podría particularizar a las diferentes metodologías ágiles.

Lo que se expone en este artículo, es un propuesta de tesis de la maestría en Ciencias y Tecnologías de la Información que se imparte en la UAM Iztapalapa. Y se encuentra bajo la dirección de la M. en C. María Guadalupe Elena Ibargüengoitia González y el M. en C. Alfonso Martínez Martínez.

B.1.2. General Process for Software Architecture Development in Agile Process Models

Urquiza-Yllescas J. F., Martínez-Martínez A., Ibargüengoitia-González G.; General Process for Software Architecture Development in Agile Process Models (Resumen y Póster), Eleventh Mexican International Conference on Computer Science, Toluca, México, Marzo 23-24, 2011.

Resumen. Agile process models have gained popularity in recent years. They are a very good solution for short term projects, particularly for those where requirements are constantly changing. The design of the software architecture is a key piece of the software development, where strong architecture depicts quality and value attributes for the end user product. If

activities that promote the use of methods for software architecture development are defined in a software development process, several benefits may be achieved, regarding the product under development. However, for agile process models such activities are not considered as important.

In this contribution we present some advances in the attempt of defining a general process for software architecture development for agile process models. Those advances include the general elements that allow the necessary requirements to explicitly incorporate the software architecture in agile process models, with no contradiction to the principles and statements expressed in the agile manifesto.

The proposed methodology for the development of the work is as follows:

- Make a comparison of agile process models.
- Choose methods of software architecture.
- Choose agile process models.
- Propose and develop a generic process to establish drivers for integrating software architecture.
- Create an instance of the process for each agile process models choosed.
- Test instances of the selected agile process models.

The results have that been obtained in this investigation can be seen reflected in Figure 1, (see explanation below). Taking the manifest as a base, we did a review process of the agile models to then come up with a generalization of them. On the other hand we have the architecture of the software, which we will use to identify the methods that can be used in the design and finally locate where in the life cycle they are coupled. These results will be very useful in developing our generic model.

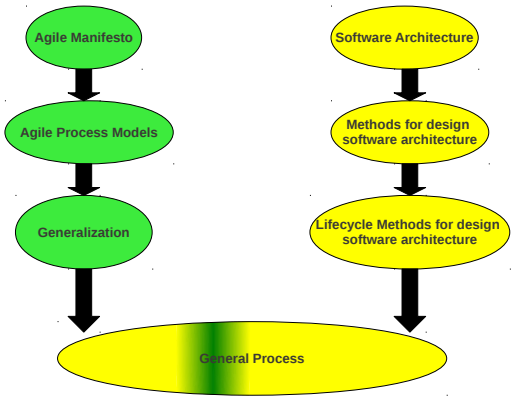

Figura B.1: Results of work

What we present here are the results of a Master's thesis in Science and Information Technology offered at the UAM Iztapalapa and is under the direction of the M. in C. Alfonso Martínez Martínez and M. in C. María Guadalupe Elena Ibargüengoitia González.

B.1.3. Trying to Link Traceability Elements in a General Agile Model Life Cycle, 12th International Conference on Agile Software Development

Urquiza-Yllescas J. F., Martínez-Martínez A., Ibargüengoitia-González G.; Trying to Link Traceability Elements in a General Agile Model Life Cycle, 12th International Conference on Agile Software Development (XP2011), Madrid, España, Mayo 10-13, 2011.

Resumen. The agile methodologies have been employed in a lot of projects in the last years, so they have won a great popularity in software development. The agile methodologies have given good solutions to short term projects, specially to projects that requirements are changing during the development. Traceability is an important part in agile software development but there are conflicts on whether the traceability is important in agile methods[39]. Traceability creates administrative task and contrasts with that stablished by the *Agile Manifesto*. This paper presents a generalization of the life cycle of agile methodologies to introduce elements of traceability and then gives an approach of how to introduce traceability in the generalization.

Bibliografía

[1] Crisp AB. Kanban. Internet: `http://www.crisp.se/kanban` Consultado: 15 de junio de 2010.

[2] Crisp AB. One day in kanban land. Internet: `http://blog.crisp.se/henrikkniberg/2009/06/26/1246053060000.html` Consultado: 15 de junio de 2010.

[3] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and architecture: Can they coexist? *IEEE Software*, 27:16–22, 2010.

[4] Scott Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[5] Scott Ambler. Agile adoption rate survey results: March 2006, Marzo 2006. Internet: `http://www.ambysoft.com/surveys/agileMarch2006.html` Consultado: Mayo 10, 2010.

[6] Scott Ambler. Agile adoption rate survey results: February 2008, February 2008. Internet: `http://www.ambysoft.com/surveys/agileFebruary2008.html` Consultado: 10 de Mayo, 2010.

[7] Scott Ambler. Modeling and documentation practices on it projects survey, July 2008. Internet: `http://www.ambysoft.com/surveys/modelingDocumentation2008.html` Consultado: Julio 7, 2010.

[8] Scott W. Ambler. The agile system development life cycle (sdlc). Internet: `http://www.ambysoft.com/essays/agileLifecycle.html` Consultado: Febrero 15, 2011.

[9] Scott W. Ambler. The agile unified process (aup). Internet: `http://www.ambysoft.com/unifiedprocess/agileUP.html` Consultado Mayo 10, 2010.

[10] Muhammad A. Babar. An exploratory study of architectural practices and challenges in using agile software development approaches. In *WICSA/ECSA*, pages 81–90. IEEE, 2009.

[11] Emanuel R. Baker, Matthew J. Fisher, and Wolfhart Goethert. Basic Principles and Concepts for Achieving Quality.

[12] Mario R. Barbacci, Robert Ellison, Anthony J. Lattanze, Judith A. Stafford, Charles B. Weinstock, and William G. Wood. Quality attribute workshops (qaws), third edition. 2003.

[13] Len Bass, Paul Clements, and Rick Kazman. *Chapter 1 The architecture Business Cycle Software Architecture in Practice, Second Edition, pp3.* Addison-Wesley.

[14] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional, April 2003.

[15] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, first edition, 1999.

[16] Kent Beck, James Grenning, and Robert C. Martin. Manifesto for agile software development. Internet: `http://agilemanifesto.org/` Consultado Mayo 10, 2010.

[17] Kent Beck, James Grenning, and Robert C. Martin. Manifesto for agile software development. Internet: `http://agilemanifesto.org/principles.html` Consultado Mayo 10, 2010.

[18] José Canós, Patricio Letelier, and Carmen Penadés. Metodologías Ágiles en el desarrollo de software. 2003. Universidad Politécnica de Valencia.

[19] Paul Clements. Comparing the sei's views and beyond approach for documenting software architectures with ansi-ieee 1471-2000. July 2005.

[20] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Boston, MA, 2003.

[21] Paul Clements, James Ivers, Reed Little, Robert Nord, and Judith Stafford. Documenting software architectures in an agile world. (CMU/SEI-2003-TN-023), 2003.

[22] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2001.

[23] Alistair Cockburn. *Agile software development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[24] Alistair Cockburn. Crystal light methods, 2003-2008. Internet: `http://alistair.cockburn.us/Crystal+light+methods` Consultado: 14 de Febrero, 2013.

[25] Secretaría de Economía. Programa para el desarrollo de la industria del software (prosoft). Internet: `http://www.prosoft.economia.gob.mx/` Consultado: Febrero, 2013.

[26] Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10):859–866, October 1972.

[27] Magdalena Manuela Dávila Muñoz. Desarrollo de una espcialización de moprosoft basada en el método ágil scrum. Master's thesis, Universidad Nacional Autónoma de México, 2008.

[28] Eclipse. Openup. Internet: `http://epf.eclipse.org/wikis/openup/` Consultado: Noviembre, 2010.

[29] Real Academia Española. Real academia española. Internet: `www.rae.es` Consultado: 10 de enero de 2013.

[30] Saima Farhan, Huma Tauseef, and Muhammad Abuzar Fahiem. Adding agility to architecture tradeoff analysis method for mapping on crystal. *Software Engineering, World Congress on*, 4:121–125, 2009.

[31] Adalberto González Ayala. Adaptar los métodos Ágiles como ventaja competitiva. Internet: `http://www.sg.com.mx/content/view/927` Consultado: Mayo 10, 2010.

[32] Rob Hathaway. What is kanban, Mayo 2009. Internet: `http://www.limitedwipsociety.org/2009/05/29/what-is-kanban-2/` Consultado: 15 de junio 2010.

[33] J.A. Highsmith. *Adaptive software development: a collaborative approach to managing complex systems*. Dorset House Pub., 2000.

[34] Jim Highsmith. *Agile software development ecosystems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[35] Watts S. Humphrey. *PSP(sm): A Self-Improvement Process for Software Engineers (Sei Series in Software Engineering)*. Addison-Wesley Professional, 2005.

[36] Software Engineering Institute. Software engineering institute. Internet: `http://www.sei.cmu.edu` Consultado: Junio, 2010.

[37] M. Isham. Agile architecture is possible you first have to believe! pages 484 –489, 4-8 2008.

[38] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[39] Veerapaneni Esther Jyothi and K. Nageswara Rao. Effective implementation of agile practices ingenious and organized theoretical framework. *International Journal of Advanced Computer Science and Applications*, 2, 2011.

[40] Rick Kazman, Jai Asundi, Mark Klein, Norton L. Compton, and Lt Col. Making architecture design decisions: An economic approach, 2002.

[41] Alan S. Koch. *Agile Software Development: Evaluating the Methods for Your Organization*. Artech House, 2005.

[42] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[43] Patricio Letelier and Carmen M. Penadés. Métodologías Ágiles para el desarrollo de software: extreme programming (xp). 2004.

[44] John D. Mcgregor. The architecture tradeoff analysis method (atam). Technical report, 2004.

[45] Robert L. Nord and James E. Tomayko. Software architecture-centric methods and agile development. *IEEE Software*, 23:47–53, 2006.

[46] Hanna Oktaba, Claudia Alquicira Esquivel, Angélica Su Ramos, Alfonso Martínez Martínez, Gloria Quintanilla Osorio, Mara Ruvalcaba López, Francisco López Lira Hinojo, María Elena Rivera López, María Julia Orozco Mendoza, Yolanda Fernández Ordóñez, and Miguel Ángel Flores Lemus. Moprosoft. Internet: `http://www.comunidadmoprosoft.org.mx/` Consultado: Noviembre, 2010.

[47] Rob Wojcik Robert L. Nord, James E. Tomayko. Integrating software-architecture-centric methods into extremme programming (xp). Technical report, Software Engineering Institue, 2004. Technical Note CMU/SEI-2004-TN-036 September 2004.

[48] Ken Schwaber. *Agile Project Management With Scrum*. Microsoft Press, Redmond, WA, USA, 2004.

[49] SEI. Appendix d: Glossary. Internet: `www.sei.cmu.edu/cmmi/tools/cmmiv1-3/upload/DEV-AppD-compare.pdf` Consultado: 27 de octubre de 2012.

[50] D. Turk, R. France, and B. Rumpe. Limitations of agile software processes, May 2002.

[51] Wikipedia. The agile unified process (aup). Internet: `http://en.wikipedia.org/wiki/Dynamic_systems_development_method` Consultado Mayo 10, 2010.

[52] Rob Wojcik, Felix Bachmann, Len Bass, Paul C. Clements, Paulo Merson, Robert Nord, and William G. Wood. Attribute-driven design (add), version 2.0. Technical report, Software Engineering Institute, November 2006.

[53] Erick Andrey Serratos Álvarez. Paquetes de puesta en operación iso/iec29110. *Software Guru*, 33(33):40, Agosto-Octubre 2011.

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Casa abierta al tiempo

UNIDAD IZTAPALAPA - DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

Modelo Genérico para el Desarrollo de la Arquitectura de Software en Metodologías Ágiles

T E S I S

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS
(CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN)

PRESENTA:

LIC. JOSÉ FIDEL URQUIZA YLLESCAS

ASESORES:
M. EN C. ALFONSO MARTÍNEZ MARTÍNEZ
M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ

JURADO CALIFICADOR:

Presidente: DRA. HANNA JADWIGA OKTABA (UNAM)
Secretario: ING. LUIS FERNANDO CASTRO CAREAGA (UAM-I)
Vocal: M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ (UNAM)

México, D.F., Julio 2013