ICT-UNPA-57-2013
ISSN: 1852 - 4516
Aprobado por Resolución Nro. 0164/13-R-UNPA

METODOLOGÍAS ÁGILES ENFOCADAS AL MODELADO DE REQUERIMIENTOS

Silvia Gabriela Rivadeneira Molina
grivadeneira@uart.unpa.edu.ar

Unidad Académica Rio Turbio – Universidad Nacional de la Patagonia Austral
Avda. de los Mineros 1260, Río Turbio, Provincia de Santa Cruz, República Argentina
Mayo 2012

RESUMEN

El dinamismo y la variabilidad de la industria del software hace necesario replantear las bases y prácticas que sustentan el desarrollo de software convencional. Un reciente estudio realizado por (Boehm, 2006), sobre tendencias en ingeniería del software, muestra que el mercado actual está caracterizado por el desarrollo rápido de aplicaciones y la reducción de la vida de los productos. En este contexto variable e inestable se requiere como ventaja competitiva aumentar la productividad y satisfacer las necesidades cambiantes de los clientes en el menor tiempo posible para proporcionar un mayor valor al negocio. Ante este panorama, cabe reflexionar sobre el grado de adaptación de las metodologías convencionales a estas circunstancias. La mayoría de los estudios coinciden en que el carácter normativo y la fuerte dependencia de planificaciones previas al desarrollo que definen a las metodologías convencionales, implican que resulten excesivamente *pesadas* para cubrir las necesidades de un amplio porcentaje del mercado software actual.

Las metodologías ágiles surgidas a fines de los ’90, no han sidomuy tenidas en cuenta por el mundo académico, quizás recién a mediados de la década anterior se comienzan a encontrar trabajos de investigación al respecto.

En este trabajo se analizan las metodologías ágiles más utilizadas enfocándose en las primeras etapas del ciclo de vida del software ydeteniéndose en las prácticas, técnicas y productos que se utilizan en las mencionadas etapas. Para ello se indagó en la literatura existente para conocer cada una de las metodologías involucradas, logrando una primera selección y comparación de las mismas, y exponiendo los resultados y sus conclusiones en este informe.

Palabras claves: metodologías ágiles, modelado de requerimientos, gestión de requerimientos, ingeniería de requerimientos, ciclos de vida de software

INTRODUCCIÓN

El objeto del presente documento es realizar un estudio del estado del arte respecto de las metodologías ágiles aplicadas al modelado de requerimientos.

Este trabajo se realiza en el marco del Proyecto de Investigación 29/B134 “Modelado de requerimientos y diseño de sistemas complejos” radicado en Unidad Académica Caleta Olivia de la Universidad Nacional de la Patagonia Austral.

El documento se organiza como sigue: en la sección 1 se presentan los marcos referenciales de este trabajo que consisten en una cronología de las metodologías de desarrollo de software, los conceptos utilizados y el marco teórico conteniendo lo que interesa a la ingeniería de software y en especial las metodologías de desarrollo de software, en la sección 2 los materiales, métodos utilizados en este trabajo, los resultados y los debates en torno a este trabajo en particular.

1. MARCOS DE REFERENCIA

1.1 MARCO HISTÓRICO

La cronología de la aparición de las distintas metodologías de desarrollos pueden clasificar según Avison y Fitzgerald (1995, citado en Carvajal 2008) en:

1. Pre – metodologías: entre los años 50 y 60.
2. Primeras metodologías: entre los años 1970 y 1980.
3. La era de las metodologías: entre los años 80 y los primeros años de los 90.
4. Era post-metodológica: En los años 90.

La era *pre metodológica* se caracteriza por la ausencia de metodologías, grandes computadoras, las cuales ejecutaban aplicaciones científicas y militares. Poco a poco se fueron introduciendo en organizaciones empresariales realizando por ejemplo informes de ventas, guardando información de usuarios o archivos.

Los sistemas eran desarrollados por programadores, planificados como pequeños ejercicios, se ejecutaban en cortos plazos de tiempo. Los usuarios no siempre quedaban satisfechos con las soluciones presentadas, la documentación era escasa y a veces no existía, los cambios requerían más tiempo y complejizaban la aplicación, y los programadores eran cotizados porque eran los únicos que conocían el funcionamiento.

Se dieron cuenta que necesitaban más tiempo para el análisis y diseño de las aplicaciones, aparecieron las figuras de los analistas de sistemas quienes eran los intermediarios entre el programador y el sistema; y, los operadores que controlaban el funcionamiento de las computadoras.

La era de las *primeras metodologías* comienza en 1971 con el nacimiento del modelo en cascada (Royce, 1970). Esta metodología consiste en una serie de etapas que es necesario

cumplir secuencialmente, cada etapa tiene asociada un producto entregable para ser considerada finalizada.

Entre los inconvenientes que poseía este modelo se encuentran la imposibilidad de obtener todas las necesidades de los clientes, la inestabilidad, la inflexibilidad, la insatisfacción de los clientes, exceso de documentación y la facilidad de desviarse de los tiempos planificados.

En la era de las *metodologías* surgieron un gran número de nuevos enfoques en respuesta a las limitaciones del modelo en cascada. Se pueden mencionar dos corrientes: la que busca mejorar el modelo en cascada y la que decide hacer algo diferente. Asimismo, en los 70 surgieron una variedad de técnicas y herramientas que mejoraron el modelo, tales como: el modelo entidad-relación, la normalización, diagramas de flujos de datos, diagramas de actividad, diagramas estructurados; y herramientas de gestión de proyectos de software, repositorios de código, de modelado, y las Computer Aided Software Engineering (CASE[1]).

Entre las metodologías que actualizan o mejoran el modelo en cascada se pueden encontrar: MERISE, Structured Systems Analysis and Design Method (SSADM[2]) o Yourdon System Method. Estas integran las herramientas y técnicas mencionadas.

Entre las metodologías alternativas emergieron (Avison y Fitzgerald 1995):

- *Sistemas.* Utilizan técnicas cualitativas para aplicar pensamiento sistemático a situaciones no sistemáticas, un ejemplo es Soft Systems Methodologies (SSM[3]).
- *Estratégicas.*Se enfocan en la planificación previa al desarrollo de software y obtener los objetivos de negocio, un ejemplo es Business Systems Planning.
- *Participativas*. Se caracterizan por la participación de los usuarios y demás interesados en las fases de análisis, diseño e implementación, un ejemplo es ETHICS.
- *Prototipadas*.Se distinguen por el uso de prototipos que permite a los usuarios visualizar y validar el desarrollo, un ejemplo es Rapid Application Development (RAD[4]).
- *Estructuradas*. Extienden los conceptos de programación estructurada al análisis y diseño, con técnicas que permiten el análisis descendente y representación compleja de procesos.
- *Análisis de datos*.Se centra en entender y documentar los datos, la técnica ampliamente utilizada es el modelado entidad – relación, un ejemplo es Information Engineering.

Los primeros años de la década del 90 nacieron un nuevo grupo de metodologías (Carvajal 2008):

- *Orientadas a objetos*. Enfocada en objetos que representan entidades del mundo real.
- *Desarrollo incremental o evolutivo*. Enfocado en prototipado y desarrollo de software incremental. La metodología Dynamic System Development Methods (DSDM[5]).
- *Específicas*. Fueron concebidas para un tipo de proyectos, tienen un propósito determinado:
    - Welti, para el desarrollo de aplicaciones Enterprise Resource Planning (ERP[6]).

[1] Ingeniería de Software Asistida por Computadora.
[2] Método de Análisis y Diseño Estructurado de Sistemas.
[3] Metodologías de Sistemas Suaves.
[4] Desarrollo Rápido de Aplicaciones.
[5] Método de Desarrollo de Sistemas Dinámicos.

- CommonKADS, para el desarrollo de aplicaciones de gestión del conocimiento.
- Process Innovation[7] para el desarrollo de aplicaciones de reingeniería de procesos de negocio.
- Renaissance, metodología que se basa en la técnica de ingeniería inversa para sistemas heredados.
- Web IS Development Methodology (WISDM[8]), para el desarrollo de aplicaciones basadas en el Web.

A fines de los 90 trasciende una corriente caracterizada por una seria reevaluación de los supuestos beneficios de las metodologías, reacciones casi violentas que dan lugar a la era *postmetodológica*. Las alternativas que se presentaron en esta era fueron:

- *Desarrollo externo*. Muchas empresas decidieron no realizar más proyectos software y compraban todos sus requisitos empaquetados.
- *Mejora continua*. Abogan por la continua mejora de las metodologías.
- *Desarrollo ad hoc y contingencia*. El desarrollo ad hoc es un regreso a la era pre-metodológica, sin metodología.
- *Desarrollo contingente o flexible*. Es un enfoque mássuavizado, menos prescriptivo, donde existe una estructura pero sus componentes se dejan a elección de los desarrolladores.
- *Desarrollo ágil*. Se enfoca en la participación de los usuarios y clientes más que en procesos y herramientas, trabajando más en el software y menos en la documentación, colaborando más con clientes en vez de negociar y responder a los cambios por encima de la planificación de los tiempos del trabajo.

1.2 MARCO CONCEPTUAL

En la introducción y marco precedente se mencionan términos que es necesario definir, tales como *modelado, requerimientos, metodologías,* a ellos les agregaremos los que utilizaremos a lo largo de este escrito *técnicas,método, proceso de desarrollo de software,ingeniería de requerimientos.*

Una *metodología* de desarrollo de software es un marco de trabajo utilizado para estructurar, planificar y controlar el proceso de desarrollo de software. Este marco de trabajo consiste en una filosofía de desarrollo de software, herramientas, modelos y métodos que asisten al proceso de desarrollo de software.

Una *técnica* es según el Diccionario de la Real Academia Española, entre otras, un conjunto de procedimientos y recursos de que se sirve una ciencia o un arte. Una técnica no es una metodología, ya que ésta última puede utilizar varias técnicas para lograr crear un producto software.

Un *proceso de desarrollo de software* **describe un enfoque para la construcción, desarrollo y, posiblemente, mantenimiento del software**, según Larman (2003). Carvajal (2008) lo

[6] Planeamiento de Recursos Empresariales.
[7] Innovación de procesos.
[8] Metodología de Desarrollo de Sistemas de Información Web.

define como el conjunto de actividades que se llevan a cabo para construir un producto software, agregando que las metodologías permitirán al proceso marcar un camino común para producir software con éxito.

Un *modelode procesos de software* es una simplificación de un proceso real, los diferentes modelos de proceso que podemos encontrar son:

- Modelo secuencial. Cuyo exponente más conocido es el Modelo en Cascada que resuelve el problema de los requisitos variables no permitiendo que se modifiquen una vez definidos.
- Modelos iterativos e incrementales.Ambos dividen el ciclo de desarrollo en etapas más cortas y repiten el modelo en cascada en cada una de ellas. El Unified Process (UP[9]) se conoce como desarrollo iterativo e incremental (Larman 2003).
    - Incremental. Reduce el tiempo de desarrollo, dividiendo el proyecto en intervalos incrementales superpuestos.
    - Iterativo. Centrado en capturar requisitos cambiantes y gestión de riesgos. El ciclo se divide en iteraciones de diferente longitud cada una produciendo un producto completo y entregable.
- Modelo en espiral. Tiene las características del ciclo de vida clásico y el prototipado (desarrollo iterativo), incluyendo análisis de alternativas, identificación y reducción de riesgos.

El Diccionario de la Real Academia Española diferencia los términos *requerimiento* y *requisito* de la siguiente forma.
Requerimiento:
1. m. Acción y efecto de requerir.
Requisito:
(Del lat. *requīsitus*).
1. m. Circunstancia o condición necesaria para algo.

A los fines de este escrito, y nuestra investigación, se utilizarán ambos términos indistintamente. A continuación se citarán algunas definiciones del término requerimiento para seleccionar y justificar la elección de una única definición en el marco de este trabajo.

La definición de *requerimiento* o *requisito* según IEEE Std. 610.12-1990:

1) **Una condición o una capacidad necesitada por un usuario para solucionar un problema o alcanzar un objetivo.**
2) **Una condición o una capacidad que se debe resolver o poseer por un sistema o un componente del sistema para satisfacer un contrato, un estándar, una especificación, o alguna impuesta formalmente por documento.**
3) **Una representación documentada de una condición o de una capacidad como en (1) o (2).**

Jackson (1997) expresa **un requerimiento es una condición sobre los fenómenos del ambiente**.

En el SWEBOK (2004) se define un requerimiento como la propiedad que resuelve un problema del mundo real.

[9]Proceso Unificado.

Sommerville (2005) define a un requerimiento como **una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste**. El autor también los clasifica según niveles de descripción en *requerimientos de usuario* para designar a los requerimientos abstractos de alto nivel y, los *requerimientos del sistema* para designar a la descripción detallada de lo que el sistema debe hacer.

Pérez y Jaime (2009) lo definen como **la descripción de una condición que debe cumplir un sistema, que puede ser derivado de una necesidad de usuario identificada, o estipulada en un contrato, estándar u otro documento formalmente impuesto al inicio del proceso**.

Este trabajo adhiere a la definición propuesta por Sommerville (2005) entendiendo como *declaración abstracta de alto nivel*a una manifestación o explicación de alguna cualidad esencial, expresada de una manera adecuada a la capacidad cognitiva humana. De esta forma, la primera clasificación realizada por el autor se relaciona con los tipos de usuario que realizan o, a la que están dirigidas, estas manifestaciones.

En cuanto a ingeniería de requerimientos, Sommerville (2005) manifiesta que **la meta de la ingeniería de requerimientos es crear y mantener un documento de requerimientos del sistema**. Además, el proceso está conformado por cuatro subprocesos, tales como: estudio de viabilidad, obtención y análisis de requerimientos, especificación de requerimientos y la validación de los mismos.

Pérez y Jaime (2009) agregan que **su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente, verificable y compacta, las necesidades de los usuarios finales** pero no contemplan el estudio de viabilidad en el proceso ya que destacan que**los resultados del proceso de ingeniería de requerimientos contribuyen a la determinación de viabilidad de desarrollar el software**.

El término ingeniería de requerimientos en SWEBOK (2004) está implícito en el área de conocimiento de requerimientos de software, donde la elicitación, el análisis, la especificación y la validación de requerimientos de software conforman subáreas.

Respecto de los términos *modelado* y *metodología* los desarrollamos más en el siguiente marco.

1.3 MARCO TEÓRICO

La ingeniería de software es una disciplina que permite construir soluciones software de calidad, a partir de distintos componentes que se ocupan de diversos aspectos del problema a resolver, aplicando una variedad de métodos, herramientas, procedimientos y paradigmas.

- Los métodos de ingeniería de softwarenos dirán “como” construir técnicamente el software. Estos abarcan una serie de actividades tales como: análisis, diseño, codificación, pruebas y mantenimiento.
- Las herramientas de ingeniería de software son instrumentos automatizados para utilizar en métodos o procedimientos.

- Los procedimientos de ingeniería de software son combinaciones de herramientas y métodos para llegar a un producto en particular.
- El paradigma de ingeniería de software representará un enfoque particular para construir el software.

Booch, Rumbaugh y Jacobson (2006) establecen que **un modelo es una simplificación de la realidad**, que modelamos **para comprender mejor el sistema que estamos desarrollando**, y agregan que cuando el sistema se complejiza, el modelado se vuelve muy importante ya que si no lo construimos **no podemos comprender el sistema en su totalidad**.

En el software existen muchas maneras de enfocar un modelo, las más comunes son: la tradicional, perspectiva algorítmica y, una más actual, perspectiva orientada a objetos. La diferencia radica en que, en un enfoque algorítmico, cuando los sistemas crezcan y los requisitos cambien, resultarán difíciles de mantener, ya que su bloque de construcción principal es el procedimiento o función. En el enfoque que trabajaremos, el principal bloque de construcción es el objeto o clase (conjunto de objetos suficientemente similares), y ese objeto tiene una identidad, un estado y un comportamiento.

Algunos autores (Carbajal, 2008) clasifican las prácticas de desarrollo en metodologías tradicionales y metodologías ágiles, colocando dentro de las tradicionales las orientadas a objetos y como principal abanderado el UP.

Larman (2003) declara que el análisis pone énfasis en la investigación del problema y los requisitos, en lugar de ponerlo en la solución, y que por lo tanto, al ser el término *análisis* amplio prefiere calificarlo como *análisis de requisitos* o *análisis de objetos*. Asimismo, establece que el *análisis orientado a objetos* presta especial atención a encontrar y describir los objetos (o conceptos) del dominio del problema. El autor, en su obra, se enfoca, aunque no completamente, en el UP centrándose en ideas y artefactos comunes relacionados con Análisis y Diseño Orientado a Objetos (A/DOO) y el análisis de requisitos.

El análisis de requerimientos según Pressman (2002) **es una tarea de ingeniería del software que cubre el hueco entre la definición del software a nivel sistema y el diseño de software**. El modelado de análisis debe lograr tres objetivos básicos: (1) describir lo que desea el cliente, (2) construir una base para el diseño de software, y (3) definir un conjunto de requerimientos que se pueda validar.

Tal como lo mencionara anteriormente, para resolver los problemas reales en una organización los ingenieros de software deben seleccionar una estrategia de desarrollo que habitualmente se denomina modelo de proceso de software o paradigma de ingeniería de software. Castro (2004) en su artículo brinda una explicación breve pero práctica de algunas técnicas y metodologías que produce la intersección entre Ingeniería de Software y la Orientación a Objetos. El autor menciona a la gestión de flujos de trabajo como una de las tecnologías orientadas a resolver los problemas de desarrollo de software cuyas dos aproximaciones más populares son el Rational Unified Process (RUP[10]), un refinamiento detallado del UP, y la Extreme Programming (XP[11]), una de las metodologías ágiles de desarrollo.

[10]Proceso Unificado de Rational.
[11]Programación Extrema.

Las metodologías ágiles surgen por ser las más adecuadas para proyectos pequeños donde el entorno del sistema es muy cambiante y se exige reducir al máximo los tiempos de desarrollo manteniendo una alta calidad.

En marzo de 2001 se reúnen en Salt Lake City 17 representantes de metodologías ágiles para discutir los nuevos métodos, allí nace el Manifiesto por el Desarrollo Ágil del Software (Figura 1).

Figura 1. Manifiesto Ágil.

Los que firmaron el manifiesto original fueron:

Kent Beck (XP)	James Grenning (XP)	Robert C. Martin (XP)
Mike Beedle (DSDM)	Jim Highsmith (ASD)	Steve Mellor (Scrum)
Arie van Bennekum (DSDM)	Andrew Hunt (Pragmatic Programming)	Ken Schwaber (Scrum)
Alistair Cockburn (Crystal)	Ron Jeffries (XP)	Jeff Sutherland (Scrum)
Ward Cunningham (XP)	Jon Kern (FDD)	Dave Thomas (Pragmatic Programming)
Martin Fowler (XP)	Brian Marick (XP)	

Las metodologías ágiles impulsan generalmente una gestión de proyectos que promueve el trabajo en equipo, la organización y responsabilidad propia, un grupo de buenas prácticas de ingeniería de software que brindan una entrega rápida de software de alta calidad, y un enfoque de negocios que alinea el desarrollo con las necesidades del cliente y los objetivos de la compañía (LNCS 2009).

Existe una gran variedad de metodologías ágiles cada una tiene sus particularidades, manejan roles muy peculiares, suponen un conjunto de expertos con habilidades para resolver desde problemas técnicos hasta los más abstractos y complejos, por lo tanto exigen experiencia. Definen ciclos, que en algunos casos incluyen varias fases, similares a los procesos de desarrollo de software tradicionales (como el UP). Generan documentación indispensable, aunque el proceso para generarlas es menos rígido.Aunque lo más importante es que están

abiertas a modificar requerimientos, sin importar su impacto en la arquitectura del sistema (Urquiza et al 2010).

Una comparación básica entre metodologías ágiles y metodologías tradicionales puede verse en la Tabla 1 (Canós et al 2003). Estas diferencias no solo afectan al proceso de desarrollo de software, sino también al contexto del equipo y a la organización.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos Más artefactos	Pocos roles Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 1. Diferencias entre metodologías ágiles y metodologías tradicionales.

2. RESULTADOS, ANÁLISIS Y DISCUSIÓN

2.1 MATERIALES Y MÉTODO

Los materiales a utilizar en este trabajo serán artículos científicos teóricos y empíricos referentes alas Metodologías Ágiles aplicadas al Modelado de Requerimientos, y el método será Investigación Básica.

2.2 RESULTADOS

La atención de los estudios sobre metodologías ágiles realizados hasta el momento se centra en la situación antes y durante el desarrollo inicial. Por ejemplo, (Boehm y Turner) establecen cinco dimensiones de riesgo a considerar para decidir que metodología utilizar (tamaño, criticidad, dinamismo, equipo y cultura), sin considerar el tipo de desarrollo. Evidencias empíricas en este sentido podrían ayudar a las organizaciones a decidir qué método escoger.

Tal como lo mencionara en el marco teórico, la revisión de literatura nos permite visualizar que existen diversas metodologías agiles, por lo que es necesaria una selección de ellas. Se tomarán los criterios utilizados por Carvajal (2008) tales como (a) la metodología más presente en Internet, (b) la metodología más documentada, (c) la metodología con certificaciones y entrenamiento, (d) la metodología con comunidad propia, (e) la metodología más utilizada por organizaciones y (f) la metodología más utilizada en proyectos software; como resultado el autor obtuvo las siguientes:

1. Agile Project Management (APM[12])
2. Crystal Methods
3. Dynamic System Development Methods (DSDM)
4. Extreme Programming (XP)
5. Scrum
6. Test Driven Development (TDD[13])

2.2.1 AGILE PROJECT MANAGEMENT

Highsmith propone una nueva visión de gestión de proyectos ágiles, la gestión de proyectos ágiles y adaptativos. Asimismo plantea que los procesos deben ajustarse a los objetivos de negocio, si ésos son repetibles y predecibles, entonces un proceso prescriptivo es lo más adecuado, pero si los objetivos de negocio son innovadores, entonces el marco de trabajo de los procesos debe ser ágil, flexible y adaptable (citado en Carvajal 2008).

El *proceso* APM se compone de las fases:

- *Previsión*. Se define la visión del producto, objetivos del proyecto, comunidad del proyecto, y el equipo que trabajará.
- *Especulación*. Se genera un plan de entregas basado en funcionalidad más que en actividades. El plan utiliza información sobre la especificación del producto, plataforma de arquitectura, recursos, análisis de riesgo, niveles de defecto, restricciones de negocio y fechas objetivo.
- *Exploración*. Se obtiene funcionalidades de la aplicación probadas y aceptadas. Un gestor se ocupa de crear los equipos auto-organizados que puedan alcanzar los objetivos. No es lineal, es iterativo, es concurrente y no sigue el modelo en cascada.
- *Adaptación*. Se revisan los resultados liberados, la situación actual y el rendimiento de la aplicación, adaptándose si es necesario.
- *Cierre*.Se concluye el proyecto, se aprende de la experiencia y se celebra.

Los *roles* que podemos encontrar son:

- *Patrocinador ejecutivo*. Es la persona o grupo que lidera el producto.
- *Gestor de proyectos*. Es la persona que lidera el equipo que libera o entrega el producto.
- *Gestor de producto*. Es la persona que lidera el equipo que determina los resultados a entregar.
- *Ingeniero jefe*. La persona que guía los aspectos técnicos del producto.
- *Gestores*. Un grupo que puede estar a cargo de organizaciones participantes.

[12] Administración de Proyectos Ágiles.
[13] Desarrollo Dirigido por Pruebas.

- *Equipo de cliente*. Grupo que determinan las características a ser desarrolladas y priorizadas.
- *Equipo de proyecto*. Grupo que libera o entrega los resultados.
- *Proveedores*. Organizaciones externas o personas físicas que proveen servicios o componentes.
- *Gobierno*. Entes reguladores que requieren información, reportes, certificaciones y otros documentos legales.

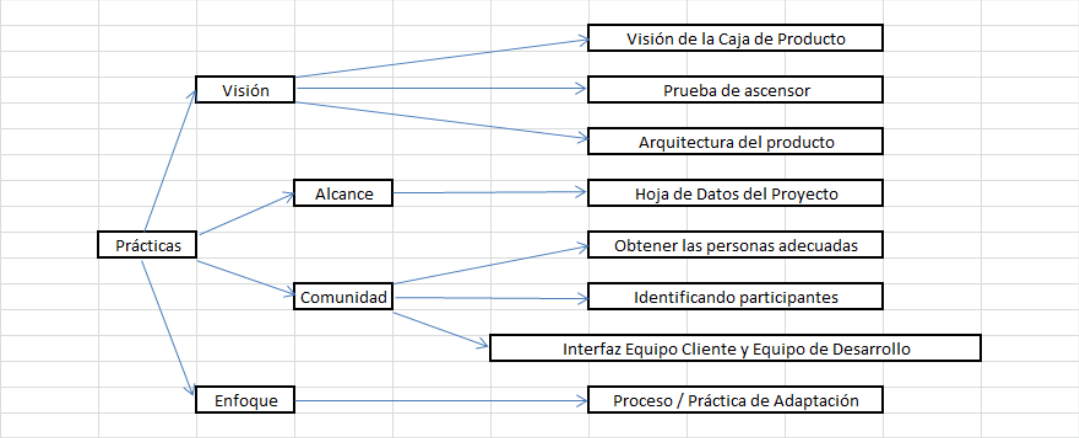

Figura 2. Prácticas de la Fase de Previsión.

Las *prácticas* que Highsmith recomienda para cada una de las fases son:

1. Prácticas ágiles iniciando proyectos y planificándolos (ver Figura 2 y 3).
    a. La *caja del producto* como visión. Se realiza una imagen visual del producto. El equipo se divide en grupos de cuatro a seis personas para diseñar la caja del producto: nombre, imagen gráfica, tres a cuatro puntos clave para vender el producto, descripción detallada y requerimientos de funcionamiento.
    b. *Arquitectura del producto*. Se muestran las partes complejas de la estructura interna del proyecto.
    c. *Hoja de datos de proyecto*. Se transmite la esencia traducida en objetivos, planificación, recursos, y como el proyecto entregará la visión.
    d. *Listado de características del producto*. Se trata de expandir la visión del producto, a través de un proceso evolutivo de definición de requerimientos del producto, una lista de requerimientos.
    e. *Tarjetas de especificación de rendimiento*. Se documentan las principales operaciones y requerimientos de rendimiento del producto que se construirá.
2. Prácticas ágiles para obtención de resultados. Corresponde a fase de exploración.
    a. *Gestión de carga de trabajo*. El gestor de proyectos debe monitorizar las actividades diarias necesarias para completar las iteraciones.
    b. *Coaching y desarrollo de equipo*. Se espera incrementar la capacidad del equipo, ayudando a sus miembros a mejorar conocimientos, autodisciplina y habilidades de trabajo en grupo.
    c. *Reuniones diarias de equipo*. Se coordinan las actividades de los miembros diariamente.
    d. *Decisiones consensuadas y participativas*. Provee a los miembros del proyecto con prácticas que permiten elaborar, hacer y analizar las decisiones a tomar a lo largo del desarrollo.

3. Prácticas ágiles para reflexionar, aprender y adaptarse. Corresponde a fases adaptación y cierre.
    a. *Revisión de adaptación del producto, proyecto y equipo*. Asegurar retroalimentación frecuente, y que se den altos niveles de aprendizaje en todas las dimensiones del proyecto. Actividades como: reuniones orientadas al cliente, revisiones técnicas, evaluaciones de rendimiento de equipo, reportes de estado de proyecto.

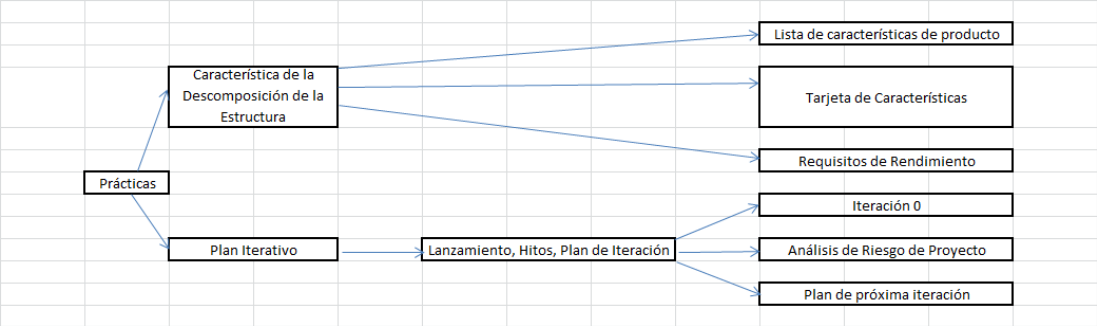

Figura 3. Prácticas de la Fase de Especulación.

2.2.1.1 MODELADO DE REQUERIMIENTOS Y APM

APM se centra en dos de las fases iniciales del ciclo de vida de desarrollo de software (inicio del proyecto y especificación de requerimientos) dando soporte para la gestión del proyecto, describiendo un proceso, prácticas, actividades y artefactos. Highsmith (2004) manifiesta que la pregunta más frecuente acerca de APM es ¿qué ocurre con las fases planeamiento, arquitectura y requerimientos? Y su respuesta es que esas cosas son actividades y no fases.

Más adelante, declara que el éxito de un producto software implica cumplir con las expectativas del cliente, de los interesados de gestión y del equipo de proyecto, pero que las expectativas no son lo mismo que los requerimientos, mientras que estos últimos son tangibles, las expectativas no lo son.

Las fases que interesan al proceso de requerimientos en APM son las de Previsión y Especulación. Las prácticas y los artefactos que se pueden obtener de las mismas se muestran en las Figuras 2 y 3.

2.2.2 CRYSTAL METHODS

Las metodologías Crystal fueron creadas por Alistair Cockburn.

La familia de métodos Crystal define un código de color para establecer la complejidad de la metodología: si es más oscuro entonces el método es más pesado. Cuanto más crítico es el sistema, más rigor se necesita. Incluye un conjunto de principios para adaptar las diferentes metodologías según las circunstancias del proyecto.

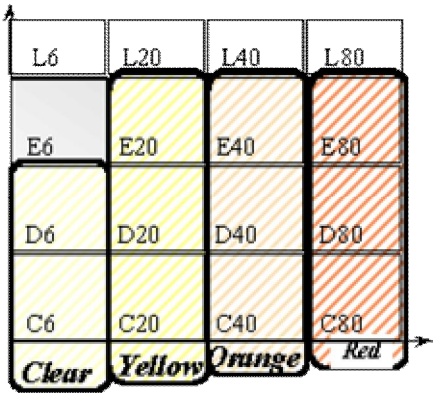

Figura 4. Criticidad de Crystal Methods.

Crystal sugiere que se elija un color de la metodología para un proyecto en función de su criticidad y tamaño. En la Figura 4 aparecen los caracteres C, D, E y L que indican los riesgos potenciales:

- C: indica pérdida de confort debido a un fallo del sistema.
- D: indica pérdida de dinero discrecional (nuestro dinero).
- E: indica pérdida de dinero esencial (el que no podemos disponer libremente).
- L: indica pérdida de vidas por el fallo del sistema.

Los números indican el número de personas afectadas al proyecto. D6 indica un proyecto con 6 personas máximo, y máxima criticidad de dinero discrecional.

Los proyectos utilizan ciclos de desarrollo incremental, de una longitud máxima de cuatro meses, sugieren periodos de uno a tres meses. Se prioriza la comunicación y la cooperación de la gente. Permiten combinarse con otras metodologías ágiles. Los métodos Crystal evocan a una gema, cada cara o faceta es otra versión del proceso, y todas se ubican alrededor de un núcleo idéntico.

Existen cuatro variantes de la metodología: Crystal Clearpara equipos de 8 o menos integrantes, Crystal Yellowpara 8 a 20 integrantes, Crystal Orangepara 20 a 50 integrantes y Crystal Redpara 50 a 100 integrantes. En este trabajo mencionaremos dos de las más utilizadas: Crystal Clear (CC) y Crystal Orange (CO).

CC está diseñado para proyectos pequeños (categoría D6), que como mencioné comprenden hasta 6 desarrolladores. Sin embargo, con alguna extensión de comunicación y pruebas puede aplicarse a proyectos E8 a D10. Para evitar las limitaciones de comunicación de la estructura es conveniente que el equipo se ubique en una oficina común.

CO está diseñado para proyectos de mediana envergadura, para equipos con una cantidad de 10 hasta 40 integrantes (categoría D40) y con una duración de proyecto de uno a dos años. Podrían existir proyectos E50 si agregamos procesos de verificación de pruebas. El proyecto puede dividirse en equipos multifuncionales (utilizando estrategias de diversidad holística). Este tipo de proyectos enfatiza la importancia del uso de Time-To-Market (TTM[14]). El equilibrio entre entregas amplias y el cambio rápido de requerimientos y diseño en un número

[14] Tiempo desde que se concibe el producto como una idea, hasta que está disponible en el mercado.

limitado de versiones entregables permiten reducir el costo del mantenimiento y se mantiene funcionando la comunicación entre los equipos.

Los *principios* de las metodologías Crystal son:

- *Normas de política*.Son prácticas que necesitan ser aplicadas durante el proceso de desarrollo.
  - Entrega incremental sobre una base regular. CC las entregas entre dos a tres meses y CO puede ser extendido a cuatro meses como máximo.
  - Seguimiento del progreso por hitos basados en entregas de software y decisiones importantes en lugar de documentos escritos.
  - Involucramiento directo del usuario.
  - Pruebas de funcionalidad de regresión automatizadas.
  - Talleres de producto (y metodología) ajustados a principios y mediados de cada incremento.
- *Artefactos*.Algunos artefactos difieren entre ambas metodologías, pero los comunes son: secuencias de entregas, modelos de objetos, manuales de usuario, casos de prueba y código. Ahora las diferencias:
  - CC incluye descripciones de características del producto y casos de uso anotados y CO necesita un documento de requerimientos.
  - La planificación en CC se debe hacer teniendo en cuenta las diferentes entregas y las reuniones con los clientes. CO exige algo más exhaustivo, por ejemplo documentos de diseño de la interfaz de usuario, reportes de estado y especificaciones de los diferentes equipos.
- *Asuntos locales*. Son procedimientos que CC especifica que deben realizarse pero no especifica cómo.
- *Herramientas*. CC necesita un compilador, herramienta de versionado y una herramienta de gestión, así como pizarras y reuniones. En el caso de CO las herramientas relacionadas con control de versiones, programar, pruebas, comunicación, seguimiento de proyecto, dibujo y medición de rendimiento.
- *Roles*.
  - En CC tenemos un sponsor, diseñador de programas senior, diseñador de programas, usuario (a tiempo parcial como mínimo).
  - En CO tenemos sponsor, experto de negocio, experto en usabilidad, facilitador técnico, diseñador/analista de negocio, gestor del proyecto, arquitecto, mentor de diseño, jefe de diseñadores-programadores, diseñador de interfaz de usuario, escritor encargado de documentar, probador.
- *Actividades y estándares*. CO propone seleccionar notaciones estándares, convenciones de diseño, estándares de formato y de calidad.

CC se enfoca en un proceso con muchos ciclos anidados, como se muestra en la Figura 5.En la mayoría de los proyectos se perciben siete ciclos (1) el proyecto, (2) ciclo de entrega de una unidad, (3) la iteración (CC requiere múltiples entregas pero no muchas iteraciones)

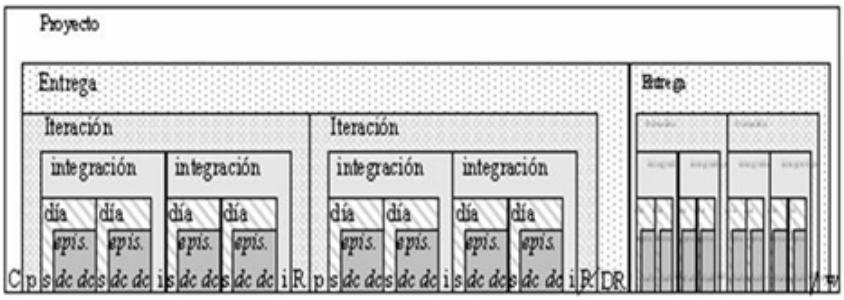

Figura 5. Ciclos Anidados de Crystal Clear.

Las letras denotan Chartering, Planeamiento de iteración, reunión diaria de pie (Standup), Desarrollo, chequeo (Check-in), Integración, taller de Reflexión, Entrega (Delivery), y empaquetado del proyecto (Wrapup).

Las *prácticas* en las metodologías Crystal son:

- Puesta en escena (staging). Consiste en la planificación del siguiente incremento. La planificación debe finalizar con una planificación ejecutable cada tres o cuatro meses. El equipo selecciona los requerimientos que serán implementados en el incremento y planifican lo que harán.
- Revisiones. Cada incremento tiene varias iteraciones y cada iteración incluye las actividades de construcción, demostración y resumen de objetivos del incremento.
- Monitoreo. Los progresos son monitoreados a partir de las diferentes entregas. El proceso se mide con los hitos clave y la estabilidad de las fases.
- Paralelismo y flujo. Cuando el monitoreo nos brinda un estado suficientemente estable es hora de pasar a la próxima etapa. En CO nos indica que los equipos pueden trabajar con la máxima eficiencia concurrente.
- Estrategia de diversidad holística. Se utiliza en CO para dividir grandes equipos funcionales en equipos multifuncionales.
- Técnica de puesta a punto de la metodología. Se basa en entrevistas y talleres para elaborar una metodología específica para el proyecto. Sirve para modificar o fijar el proceso de desarrollo.
- Puntos de vista del usuario. En CC se recomienda la opinión de dos usuarios por cada versión del producto, en CO tres revisiones por parte del cliente en cada iteración.

2.2.2.1 MODELADO DE REQUERIMIENTOS Y CRYSTAL METHODS

Los métodos Crystal no prescriben las prácticas de desarrollo, herramientas o productos, y como mencionamos se pueden combinar con otras metodologías.

Crystal cubre pocas fases del ciclo de vida pero exige la existencia y creación de artefactos en otras fases. Los artefactos que pueden utilizarse para el modelado de requerimientos en estas metodologías pueden ser:

- Declaración de la misión. Documento de un párrafo o una página que describe el propósito.

- Lista de actores – objetivos. Puede utilizarse alguna planilla de cálculo, una lista de dos columnas.
- Casos de uso anotados. Para los requerimientos funcionales.
- Archivo de requerimientos. Documentación que indica que debe construirse, quiénes los usarán, que valor tienen y qué restricciones afectan al diseño.

2.2.3 DYNAMIC SYSTEM DEVELOPMENT METHODS

El método de desarrollo de sistemas dinámicos (DSDM) se origina en 1994 en Gran Bretaña con los trabajos de Jennifer Stapleton directora del DSDM Consortium. DSDM, además, proporciona un marco de trabajo completo de controles para desarrollo rápido de aplicaciones (RAD) y lineamientos para su utilización y se puede complementar con otras metodologías.

El objetivo principal de DSDM es definir primero tiempo y costo, una vez fijados, se definen las funcionalidades que se pueden implementar en el producto. Esto es expresado en reglas que se conocen con el nombre de MoSCoW, a saber:

- Musthave (debe tener). Son los requerimientos fundamentales del sistema. El subconjunto mínimo debe ser satisfecho por completo.
- Shouldhave (debería tener). Son requerimientos importantes para los que habrá una solución a corto plazo.
- Couldhave (podría tener). Son requerimientos que podrían quedar fuera del sistema si no hay más remedio.
- Want to have but won't have this time around (se desea tener pero no lo tendrá en este momento). Son requerimientos valorados pero pueden esperar.

DSDM está compuesto por cinco fases:

1. *Estudio de viabilidad*. Se evalúa si se utiliza DSDM u otra metodología. Si se opta por DSDM se analizan las posibilidades técnicas y riesgos. Los artefactos son el Reporte de Viabilidad y Plan Sumario para el Desarrollo. Si la tecnología no se conoce se hace un pequeño prototipo para probar. Se espera una duración de pocas semanas.
2. *Estudio de negocio*.Se analizan las características del negocio y la tecnología. Se desarrollan talleres donde los expertos del cliente consideran las facetas del sistema y acuerdan prioridades. Se genera una Definición del Área de Negocios con los procesos de negocio y las clases de usuario, utilizando descripciones de alto nivel como DER o modelos de objetos de negocio. Otros artefactos son Definición de Arquitectura del Sistema (es un primer borrador y se admiten cambios) y el Plan de Bosquejo de Prototipado (establece un plan de prototipado de las siguientes etapas y gestión de configuración).
3. *Iteración de Modelo Funcional.* En cada iteración se planea el contenido y la estrategia, se realiza la iteración y se analizan los resultados pensando en las siguientes. Se produce un Modelo Funcional conteniendo el código del prototipo y los modelos de análisis. Se realizan pruebas constantemente. Hay cuatro artefactos que resultan de esta fase: Listado de Funciones priorizadas, Documentos de revisión del prototipado funcional, Listado de Requerimientos Funcionales, y, Análisis de Riesgos de Desarrollo Ulterior.

4. *Iteraciones de diseño y construcción*. El artefacto es un sistema probado que cumplimenta las reglas MoSCoW. El diseño y la construcción son iterativos, y los prototipos funcionales revisados por los usuarios.
5. *Implementación*.El sistema se transfiere al ambiente de producción. Se capacita a usuarios. Otros artefactos son: el Manual del Usuario y Reporte de Revisión del Sistema. Hay cuatro cursos de acción posibles:
    a. *Si el desarrollo satisface a los usuarios, se ha terminado*.
    b. *Si quedan requerimientos por resolver, se comienza nuevamentedesde las primeras fases*.
    c. *Si se ha dejado de lado alguna prestación no crítica, se puede comenzar desde la fase de Iteración del Modelo Funcional*.
    d. *Si algunas cuestiones técnicas no pudieron resolverse se puede comenzar desde la fase de Iteraciones de diseño y construcción*.

DSDM define hasta quince *roles* distintos, pero los más importantes son:

- *Programadores y Programadores Senior*. Senior indica el nivel de liderazgo en el equipo. Se cubren todos los roles de desarrollo.
- *Coordinador técnico*. Define la arquitectura del sistema y es responsable de la calidad técnica del proyecto, control técnico y configuración del sistema.
- Usuario embajador. Proporciona el conocimiento de la comunidad de usuarios y difunde información sobre progresos. Adicionalmente, puede existir un *Usuario Asesor* (Advisor) con otros puntos importantes de los usuarios, puede ser personal TI o un auditor funcional.
- *Visionario*. Es un usuario participante que tiene la percepción más exacta de objetivos del sistema y proyecto. Asegura que los requerimientos esenciales se cumplan y que el proyecto vaya en una dirección adecuada a éstos.
- *Patrocinador ejecutivo*. Es la persona de la organización que detenta autoridad y responsabilidad financiera.
- *Facilitador*. Es responsable de administrar el progreso del taller y motor de la preparación y comunicación.
- *Escriba*. Registra los requerimientos, acuerdos y decisiones alcanzadas en las reuniones, talleres y sesiones de prototipado.

DSDM posee nueve *prácticas* o principios, estos son:

1. Es imperativo el compromiso activo del usuario.
2. Los equipos DSDM deben tener el poder de tomar decisiones.
3. El foco radica en la frecuente entrega de productos.
4. El criterio esencial para la aceptación de los entregables es la adecuación a los propósitos del negocio.
5. Se requiere desarrollo iterativo e incremental.
6. Todos los cambios durante el desarrollo son reversibles.
7. La línea de base de los requerimientos es de alto nivel.
8. La prueba está integrada a través de todo el ciclo de vida.
9. Es esencial una estrategia colaborativa y cooperativa entre todos los participantes.

2.2.3.1 MODELADO DE REQUERIMIENTOS Y DSDM

DSDM proporciona soporte de gestión de proyecto y un proceso para las fases iniciales de ciclo de vida de un proyecto, pero ninguna práctica, actividad o artefacto específico para esas fases. Aunque sugiere complementar técnicas tradicionales con técnicas ágiles, por ejemplo en el artefacto Definición de área de negocio de la fase Estudio de Negocio. Tudor y Walter (2006) revelan que DSDM utiliza las reglas MoSCoW para priorizar los requerimientos.

Tuffs et al (1999) comparan DSDM con RUP y expresan que RUP podría considerarse una implementación de DSDM agregando que ambos se enfocan en el desarrollo iterativo, en el desarrollo de software que cumpla con las necesidades del usuario, cada proceso promueve pruebas continuas, gestión de configuración y priorización de requerimientos.

DSDM Consortium (2002) compara DSDM con XP (ver apartado 2.2.4) y encuentran muchas similitudes, por lo que es posible combinarlas, y agregan que XP es muy liviano en control de proyectos, no provee cobertura para todo el ciclo de vida y es difícil escalar a grandes proyectos, pero DSDM es liviano en técnicas de programación y bueno en prácticas. Otra diferencia es que DSDM define un ciclo de vida y conjunto de productos, mientras que XP define un conjunto de técnicas.

2.2.4 EXTREME PROGRAMMING

XP se basa en cuatro *valores*: *comunicación*, *simplicidad*, *retroalimentación* y *coraje*.

El *proceso* está constituido por seis fases:

- *Fase de exploración*. En esta fase los clientes escriben las tarjetas de historia que serán incluidas en la primera versión. Cada una de estas tarjetas describirán una funcionalidad que será agregada al programa. El periodo de tiempo de esta fase puede variar entre unas pocas semanas a unos pocos meses, dependiendo del conocimiento que posea el equipo de desarrollo con las tecnologías a utilizar.
- *Fase de planificación*. Se define la prioridad de las distintas historias y se acuerda el contenido de la primera entrega del proyecto. La primera entrega no suele tardar más de dos meses en darse. La duración de esta fase no suele exceder unos pocos días.
- *Fase de iteraciones*. La planificación divide el tiempo en varias iteraciones, de duración variable, entre una semana y cuatro. Los usuarios deciden que historias se realizarán en cada iteración, ya que la primera entrega suele contener toda la arquitectura del sistema. Las pruebas funcionales son creadas por el cliente y se ejecutan al término de cada iteración.
- *Fase de producción*. Se ejecutan una serie de pruebas extra, de rendimiento, de funcionamiento necesarias antes de entregar el producto al cliente. Si se deben hacer cambios debe decidirse si incluirlos en esta entrega o en las próximas. Las iteraciones de esta fase no deben superar las tres semanas.
- *Fase de mantenimiento*. Liberada la versión al cliente, el proyecto se debe mantener en el entorno siempre que sigan habiendo iteraciones en esa fase. Esto supone un esfuerzo adicional por lo que se sugiere contratación de personal para dar soporte a clientes.

- *Fase de cierre de proyecto*. Los clientes ya no tienen historias para ser implementadas, por lo que es necesario estar seguros que estamos cumpliendo con todas las necesidades de los clientes, y aspectos como fiabilidad, rendimiento. La documentación del proyecto se hace aquí, ya que no habrá más cambios.

Los *roles* XP son pocos. Un *cliente* que escribe historias y pruebas de aceptación; *programadores* en pares; *verificadores* que ayudan al cliente a desarrollar pruebas; *consultores técnicos*; un *consejero* que interviene y enseña; y un *rastreador* que recoge las métricas y avisa cuando se encuentra con una estimación riesgosa; además de un *Gran Jefe*.

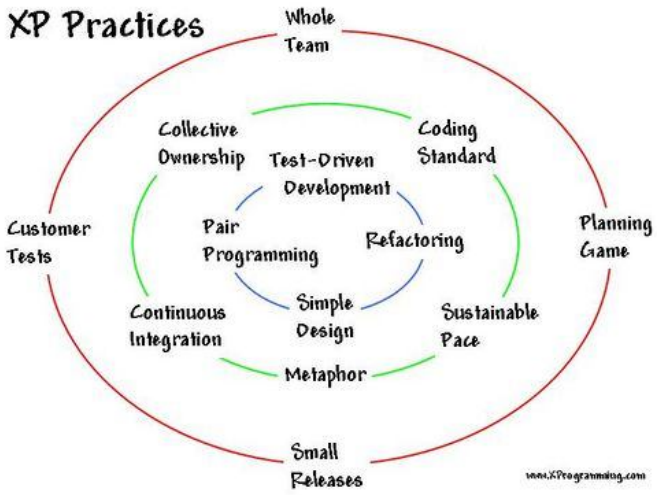

Figura 6. Prácticas XP.

Sus *prácticas* son las siguientes (ver Figura6):

1. *Juego de planeamiento.*Los programadores estiman el esfuerzo necesario para implementar las *historias del cliente* y el cliente decide sobre el alcance y la agenda de las entregas. Las historias se escriben en pequeñas fichas, que a veces se descartan. Cuando sucede, lo que queda que es parecido a un requerimiento son varias pruebas automatizadas, las pruebas de aceptación.
2. *Entregas pequeñas y frecuentes*. Se produce un pequeño sistema rápidamente (con pequeños cambios cada vez), al menos uno cada dos o tres meses.
3. *Metáforas del sistema.*El sistema es definido mediante una metáfora o conjunto de metáforas. Una metáfora es una historia compartida por clientes y programadores que guiará todo el proceso describiendo el funcionamiento del sistema.
4. *Diseño simple.*Se elimina lo innecesario, se define la menor cantidad de clases. Se intenta no duplicar código.
5. *Prueba continua*. Este desarrollo se orienta a pruebas (Test Driven Development, TDD). Existen dos clases de pruebas: la prueba unitaria y la prueba de aceptación. Los clientes escriben pruebas funcionales. Las pruebas unitarias se escriben antes que el código y lo hace el desarrollador.
6. *Refactorización continua*. Reestructurar el sistema eliminando duplicaciones, mejorando la comunicación y añadiendo flexibilidad.
7. *Programación en pares.*El código es escrito por un par de programadores en la misma computadora. Mientras uno escribe el otro va revisando el código en tiempo real.
8. *Propiedad colectiva del código*. Cualquier persona puede modificar el código en cualquier momento, siempre que escriba la prueba correspondiente.

9. *Integración continua*.Cada pieza se integra al código fuente siempre que esté lista, varias veces al día. Debe ejecutarse la prueba antes y después de la integración.
10. *Ritmo sostenible*.Se establece un máximo de 40 horas semanales de trabajo.
11. *Todo el equipo en el mismo lugar*. El cliente debe estar disponible siempre que se lo necesite.
12. *Estándares de codificación*. Es necesario seguir reglas de codificación y comunicarse a través de código.
13. *Espacio abierto*. Se recomienda una sala grande con pequeños cubículos, o mejor aún, sin divisiones. Los pares de programadores deben estar en el centro.
14. *Reglas justas*. El equipo tiene sus propias reglas para cumplir pero pueden ser modificables.

2.2.4.1 MODELADO DE REQUERIMIENTOS Y XP

Reynoso (2004) establece que **XP demanda comunicación oral tanto para los requerimientos como para el diseño**. Entre los artefactos que utiliza XP para el modelado de requerimientos encontramos las *tarjetas de historia* (storycards), son tarjetas simples en papel donde se describen breves requerimientos (en una o dos líneas), pueden adoptar el esquema de Clase – Responsabilidad – Colaboración (CRC), tienen una granularidad de diez o veinte días. Se utilizan para estimar prioridades, alcance y tiempo de realización. Si existen discrepancias se usa la más optimista. Otros artefactos son: *listas de tareas* en papel o pizarra y *gráficos* visibles en la pared.

Historia de Usuario	
Número: **1**	Nombre: **Ingreso o supresión de Roles**
Usuario: **Administrador**	
Modificación de Historia Número:	Iteración Asignada:
Prioridad en Negocio: **Baja** (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: (Alto / Medio / Bajo)	Puntos Reales:
Descripción: **En la administración del sistema tendrá la opción de administrar usuarios, al ingresar a esta opción se desplegará un listado de los usuarios, los usuarios van a tener la opción de asignar roles, el administrador hace clic sobre esta opción relacionada con el usuario y el sistema le despliega el listado de roles disponibles para que el administrador seleccione los adecuados para ese usuario. Una vez el usuario administrador del sistema de la opción de guardar, el sistema pide confirmación y luego procederá a almacenar los cambios.**	
Observaciones:	

Figura 7. Historia de usuario.

Las tarjetas de historia también pueden denominarse Historias de Usuario (user stories) y tienen el formato como se ve en la Figura 7.Los usuarios describen requerimientos funcionales y no funcionales. Las historias de usuario son descompuestas en tareas de programación (task card) y asignadas a los desarrolladores (Orjuela y Rojas 2008).

XP utiliza YAGNI (You Aren't Gonna Need It, no vas a necesitarlo) para el diseño simple con la idea de no hacer nada que no se necesite ahora, o algo que vaya a necesitarse más adelante, minimizando diagramas y documentos. Gottesdiener (2007) sugiere implementarlo en el documento de requerimientos expresando**no significa que no documentaremos los requerimientos porque no lo vamos a necesitar o usar nunca** pero vamos a prestar más

atención en lo que documentaremos y como. Y agrega como deberíamos ajustar la forma y el formato de nuestro documento:

- *Forma*: La forma de los requerimientos del negocio puede ser una lista de metas y objetivos. Las necesidades de los usuarios puede tomar la forma de escenarios, reglas de negocio y un modelo de datos. La forma de los atributos de calidad pueden ser declaraciones de texto, un subconjunto de "Planguage" (Lenguaje de Planificación) elementos, o escenarios de calidad de los atributos.
- *Formato*: El formato de los escenarios podría ser las declaraciones escritas a mano en tarjetas o escenarios con plantilla registrados en un documento de Excel o Word. El formato de su modelo de datos puede ser un borrador dibujado en un cartel o pizarra, o un diagrama dibujado en una herramienta como Visio. Si usted escribe sus atributos de calidad que utilizan Planguage, puede darles formato en un documento de Word que se remonta a los casos de prueba en una herramienta de gestión de requisitos.

2.2.5 SCRUM

La metodología fue desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. El término *scrum* proviene del rugby que define al acto de preparar el avance del equipo en unidad pasando la pelota a uno u otro jugador. Tal como el deporte, **Scrum es adaptativo, ágil, auto-organizante y con pocos tiempos muertos** (Reynoso, 2004).

Scrum no está concebido para ser utilizado independientemente, sino en combinación con otras metodologías. Se enfoca en valores y prácticas de gestión, sin mencionar requerimientos, implementación u otros temas técnicos. Utiliza técnicas de control de procesos que aplica en gestión y control de proyectos.

Los *valores* son:

- *Equipos auto-dirigidos y auto-organizados*. No hay un administrador que decida, hay miembros del equipo o cerdos, los observadores externos se denominan gallinas.
- *Elegida una tarea no se agrega trabajo extra*. Si se agrega quitar otra cosa.
- *Encuentros diarios de 15 minutos* para responder a las siguientes tres preguntas: ¿qué has hecho desde el último encuentro? ¿qué obstáculos hay para cumplir la meta? ¿qué harás antes del próximo encuentro?
- *Iteraciones de treinta días*. Pueden ser más frecuentes.
- *Demostración a participantesexternos al finalizar cada iteración*.
- *Al principio de cada iteración, el cliente guía el planeamiento adaptativo*.

El *proceso* de Scrum se compone de cuatro fases:

- *Pre-Juego → Planeamiento*. Se establece la visión, las expectativas y se asegura la financiación. Las actividades son: escritura de la visión, el presupuesto, el sprint backlog del producto inicial (de alto nivel de abstracción) y los ítems estimados, también la arquitectura de alto nivel, el diseño exploratorio y prototipos.
- *Pre-Juego → Montaje*. Se identifican más requerimientos y priorizan tareas para la primera iteración. Las actividades son planificación, diseño exploratorio y prototipos.
- *Juego o Desarrollo*. Se implementa un sistema listo para entrega en una serie de iteraciones de treinta días (sprints), Las actividades son un encuentro de planeamiento

de sprints en cada iteración, la definición del product backlog, los estimados y los encuentros diarios.

- *Post-Juego → Liberación*. Se despliega operacionalmente el producto. Las actividades son documentación, entrenamiento, mercadeo y venta.

Un sprint consiste en (ver Figura 8):

1. *Planificación del sprint.* Se define el product backlog, si no ha sido definido, consiste en una lista priorizada de requerimientos que puede ser modificado continuamente. En cada iteración (sprint) el product backlog es revisado por el equipo. Se planifica el primer sprint. En la reunión al inicio del sprint se obtiene el sprint backlog que contiene un listado de tareas y el objetivo principal del sprint.
2. *Seguimiento del sprint.*Se hacen reuniones diarias para conocer el avance de las tareas y el trabajo previsto para la jornada.
3. *Revisión del sprint.* Terminado el sprint se realiza análisis y revisión del incremento generado. En esta reunión se presentan resultados finales, sugiriendo presentar un prototipo.

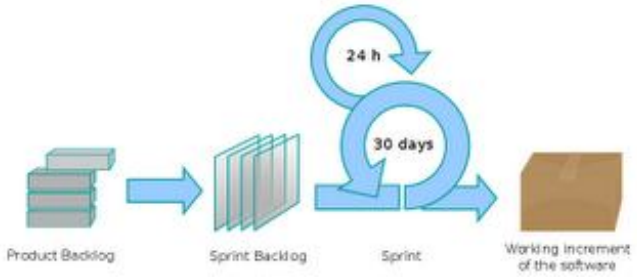

Figura 8. Ciclo del sprint.

Los *roles* en Scrum son seis:

- *Scrum Master*. Debe interactuar con el equipo, el cliente y los gestores. Garantiza el funcionamiento de los procesos y la metodología. Debe ser miembro del equipo y trabajar a la par. Coordina los encuentros diarios y se encarga de eliminar obstáculos.
- *Propietario del producto*. Es el responsable oficial del proyecto, gestión, control y visibilidad del product backlog. Elegido por el Scrum Master, el cliente y los ejecutivos.
- *Equipo de desarrollo*. Tiene autoridad para reorganizarse y definir acciones necesarias o sugerir eliminar problemas para cumplir con los objetivos del sprint.
- *El cliente*.Participa en la creación del product backlog.
- *El gestor*. Toma las decisiones finales, participa en la elección de objetivos y requisitos. Selecciona el Propietario del producto.
- *El usuario.*

Scrum no requiere o provee ninguna práctica específica, dispone de prácticas y herramientas para la gestión de sus diferentes fases. Las principales *prácticas* son:

- *Product Backlog*. Define los requerimientos del sistema o el trabajo a realizar a lo largo del proyecto. Lo mantiene el propietario del producto.
- *Sprint Backlog*. Es un listado de tareas, personas, tiempos y recursos que el equipo se compromete a realizar para generar el incremento previsto.

- *Estimación de esfuerzos*.Es un proceso iterativo donde las estimaciones del product backlog son ajustadas de acuerdo a la última iteración. Lo hace el equipo de desarrollo y el propietario del producto.
- *Gráfico Burn-down*. Es una herramienta para gestionar y seguir el trabajo de cada sprint y representa gráficamente el avance del mismo.
- *Gráfico Burn-up*. Es una herramienta que permite al propietario del producto gestionar y seguir las versiones previstas del producto, funcionalidades, velocidad estimada, fechas probables de cada versión, margen de error previsto y avance real.
- *Planning Poker*. Es un juego para ayudar al equipo a definir una estimación de tareas en la reunión inicial del sprint.

2.2.5.1 MODELADO DE REQUERIMIENTOS Y SCRUM

Como lo mencionáramos anteriormente, Scrum no se ocupa de los detalles relacionados con el modelado de requerimientos, pero claramente se puede leer que el artefacto denominado product backlog (ver Tabla 2) recoge los requerimientos del sistema o las necesidades de los clientes, así como las estimaciones.

Product Blacklog	Fecha:
	Estimado:
Tipo: Nuevo ____ Mejora ____ Arreglo ____	Fuente:
Descripción:	
Notas:	

Tabla 2. Product Backlog.

Los desarrolladores no pueden implementar los requerimientos si no están dentro del Product Backlog. Usando los casos de prueba como una medida absoluta ya que antes que el desarrollador distribuya el software trabajado, los casos de prueba deben ser generados para poder implementar los requerimientos.

En cada iteración, el equipo de desarrollo comienza con la lista priorizada de requerimientos almacenada en el Sprint Backlog. Luego, cuando los requerimientos surgen y evolucionan debido al entorno dinámico de negocios o una solicitud de cambio de las partes interesadas, la modificación de las prioridades de los requerimientos en el product blacklog se produce sobre la base actualdel valor de los requerimientos de negocio (Oyeyipo y Mueller, 2011).

2.2.6 TEST DRIVEN DEVELOPMENT

El Desarrollo Orientado a Pruebas (TDD) es una técnica que utiliza la metodología Extreme Programming (una de las buenas prácticas de Agile Modeling, Ambler 2005), podría considerarse como metodología porque posee un conjunto de prácticas y métodos de desarrollo (Carvajal, 2008).

TDD permite mantener un conjunto de pruebas, ningún código pasa a producción sino pasa los juegos de prueba asociados, se escriben primero las pruebas que determinan el código que se necesita escribir. Los *principios* de TDD son:

- *Mantener un juego exhaustivo de pruebas del programador*. Las pruebas del programador prueban que las clases se comporten como deben, similares a las pruebas unitarias. No se puede hacer el código si no existe la prueba.
- *Todo código que pasa a producción tiene su prueba asociada*. Una funcionalidad no existe hasta que no exista una prueba asociada a él.
- *Escribir las pruebas primero*. Antes de implementar escribir las pruebas.
- *Las pruebas determinan el código a escribir*. Se escribe lo necesario para que la prueba funcione.

TDD se engloba en la metodología XP, en las fases de iteración, producción y mantenimiento, pero puede ser aplicada independientemente de XP.

Los *roles* que podemos encontrar son:

- *Cliente*. Desarrolla las historias con los desarrolladores, prioriza y escribe pruebas funcionales.
- *Desarrollador*. Desarrolla historias con el usuario, estima y utiliza TDD para desarrollarlas.

Las *prácticas* que se mencionan forman parte de XP pero están muy vinculadas a TDD:

- *Refactorización*. Es el proceso de aplicar cambios a un código existente y funcional sin modificar comportamiento externo, con el propósito de mejorar su estructura interna y aumentar legibilidad y entendimiento. En TDD muchas veces duplicamos código o no dejamos un código muy claro por tratar de hacer que pase las pruebas.
- *Integración continua*.La integración a la acción incluye una porción de código a la totalidad del código. Cada integración es verificada con un juego exhaustivo de pruebas que se ejecutan automáticamente.

2.2.6.1 MODELADO DE REQUERIMIENTOS Y TDD

TDD no contempla ni soporte de gestión para proyecto, ni procesos, prácticas, actividades o artefactos para las primeras fases del ciclo de vida de desarrollo de software.

2.3 DISCUSIÓN

No hay duda que las metodologías ágiles confrontan a las metodologías consagradas, consensuadas entre organismos y apreciadas por consultores, analistas de industria y corporaciones. Sus defensores se han expresado mediante manifiestos, proclamas, dejando de lado especificaciones formales.Las metodologías ágiles tienen en común el modelo de desarrollo incremental (pequeñas entregas con ciclos rápidos), cooperativo - colaborativo (desarrolladores y usuarios trabajan juntos en estrecha comunicación), directo (el método es simple y fácil de aprender) y adaptativo (capaz de incorporar los cambios) (Reynoso, 2004).

Algunos nombres respetados como (Fowler, Hock, Krutchen, Jacobson) consideran que estas metodologías no se deberían dejar de lado.

Reynoso (2004) declara que entre las metodologías ágiles existentes hay dos subgrupos uno es auto-organizativo, anárquico, igualitario o emergente de abajo hacia arriba; el otro es la expresión de una disciplina de gestión, innovadora, pero más formal que transgresora y más rigurosa que temeraria.

XP parece no necesitar ninguna otra herramienta fuera de las que utiliza en la programación y la prueba. Sus obstáculos tienen que ver con la creencia que el cliente permanecerá en el lugar de desarrollo siempre que se lo necesite y la resistencia de los programadores en trabajar en pares. Larman (2004) manifiesta que no existe un énfasis en arquitectura durante las iteraciones iniciales ni métodos de diseño arquitectónico. XP ha sido el que más resistencia ha encontrado.

3. CONCLUSIONES

Podemos concluir que todas las metodologías analizadas en este trabajo se pueden combinar con otras sean tradicionales o más actuales, algunas como el caso de TDD forman parte de otras. Las combinaciones pueden darse en algunas o en todas las fases para poder lograr cubrir todo el ciclo de vida. La Figura 9 contiene una comparación de las metodologías descriptas en este trabajo en cuanto a las fases o etapas que cada una cubre del ciclo de vida básico del software (Carvajal 2008). Algunas de las conclusiones son:

- Las metodologías que cubren la etapa de análisis con soporte para la gestión, prácticas, actividades y artefactos son APM y Scrum.
- La metodología que cubre gestión y describe un proceso pero no prescribe la forma de utilizar prácticas, actividades y artefactos en las primeras etapas es DSDM.
- Las metodologías que no cubren ni el inicio del proyecto ni la etapa de análisis son Crystal Methods y TDD.
- La metodología que posee un proceso con prácticas, actividades y artefactos en la etapa de análisis es XP.

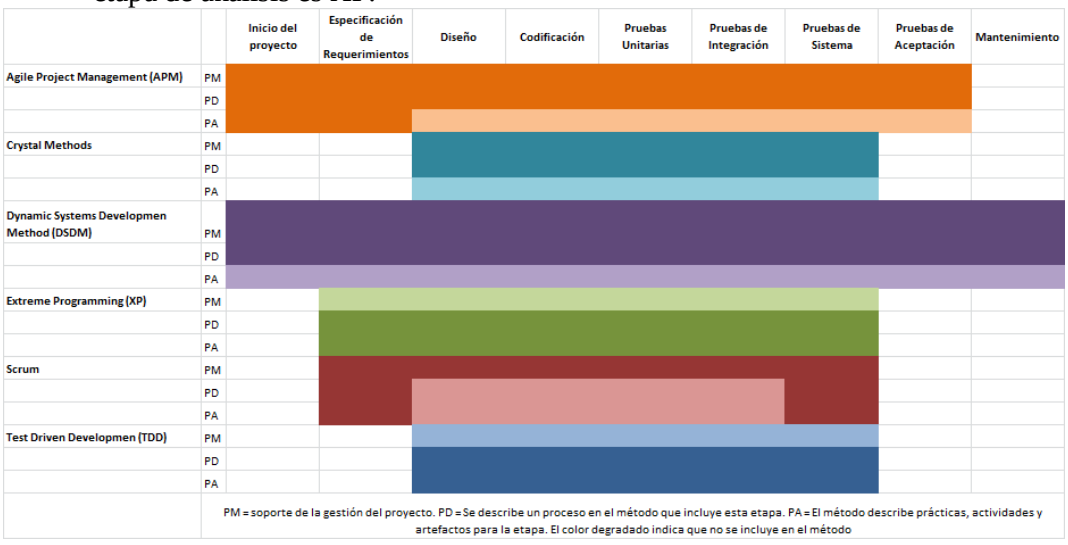

Figura 9. Comparación entre metodologías ágiles según etapa cubierta.

Gottesdiener (2007) entiende que el desarrollo ágil se basa en buenas prácticas y que una serie de ellas pueden aplicarse a los requerimientos, tales como:

- *Asegurar la participación del cliente*. La participación temprana y continua de los clientes no es nueva, se aplica desde el "abuelo" de los métodos ágiles como es DSDM cuyo primer principio es "la participación del usuario activo es imprescindible".
- *Gestionar los cambios en los requerimientos*. Los proyectos ágiles manejan el cambio a través del uso combinado del product backlog y las iteraciones.
- *Iterar-Inspeccionar-Adaptar*. Las iteraciones se realizan en periodos fijos de tiempo, en ellas el cliente selecciona con el equipo los requerimientos a desarrollar, el equipo en un taller de planificación estiman las tareas y el tiempo, se inspecciona la demo del producto construido al fin de cada iteración, luego el equipo adapta su proceso mediante una sesión retrospectiva poniendo fin a la iteración con el siguiente inicio de iteración en mente.
- *Alrededor del juego*. Construir prototipos y demostraciones. Durante una demostración los desarrolladores muestran el funcionamientodel software por cada historia. A veces puede generar nuevas historias. Sumado a los prototipos, se podría complementar las historias con guiones gráficos (utilizando pizarras, posters, post it), reglas de negocio y pruebas de aceptación de usuario.
- *Talleres de colaboración*. Tienen sus orígenes en los Joint Application Design/Development (JAD[15]en la década del 80) y en DSDM. Se pueden utilizar para definir una hoja de ruta de un producto, para planear entregas e iteraciones, conducir retrospectivas de iteraciones.

Aquellas metodologías que cubren las etapas de requerimientos, suelen hacerlo en una o dos fases, en las práctica para obtener una visión general del sistema trabajan con los requerimientos de más alto nivel, tanto funcionales como no funcionales, a través del consenso entre el equipo de desarrollo y los clientes; y en la especificación logran la especificación de requerimientos generales, refinándolos concurrentemente hasta llevarlos a especificaciones más detalladas.

Este trabajo es un estudio preliminar, por lo que es necesario profundizar en cada una de las metodologías, en especial en las etapas de análisis y diseño que interesan al proyecto.

4. AGRADECIMIENTOS

Agradezco a mi marido Cristian por su constante apoyo, a mis hijos Lucas, Ian Marco y Fabrizzio por su paciencia, a mis padres por dejarme estudiar, a la Lic. Gabriela Vilanova por ser parte de su proyecto de investigación y comenzar a trabajar en el área que deseo, y a la Universidad Nacional de la Patagonia Austral a la que pertenezco con mucho orgullo y, por supuesto, financia el proyecto del que soy parte.

[15] Desarrollo/Diseño Conjunto de Aplicaciones.

5. REFERENCIAS

Abrahamsson P., Salo O., Ronkainen J., Warsta J. 2002. Agile software development methods. Review and analysis. VTT. Finlandia.

Ambler S. 2005. Effective Practices for Modeling and Documentation. *En*: http://www.agilemodeling.com/

Beck, K., et al.: The Agile Manifesto. Manifesto for Agile Software Development. *En*: http://www.agilemanifesto.org

Boehm, B., Turner, R. 2004."Balancing Agility and Discipline. A Guide for the Perplexed" ISBN 0321186125 Addison-Wesley.

Boehm, B. 2006.A View of 20th and 21st Century Software Engineering. In: 28th international conference on Software engineering, pp. 12--29. Shanghai, China.

Booch, G, Rumbaugh, J., Jacobson, I. 2006. El lenguaje unificado de modelado. Pearson Educación. España.

Canós J., Letelier P., Penadés M. 2003. Metodologías Ágiles en el Desarrollo de Software. JISBD 2003. España.

Cao, L., Ramesh, B. 2008. Agile Requirements Engineering Practices: An Empirical Study. In: IEEE Computer Society Press Los Alamitos, CA, vol 25, pp-60-67.

Carvajal J. 2008. Metodologías Ágiles: Herramientas y Modelo de desarrollo para aplicaciones Java EE como metodología empresarial.

Castro, P. 2004. Ingeniería del software orientada a objetos. *En*: Revista del Instituto Tecnológico de Informática Actualidad TIC. Nro 5. Pp. 8 - 11. España.

Cockburn, A. 2002. 7 properties of highly successful projects from Crystal Clear. *En*: http://alistair.cockburn.us/.

DSDM Consortium. 2002. DSDM and Extreme Programming. *En*: http://www.agilealliance.org/resources/articles/

Gottesdiener, E. 2007.YAGNI your requirements docs.Success with Requirements, Vol. 1, No. 9.

Gottesdiener E. 2007. "Requirements Practices on Agile Projects," Successwith Requirements, Vol. 1, No. 8.

Highsmith, J. 2004. Agile Project Management: creating innovative products. Pearson Education. USA.

IEEE.1990.IEEE Standard Glossary of Software Engineering Terminology. *En*: http://segoldmine.ppi-int.com/content/standard-ieee-std-610-ieee-standard-glossary-software-engineering-terminology.

Jackson, M. 1997.The meaning of requirements.Annals of Software Engineering 3. Pp. 5 – 21.

Kruchten, P. 2000.TheRationalUnifiedProcess: An introduction. Addison-Wesley.

Laboratorio Nacional de Calidad de Software. 2009. Ingeniería del Software: Metodologías y Ciclos de Vida. INTECO. España.

Larman, C. 2003. UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Pearson Educación. España.

Larman.C. 2004.Agile & Iterative Development. Addison Wesley.

Orjuela A., Rojas M. 2008. Las metodologías de desarrollo ágil como una oportunidad para la Ingeniería de Software Educativo. *En*: Revista Avances en Sistemas e Informática. Vol. 5. Nro. 2. Colombia.

Oyeyipo E., Mueller C. 2011. Requirements Management in Agile-Scrum. Texas State University. San Marcos.

Pérez, S., Jaime, J. 2009. Procedimiento para la obtención de requerimientos funcionales a partir del análisis de negocios.

Pressman, R. 2002. Ingeniería del Software. Un enfoque práctico. Mc Graw Hill Interamericana de España.

Reynoso, C. 2004. Métodos Heterodoxos en Desarrollo de Software. UBA. Argentina.

Royce, W. 1970. Managing the Development of Large Software Systems, Proceedings of IEEE Westcon.

Sommerville, I. 2005. Ingeniería de Software. Pearson Educación. España.

SWEBOK. 2004. Guide to the Software Engineering Body of Knowledge. *En*: http://www.computer.org/portal/web/swebok/htmlformat.

Tudor D., Walter G. 2006. Using an Agile Approach in a Large, Traditional Organization. Agile 2006 Conference. IEEE Computer Society.

Tuffs, D., Stapleton, J., West, D., Eason, Z. 1999.Inter-operability of DSDM with the Rational Unified Process. DSDM Consortium.

Urquiza J., Martínez A., Ibargüengoitia G. 2010. Las metodologías ágiles y las arquitecturas de software. Coloquio Nacional de Investigación en Ingeniería de Software y Vinculación Academia-Industria.

Lindvall, et al. 2002. Empirical Findings in Agile Methods. In: Lecture Notes In Computer Science; Vol. 2418. Proc. of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe. Berlin, Germany

Nerur, S., Mahapatra, R., Mangalaraj, G. 2005. Challenges of migrating to agile methodologies. In: Commun. ACM, vol. 48, no. 5, pp. 72-78.