Universidad de las Ciencias Informáticas
Facultad 3

Título: "Método Cuantitativo de Evaluación de Metodologías Ágiles."

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autores:

Jolfri Rodríguez Legrá

Reynold Salmon Trujillo

Tutor:

Ing. Rolando Pérez Pinto

Ciudad Habana, Junio 2008

"Confirmar es crear. Lo que hace crecer el mundo no es el descubrir cómo está hecho sino el esfuerzo que cada uno hace para descubrirlo."
José Martí.

DECLARACIÓN DE AUTORÍA

Declaramos ser los autores de la presente tesis y concedemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____
Jolfri Rodríguez Legrá

Autor

_____
Reynold Salmon Trujillo

Autor

_____

Ing. Rolando Pérez Pinto

Tutor

AGRADECIMIENTOS

Quisiera agradecer a las siguientes personas que han contribuido de una u otra forma durante la elaboración de la tesis. Mis agradecimientos son para: Rolando Pérez Pinto, Carlos Matos Carbonell y Pedro Piñeiro Pérez,

Finalmente quisiera agradecer a mi madre y a mi novia. Así mismo hago extensivo el agradecimiento a toda mi familia y demás personas que olvido en este momento mencionar.

Jolfri

Agradezco

A mi madre que es lo más grande que tengo por su apoyo, sus consejos, su confianza. A Yoel que me forjo y guío siempre por el buen camino y nunca me dio la espalda… A esos tíos, tías y primos que se han preocupado por mí, que me han brindado todo su apoyo, especialmente a Brinci, Marylin y Rafael… A mi abuela… A mi Dino y su familia por demostrarme en todos estos años que soy parte de ellos… A Maylen, por toda la ayuda y el apoyo que me brindó. A Yami por haberme ayudado tanto, Ana, Susan y Dania. A los que han sido mis hermanos dentro y fuera de la escuela. A Lainer y Estrella que han sabido ser compañeros y amigos en estos 5 años… A mi tutor y compañeros de estudio que contribuyeron en la realización de esta tesis… A la revolución y en especial a Fidel por creer tanto en nosotros…

De forma general:

A todas las personas que de una forma u otra me ayudaron no solo en la realización de esta tesis sino también en el transcurso de mi carrera profesional y deportiva, profesores, familia en general, amigos y personas allegadas especialmente a Gregorio mi entrenador por enseñarme a ser un hombre sencillo y honesto a el debo mucho de lo que hoy soy.

Reynold

DEDICATORIA

El resultado de la presente tesis esta dedicado a la grandiosa obra de La Revolución Cubana y en especial a nuestro comandante en jefe por tener la ingeniosa idea de crear esta universidad.

Jolfri

El resultado de la presente tesis está dedicado:

Especialmente

A mi madre por ser mi máxima inspiración…

A nuestro líder y guía Fidel Castro Ruz por crear este proyecto que nos permitió no solo formarnos como profesionales de ciencias sino también como hombres revolucionarios…

De forma general

A familiares, amigos, profesores y todas las personas que estuvieron cerca durante estos cinco años…

Reynold

Resumen

EL desarrollo de las Metodologías Ágiles, ha venido a tratar de resolver una parte de los problemas relacionados con La Ingeniería de Software. Esto ha conllevado a la creación de varios procesos de evaluación de metodologías de desarrollo de software pero en su gran mayoría realizan una evaluación de forma subjetiva dejando la evaluación de aspectos de gran importancia de los procedimientos metodológicos a consideraciones puntuales de los autores de la evaluación. Por lo que realizar un método para la evaluación cuantitativa de las Metodologías Ágiles es el objetivo de la tesis.

En la investigación se realiza una caracterización de las Metodologías Ágiles más conocidas a nivel internacional, el análisis de un framework enfocado en la determinación del grado de agilidad de Metodologías Ágiles. Así como la definición de 5 dimensiones que involucran el proceso de un software, con el objetivo de cuantificar los elementos definidos para el desarrollo de un software de forma ágil.

El Método de Evaluación de Metodologías Ágiles obtenido en la presente tesis, facilita la obtención del grado de agilidad de los procedimientos metodológicos al que le sea aplicado y a partir de aquí se pueden establecer consideraciones para tener en cuenta una determinada metodología a la hora de realizar un software, ya que son obtenidos valores concretos relacionados con la velocidad de desarrollo en las dimensiones definidas.

Palabras Claves

Metodologías Ágiles, Método de Evaluación, Proceso de Software.

TABLA DE CONTENIDOS

AGRADECIMIENTOS I

DEDICATORIA II

Resumen III

Introducción 1

1. Capítulo 1: Fundamentación Teórica 4

1.1. Introducción 4

1.2. Historia de las Metodologías Ágiles 4

1.2.1. Manifiesto Ágil 5

1.3. ¿Que son las Metodologías Ágiles? 9

1.4. Características de las Metodologías Ágiles 9

1.5. Empresas que usan Metodologías Ágiles 9

1.6. Principales Metodologías Ágiles 10

1.6.1. XP (Programación Extrema) 11

1.6.2. SCRUM 15

1.6.3. ASD (Desarrollo de Software Adaptativo) 18

1.6.4. Crystal Methodologies 20

1.6.5. DSDM (Método de Desarrollo de Sistemas Dinámicos) 23

1.6.6. FDD (Desarrollo Conducido por Rasgos) 26

1.6.7. MSF (Marco de Trabajo de Soluciones de Microsoft) 29

1.7. Estado Actual de los Procesos de Evaluación de Metodologías de Desarrollo de Software. 33

1.8. Framework de Evaluación de Metodologías de Desarrollo de Software. 34

1.8.1. 4-DAT 34

1.9. Marco de desarrollo nacional 35

2. Capítulo 2: Desarrollo del método cuantitativo para la evaluación de Metodologías Ágiles. 37

2.1. Introducción 37

2.2. Dimensiones 37

2.3. Cálculo Cuantitativo del Grado de Agilidad. 38

2.4. Descripción para la obtención del valor cuantitativo en el alcance de la metodología. 39

2.4.1. Descripción para la obtención del coeficiente de completitud para la etapa de análisis. 41

2.4.2. Descripción para la obtención del coeficiente de completitud para la etapa de diseño. 43

2.4.3. Descripción para la obtención del coeficiente de completitud para la etapa de implementación 44
2.4.4. Descripción para la obtención del coeficiente de completitud para la etapa de pruebas. 45
2.5. Descripción para la obtención del valor cuantitativo en los parámetros relacionados con la agilidad de las metodologías ágiles en las distintas etapas del software 47
2.6. Descripción para la obtención del valor cuantitativo en los valores ágiles 53
2.7. Descripción para la obtención del valor cuantitativo en la caracterización del proceso de software 56
2.8. Descripción para la obtención del valor cuantitativo en la definición de roles 58
3. Capítulo 3: Aplicación del método cuantitativo para la evaluación de Metodologías Ágiles. 60
3.1. Introducción 60
3.2. Aplicación del Método a las Principales Metodologías Ágiles 60
3.2.1. Aplicación del Método a XP (Programación Extrema) 60
3.2.2. Aplicación del Método a DSDM (Método de Desarrollo de Sistemas Dinámicos) 70
3.2.3. Aplicación del Método a SCRUM 78
3.2.4. Aplicación del Método a RUP (Proceso Unificado del Rational). 84
3.2.5. Límites del Método Cuantitativo para La Evaluación de Metodologías Ágiles 91
3.2.6. Resultados de la aplicación del método 91
3.3. Valoración de los resultados de la aplicación del método. 92
CONCLUSIONES 94
RECOMENDACIONES 95
REFERENCIA BIBLIOGRAFICA 96
BIBLIOGRAFIA ONLINE 98
ANEXOS 99
GLOSARIO DE TERMINOS 106

INTRODUCCIÓN

Cuando se dispone a realizar un proceso de desarrollo de software es de vital importancia reconocer que este puede ser muy riesgoso y a la vez difícil de controlar, por tales razones es importante llevar a cavo una metodología de desarrollo de software, que en caso de no seguir los procedimientos que establecen estas; estos problemas serán catastróficos y lo que se obtendría como resultado final es clientes insatisfechos y sobre todo desarrollares diseccionados.

Una incógnita sobre sale en los inicios de desarrollo de un determinado producto y no es más que: ¿Cuál sería la metodología de desarrollo de software adecuada para desarrollar un producto de software partiendo del grado de agilidad del procedimiento? Aunque resulta algo simple, la respuesta no lo es, ya que, el proceso de desarrollo de software no es una tarea fácil, siendo centro de atención en los últimos años la solución más factible y rentable. Muestra de ello es la existencia de varias propuestas metodológicas que inciden en las distintas y variadas aristas del proceso de software. Por un lado se encuentran aquellas que son llamadas tradicionales o pesadas que se centran específicamente en el control del proceso, definiendo rigurosamente las actividades involucradas, los artefactos que se deben obtener, las herramientas y notaciones que se usarán.

Estas metodologías han demostrado en gran medida ser muy efectivas y necesarias en la inmensa mayoría de los proyectos, pero también han sido parte del fracaso en muchos sistemas en las que han sido aplicadas. Por otro lado existen las Metodologías Ágiles que en esencia su filosofía consiste en el factor humano y en el producto como tal, es importante valorar la importancia que le dan estas a los desarrolladores, a la colaboración constante con el cliente y al desarrollo incremental del software con iteraciones cortas.

Este tema se impone en la actualidad, tal es el caso que se ha despertado gran interés en la mayoría de los ingenieros informáticos, en los profesores vinculados con el proceso docente relacionado con la ingeniería de software e incluso en los estudiantes que están vinculados a la producción. Ya que para muchos las metodologías tradicionales les resultan muy distintas a su estilo de trabajo, por las dificultades a la hora de introducirlas en un determinado software e incluso por la inversión que lleva asociada en formación y en herramientas.

Sin embargo las Metodologías Ágiles han sido diseñadas para ser aplicadas a un amplio rango de procesos industriales de desarrollo de software, específicamente aquellos en los cuales su equipo de desarrollo es pequeño, a los que posean plazos reducidos para la entrega del producto, los de

requisitos volátiles y los de uso de nuevas tecnologías. Sin dudas esto último está abriendo atrayentes canales en la producción actual de software, ya que muchos de los proyectos que se desarrollan tienen características semejantes a esos para los cuales han sido fundamentalmente creadas las Metodologías Ágiles.

El surgimiento de las Metodologías Ágiles también ha creado una expectativa un poco interesante desde el punto de vista de la creación de algunos framework para la evaluación de estas, que guían un poco la filosofía de estos enfoques ágiles a la hora de tenerlos en cuenta para su posible aplicación en un determinando software que se valla a realizar, a partir de determinados aspectos relacionados con el ciclo de vida del software.

Diseño Teórico de la Investigación

Situación Problemática:

Los procesos de evaluación de Metodologías Ágiles que son referenciados en la literatura se presentan en su mayoría de forma subjetiva, dejando la evaluación de aspectos de gran importancia de la metodología a consideraciones puntuales de los autores de la evaluación, lo cual interfiere en el grado de confiabilidad y precisión de los mismos.

Problema:

¿Cómo evaluar de forma cuantitativa una metodología ágil, considerando las etapas básicas de análisis, diseño, implementación y pruebas del ciclo de vida de un software?

Objeto de estudio:

Las Metodologías de Desarrollo de Software.

Campo de acción

La evaluación de Metodologías Ágiles de Desarrollo de Software.

Objetivo general:

Proponer un método que evalué de forma cuantitativa las Metodologías Ágiles.

Hipótesis:

Si se realiza un método que evalué de forma cuantitativa Las Metodologías Ágiles de Desarrollo de Software, será más fácil saber cuan ágiles son estas.

Tareas investigativas:

Estudio de las Metodologías Ágiles de Desarrollo de Software.

Estudio de Framework relacionados con la evaluación de Metodologías Ágiles.

Descripción de las etapas básicas para la construcción de software.

Establecimiento de las dimensiones para realizar el cálculo cuantitativo de las Metodologías Ágiles.

Definición de parámetros que incluyan indicadores medibles relacionados con la agilidad.

Definición de ecuaciones para calcular el grado de agilidad total de la metodología analizada, y de las dimensiones

Obtención del método cuantitativo para la evaluación de Metodologías Ágiles.

Aplicación del método obtenido a las Metodologías Ágiles más conocidas.

Valoración de la aplicación del método.

Se pretende tener como posibles resultados un método para la evaluación de forma cuantitativa de las Metodologías Ágiles.

La investigación está sustentada en 3 capítulos, estructura que se describe a continuación. En el capítulo 1 se realizó el estado del arte de la investigación, donde se esclarecieron los conceptos básicos sobre las Metodologías Ágiles de más renombre a nivel internacional así como sus características, a demás de un análisis de algunos framework relacionados con la evaluación de Metodologías Ágiles. En el capítulo 2 se realiza un método para la evaluación de forma cuantitativa de las Metodologías Ágiles partiendo de 5 dimensiones definidas que a groso modo describen la trayectoria del proceso de software basado en los enfoques ágiles y por último en el capítulo 3 se le aplica el método a algunos procedimientos ágiles con el fin de determinar el grado de agilidad.

1. CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

La producción de software hoy en día constituye una parte crítica de todo negocio. Sin embargo aún existen problemas en el desarrollo de software como son la calidad insuficiente, los prolongados tiempos en el desarrollo y elevados costos. Con el objetivo de propiciarle una solución a estos problemas se ha optado por mejorar el proceso de desarrollo, o sea la forma en que debe construirse el software, lo que equivale a ser más selectivo a la hora de seleccionar las metodologías adecuadas, partiendo de su grado de agilidad a la hora de realizar las tares esenciales que incluye un proceso de software.

Con el transcurso de los años han surgidos varios enfoques metodológicos pero en ocasiones el contenido del proyecto es impredecible o menos estable de lo que realmente se desea, por tales motivos es de vital importancia valorar a fondo las Metodologías Ágiles, ya que pueden resultar útiles en muchos entornos de la producción de software.

A principios de la década del ’90, surgió un enfoque que fue bastante revolucionado para su momento ya que iba en contra de la creencia de que mediante procesos altamente definidos se iba a lograr obtener software en tiempo, con los costos adecuados y con la requerida calidad. El enfoque fue planteado por primera vez por Martin Flower en 1991 y se dio a conocer en la comunidad de ingeniería de software con el mismo nombre que su libro, RAD o (Rapid Application Development). RAD consistía en un entorno de desarrollo altamente productivo, en el que participaban grupos pequeños de programadores utilizando herramientas que generaban código en forma automática tomando como entradas sintaxis de alto nivel. En general, se considera que este fue uno de los primeros hitos en pos de la agilidad en los procesos de desarrollo de software.

1.2. Historia de las Metodologías Ágiles

En febrero del 2001, a raíz una reunión celebrada en Utah, Estados Unidos, surge el término ágil aplicado al desarrollo de software. En dicha reunión participaron un grupo de expertos de la industria del software, incluyendo a varios de los creadores o impulsores de las metodologías de software. La esencia fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a través del proyecto.

A raíz de esta reunión se creó La Alianza Ágil[1], una organización consagrada a promover los conceptos y la filosofía relacionada con el desarrollo ágil de software para ayudar y guiar a las organizaciones que adopten dichos conceptos. El inicio de toda esta revolución de procedimientos ágiles fue el Manifiesto Ágil[2], un documento que tiene explicito un resumen de las concepciones de los procedimientos ágiles.

1.2.1. Manifiesto Ágil

Propósito: Descubrir mejores modos de desarrollar software haciendo y ayudando a otros hacerlo.

Valorando a través del mismo a:

Los individuos e iteraciones por encima de los procesos y las herramientas.

El software activo por encima de la documentación comprensiva.

La negociación del cliente sobre la negociación del contrato.

La respuesta al cambio sobre el seguimiento de un plan.

Principios

1) La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporten un valor[2].

   La implementación del principio de "valor al cliente "es una de esas actividades que son más fácil de decir que de hacer. Las prácticas de dirección de proyectos tradicionales asumen que lograr un plan equivale al éxito del proyecto y al valor del cliente demostrado. La volatibilidad asociado con esos proyectos de hoy en día, demanda que el valor del cliente constantemente se revalué, y que la satisfacción de los planes del proyecto originales puede no tener que ver mucho con el éxito final de un proyecto.

2) Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva[2].

   La imprevisibilidad creciente del futuro es uno de esos aspectos más desafiantes de la nueva economía. La turbulencia en ambos, negocios y tecnología provoca el cambio, puede verse como una amenaza de la cual cuidarse o como una oportunidad a ser abrazada.

   En lugar de resistirse al cambio, el enfoque ágil se esfuerza por acomodarlo tan fácil y eficazmente como sea posible, mientras se mantiene conciencia de sus consecuencias.

Aunque la mayoría de las personas están de acuerdo con que la retroalimentación es importante, ignoran a menudo el hecho de que el resultado de una retroalimentación aceptada, es el cambio. Las Metodologías Ágiles aprovechan este resultado, porque sus defensores entienden que facilitar el cambio es más eficaz que intentarlo prevenir.

3) Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas [2].

   Durante muchos años los especialistas en procesos han estado diciendo la importancia del uso de un estilo incremental o reiterativo de desarrollo del software, con entregas múltiples de funcionalidad siempre crecientes. Mientras está práctica ha crecido, todavía no es predominante; sin embargo, es esencial para los proyectos ágiles. Además, que se presiona bastante para reducir el tiempo del ciclo de la entrega.

4) La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto[2].

   En los inicios no se espera un conjunto detallado de requisitos que sea visado al principio del proyecto; más bien, se tiene una visión de alto nivel de los requisitos que está sujeta a cambios frecuentes. Claramente, esto no es suficiente para diseñar y codificar, de manera que el hueco se cierra con la iteración frecuente entre las personas del negocio y los desarrolladores. La frecuencia de este contacto sorprende a menudo a las personas. Se pone “diariamente “al principio para dar énfasis al compromiso continuó del cliente de tomar parte activa, y de hecho, tomar responsabilidades en conjunto para el proyecto de software.

5) Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo[2].

   Desplegar todas las herramientas, tecnologías y procesos necesarios, incluso los procesos ágiles, pero al final, son las personas las que diferencia el éxito y el fracaso. Se debe comprender que a pesar de lo duro que se trabaje proponiendo ideas del proceso, lo más que se puede esperar es un efecto de segundo orden en un proyecto. Así que es importante maximizar ese factor de personas de primer orden.

6) El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo[2].

   AL discutir las Metodologías Ágiles, surge el tema de la documentación. Los opositores parecen insultados a veces, mientras se burlan de la falta de documentación. Es suficiente

para gritar, el asunto no es la documentación el asunto es la comprensión“ sí, la documentación física tiene peso y sustancia, pero la medida real de éxito es abstracta: ¿las personas involucradas ganarán el entendimiento que ellos necesitan? Se usa porque tiene que hacerse, pero la mayoría de los equipos de proyecto pueden y deberían usar técnicas de comunicación más directas que la escritura.

De manera que la distinción entre Metodologías Ágiles y documento céntrico no es la de documentación extensa contra ninguna documentación; sino más bien un concepto diferente de la mezcla de documentación y conversación requerida para facilitar el entendimiento.

7) El software que funciona es la medida principal de progreso[2].

A menudo, se ha visto equipos de proyectos que no comprenden que ellos están en problemas sino hasta un tiempo corto antes de la entrega. Ellos hicieron los requisitos a tiempo, el plan a tiempo, quizá incluso el código a tiempo, pero las pruebas y la integración tomó más tiempo de lo que ellos pensaron. Aquí se favorece el desarrollo reiterativo principalmente porque proporciona hitos que no pueden ser diluidos y que imparten una medida exacta del progreso y un entendimiento más profundo de los riesgos involucrados en cualquier proyecto dado.

8) Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante[2].

La industria se caracteriza por noches y fines de semana largos, durante los cuales las personas intentan deshacer los errores de una planificación inerte. Irónicamente, estas largas horas realmente no llevan a una productividad mayor.

La agilidad se basa en personas que están alertas y son creativas, y que puedan mantener esa vigilancia y creatividad a lo largo de todo el proyecto de desarrollo de software. El desarrollo sostenible significa hallar un ritmo de trabajo (40 o algo así, de horas por semana) que el equipo pueda sostener con el tiempo permaneciendo saludable.

9) La atención continua a la calidad técnica y al buen diseño mejora la agilidad[2].

Cuando muchas personas miran al desarrollo ágil, ellos ven recordatorios de los esfuerzos del RAD (Desarrollo de Aplicación Rápido- RAD por sus siglas en inglés.) de la última década. Pero, mientras el desarrollo ágil es similar al RAD por lo que se

refiere a la velocidad y flexibilidad, hay una diferencia grande cuando se refiere a la limpieza técnica. Los enfoques ágiles dan énfasis a la calidad del diseño, porque la calidad del diseño es esencial para mantener la agilidad.

Uno de los aspectos de cuidado, sin embargo, es el hecho que los procesos ágiles asumen y animan la alteración de los requisitos mientras el código está siendo escrito. Como tal, el diseño no puede ser una actividad inicial completamente terminada antes de la construcción. En cambio, el diseño es una actividad continua que se realiza a lo largo del proyecto. Todas y cada una de las iteraciones tendrán un trabajo de diseño.

10) La simplicidad es esencial [2].

Cualquier tarea de desarrollo de software puede ser abordada con una multitud de métodos. En un proyecto ágil, es particularmente importante usar enfoques simples, porque ellos son más fáciles de cambiar. Es más fácil agregar algo a un proceso que es demasiado simple que sacarle algo a un proceso que es demasiado complejo. Hay un sabor fuerte de minimalismo en todos los métodos ágiles. Incluyen solamente lo que todos necesitamos en lugar de lo que alguno necesite, para hacerle más fácil a los equipos agregar algo que satisfaga sus propias necesidades particulares.

11) Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos[2].

Contrariamente a lo que se ha oído, la forma no sigue a la función: La forma sigue al fracaso. "La forma de cosas hechas siempre están sujeta al cambio en respuestas a sus deficiencias o fracasos reales o percibidos para poder funcionar correctamente"[3].

Las visiones de Petroski son similares a uno de los dos puntos claves de este principio , que los mejores diseños (arquitecturas, requerimientos) surgen del desarrollo y uso iterativo en lugar de los planes previos. El segundo punto del principio es que las propiedades emergentes (emergencia, una propiedad de sistemas complejos, dificultosamente se traduce a innovación y creatividad en las organizaciones humanas) se generan mejor en equipos auto organizados en que las iteraciones son altas y las reglas del proceso son pocas.

12) En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto, ajusta su comportamiento[2].

> Los métodos ágiles no son algo que se escoge y sigue servilmente. Puede empezar con uno de estos procesos, pero se reconoce que no se puede proponer el proceso que será correcto para toda situación. Así que cualquier equipo ágil debe refinar y reflexionar conforme a su avanzada, mientras mejora constantemente sus práctica locales.

1.3. ¿Que son las Metodologías Ágiles?

Las Metodologías Ágiles se entienden como un conjunto de metodologías para el desarrollo, surgidas en la década de los 90. Lo ágil se define como la habilidad de responder de forma versátil al cambio para lograr mayores beneficios. Así como que sus valores y principios fueron enunciados en el Manifiesto Ágil por La Alianza Ágil antes mencionada.

Las Metodología Ágiles o "ligeras" constituyen un nuevo enfoque en el desarrollo de software, mejor aceptado por los desarrolladores de proyectos que las metodologías convencionales (ISO-900, CMM, etc.) debido a la simplicidad de sus reglas y prácticas, su orientación a equipos de desarrollo pequeño, su flexibilidad ante los cambios y su ideología de colaboración[4].

1.4. Características de las Metodologías Ágiles

- Basadas en heurísticas provenientes de prácticas de producción de código.
- Especialmente diseñadas para cambios imprevistos durante el desarrollo del proyecto.
- Impuesta internamente (Por el equipo).
- Constan de procesos menos controlados, con pocos principios.
- En ellas no existe contrato tradicional o al menos es bastante flexible.
- El cliente es parte del equipo de desarrollo.
- Definen pequeños grupos de trabajo y trabajando en el mismo sitio.
- Obtienen pocos artefactos.
- Definen pocos roles.
- Realizan menos énfasis en la arquitectura del software.

1.5. Empresas que usan Metodologías Ágiles

En una de las últimas declaraciones del gigante en la producción de software" Microsoft" ha dado a conocer a través de uno de sus ejecutivo, que están promoviendo el uso interno de Scrum y de ciertas prácticas de XP, como la programación en parejas, con el fin de lograr entregar software de calidad a tiempo y en sintonía con los requisitos de los clientes. Lo cierto es que otras grandes empresas están introduciendo métodos ágiles entre sus prácticas de desarrollo. Google incluye conocimiento de las mismas como deseable para los que soliciten trabajo en ella. Por otro lado en un reciente congreso de SCRUM (SCRUM Catering 2005)[5] se explicaba la introducción de SCRUM en Yahoo. En este mismo congreso se menciona a otras empresas como Sun, Siemens, State Farm, IBM y Federal Reserve Bank que en concreto usan Scrum. Otro ejemplo de uso estos procedimientos ágiles es en Inglaterra donde se estima que 1 de cada 5 desarrolladores utiliza DSDM y que más de 500 empresas de desarrollo de software lo han adoptado como vía para el desarrollo de sus aplicaciones.

1.6. Principales Metodologías Ágiles

Entre las Metodologías Ágiles más destacadas hasta el momento podemos nombrar:

- XP (Programación Extrema)
- SCRUM
- Crystal Clear
- DSDM (Método Dinámico de Desarrollo de Sistemas)
- MSF (Microsoft Solution Framework)
- FDD (Desarrollo Conducido por Rasgos)
- ASD (Desarrollo de Software Adaptativo)

Las metodologías relacionadas, a continuación serán caracterizadas teniendo en cuenta los siguientes aspectos:

- Características Principales
- Roles
- Ciclo de desarrollo o proceso
- Prácticas

1.6.1. XP (Programación Extrema)

Fue creada por Kent Beck en el año 1999. XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y esfuerzo para enfrentar los cambios. XP se define especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico[6].

Roles

- Programador. El programador escribe las pruebas unitarias y produce el código del sistema.
- 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.
- Encargado de pruebas (Tester). 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.
- Encargado de seguimiento (Tracker). Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.
- Entrenador (Coach). Es responsable del proceso global. Debe proveer guías al 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, en el que puedan surgir problemas.
- Gestor (Big boss). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

Proceso XP

El ciclo de desarrollo consiste (a grandes rasgos) en:

1) El cliente define el valor de negocio a implementar.

2) El programador estima el esfuerzo necesario para su implementación.
3) El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4) El programador construye el valor de negocio.

El ciclo de vida ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega

(Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

Exploración**:** 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: 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 de usuarios las 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 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**:** 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**:** 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.

Mantenimiento**:** 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.

Muerte del Proyecto**:** Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

Prácticas XP

El juego de la planificación. Hay una comunicación frecuente 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.

Entregas pequeñas: Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema.

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 actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

Diseño simple: Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

Pruebas: La producción de código está dirigida por las pruebas unitarias. Estas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

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.

Programación en parejas: Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño y mayor satisfacción de los programadores).

Propiedad colectiva del código: Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

Integración continúa: 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.

Cliente in-situ: 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.

Estándares de programación: 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.

Espacio abierto: Es preferible una sala grande con pequeños cubículos o, mejor todavía, sin divisiones. Los pares de programadores deben estar en el centro. En la periferia se ubican las máquinas privadas. En un encuentro de espacio abierto la agenda no se establece verticalmente.

Reglas justas: El equipo tiene sus propias reglas a seguir, pero se pueden cambiar en cualquier momento. En XP se piensa que no existe un proceso que sirva para todos los proyectos; lo que se hace habitualmente es adaptar un conjunto de prácticas simples a las características de cada proyecto.

Ritmo sostenible, trabajando un máximo de 8 horas por día: Dado que el desarrollo de software se considera un ejercicio creativo, se estima que hay que estar fresco y descansado para hacerlo eficientemente; con ello se motiva a los participantes, se evita la rotación del personal y se mejora la calidad del producto.

1.6.2. SCRUM

Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle en 1994. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas Sprints, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración[7].

Características

- Equipos autodirigidos
- Utiliza reglas para crear un entorno ágil de administración de proyectos.
- No prescribe prácticas específicas de ingeniería.
- Los requerimientos se capturan como ítems de la lista de reserva del producto.
- El producto se construye en una serie de Sprints de un mes de duración.

Roles

- Scrum Master (Jefe del Equipo): Es un rol de administración que su función es asegurarse de que el proyecto se esta llevando a cabo de acuerdo con las prácticas, valores y reglas de Scrum y que todo funciona según lo planeado. Su principal trabajo es remover impedimentos y reducir riesgos del producto. Este rol suele ser desempeñado por un Gerente de Proyecto o Líder de equipo.

- Product Owner (Usuario interno o gerente): Es el responsable del proyecto, administra, controla y comunica El Backlog List. (Lista de Reserva). Es el responsable de encontrar la visión del proyecto y reflejarla en El Backlog List.
- Scrum Team (Equipo): Es el equipo del proyecto que tiene la autoridad para decidir como organizarse para cumplir con los objetivos de un Sprint. Sus tareas son: Effort Stimation (Estimar Esfuerzo), crear el Sprint Backlog (reserva de la carrera), revisar EL Product Backlog List (Lista de Reserva del Producto) y sugerir obstáculos que deben ser removidos para cumplir con los ítems que aparecen. Típicamente es un equipo de entre 5 y 10 personas cada una especializada en algún elemento que conforma los objetivos a cumplir, por ejemplo: Programadores, Probadores y Diseñadores de Interfaz de usuario, etc. La dedicación de los miembros del equipo debería a tiempo completo con algunas excepciones.
- Customer (Cliente): El cliente participa en las fases que involucran La Lista Product Backlog.
- Management (Gerente): Es el responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el proyecto. Participa en la selección de objetivos y requerimientos y en la selección del Scrum Owner. Tiene la responsabilidad de controlar el progreso y trabaja junto con el Scrum Master en la reducción de la Product Backlog List.

Proceso

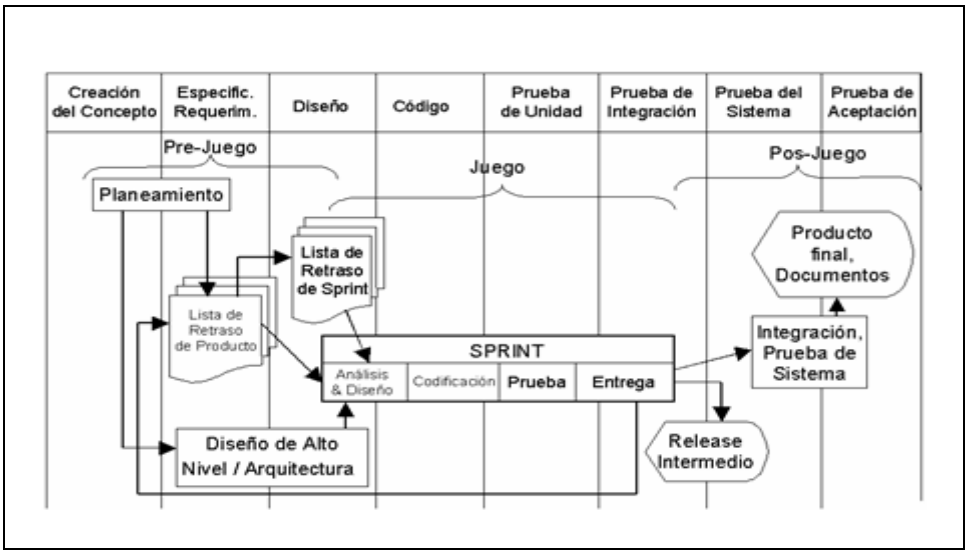
El ciclo de vida de Scrum es el siguiente:

Pre-Juego: Planeamiento: El propósito es establecer la visión, definir expectativas y asegurarse de la financiación. Las actividades son la escritura de la visión, el presupuesto, el registro de acumulación o retraso (Backlog) del producto inicial y los ítems estimados, así como la arquitectura de alto nivel, el diseño exploratorio y los prototipos.

Pre-Juego: Montaje (Staging): El propósito es identificar más requerimientos y priorizar las tareas para la primera iteración. Las actividades que se realizan: son planificación, diseño exploratorio y prototipos.

Juego o Desarrollo.: El propósito es implementar un sistema listo para entregar en una serie de iteraciones de treinta días llamadas “corridas” (Sprint). Las actividades son un encuentro de planeamiento de corridas en cada iteración, la definición del registro de acumulación de corridas y los estimados, y encuentros diarios de Scrum.

Pos-Juego: Liberación: El propósito es el despliegue operacional de las actividades, documentación, entrenamiento, mercadeo y venta.



Ciclo de Scrum, basado en Pekka Abrahamsson[8].

Prácticas

Sprint (Carrera)

Sprint Planning Meeting (Reunión de Planificación del Sprint)

Daily Meetings (Reuniones diarias)

Sprint Review Meeting (Reuniones de revisión del Sprint)

Design Review Meeting (Reuniones de revisión del Diseño)

Stabilization Sprint (Estabilización del Sprint)

Meta Scrum

Elementos

Los principales elementos de SCRUM son:

- Product Backlog: Lista de funcionalidades que necesita el cliente, la misma está priorizada según las preferencias que él mismo determine.
- Sprint Backlog: Lista de tareas que se van a realizar en un determinado sprint.

- Incremento: Parte del sistema que ha sido desarrollado en un sprint.

Los dos primeros forman parte los requisitos del sistema que se va a desarrollar, y el tercero es el valor que se le entrega al cliente al finalizar cada sprint.

1.6.3. ASD (Desarrollo de Software Adaptativo)

Esta metodología fue creada por Jim Highsmith en el año 2000. Sus principales características son: iterativo, orientado a los componentes de software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda se desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

Características

- Trabajo orientado y guiado por la misión del proyecto.
- Se basa esencialmente en la funcionalidad del producto.
- Desarrollo iterativo.
- Desarrollo acotado temporalmente.
- Trabajo tolerante ante el cambio.

Proceso

- Especulación, compuesta por 5 pasos:

1) Inicio para determinar la misión del proyecto.
2) Determinación del marco temporal del proyecto.
3) Determinación del número de iteraciones y la duración de cada una.
4) Determinación del objetivo de cada iteración.
5) Asignación de funcionalidad a cada iteración.

- Colaboración

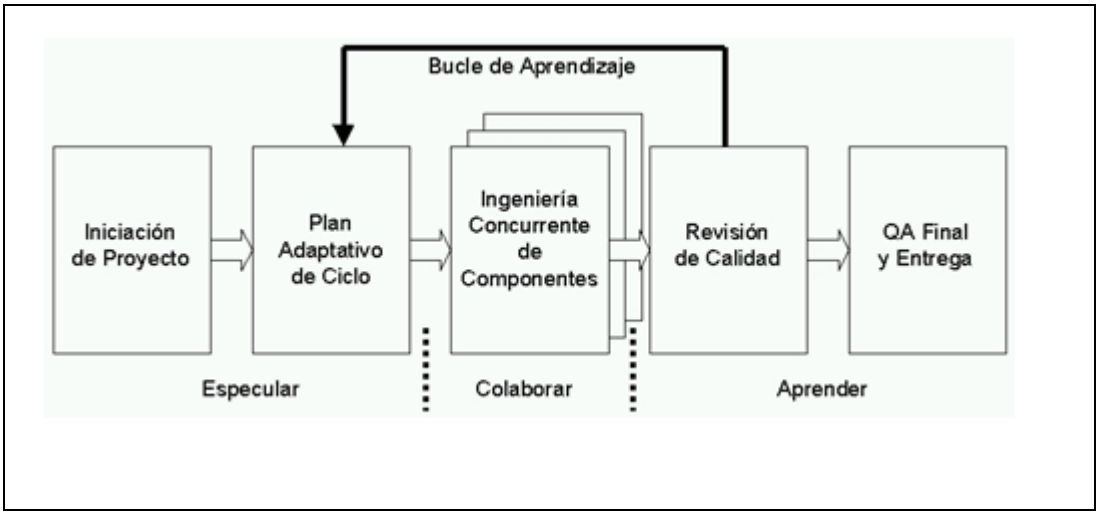
  Desarrollo concurrente del trabajo de construcción y gestión del producto.

- Aprendizaje

En cada iteración se revisa:

- Calidad, con el criterio de los clientes.
- Calidad, con el criterio de técnicos.
- Desarrollo de funcionalidad.
- El estado del proyecto.

Fases del ciclo de vida de ASD, basada en Highsmith[9].



Prácticas

Desarrollo de la misión del proyecto.

Planificación de un ciclo adaptable.

Desarrollo por componentes.

Equipos Colaborativos para un modelo de gestión adaptable.

Inspección del software.

Análisis posterior a la culminación del proyecto.

Elementos

- Un conjunto no estándar de artefactos de misión (Documentos para ambos, o sea para ti y para mi), incluyendo una versión del proyecto, una hoja de datos, un perfil de misión de producto y un esquema de su especificación.
- Un ciclo de vida, inherentemente iterativo.
- Cajas de tiempo, con ciclos cortos de entrega orientados por riesgos.

Un ciclo de vida es una iteración, dicho ciclo se basa en componentes y no en tareas, es limitado en el tiempo, orientado por riesgos y flexible al cambio. Que se basa en componentes, implica centrarse en el desarrollo de software, que funcionen, construyendo el sistema parte por parte. En este paradigma el cambio es bienvenido y necesario, pues se valora como la oportunidad de aprender y ganar así una ventaja competitiva.

La idea subyacente a ASD (y de ahí su particularidad) radica en que no proporciona un método para el desarrollo de software sino que más bien suministra la forma de implementar una cultura adaptativa en la empresa, con capacidad para reconocer que la incertidumbre y el cambio son el estado natural. El problema inicial es que la empresa no sabe lo que realmente quiere, y por tal razón debe aprender. Los cuatro objetivos de este proceso de aprendizaje son:

1) Prestar soporte a una cultura adaptativa o un conjunto mental para que se espere cambio e incertidumbre y no se tenga una falsa expectativa de orden.
2) Introducir marcos de referencia para orientar el proceso iterativo de gestión del cambio.
3) Establecer la colaboración y la interacción de la gente en tres niveles: interpersonal, cultural y estructural.
4) Agregar rigor y disciplina a una estrategia RAD, haciéndola escalable a la complejidad de los emprendimientos de la vida real.

ASD se concentra más en los componentes que en las tareas; en la práctica, esto se traduce en ocuparse más de la calidad que en los procesos usados para producir un resultado. En los ciclos adaptativos de la fase de Colaboración, el planeamiento es parte del proceso iterativo, y las definiciones de los componentes se refinan continuamente.

1.6.4. Crystal Methodologies

Capítulo 1

Crystal Methodologies fue creada por Alistair Cockburn en el año1998. Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en el equipo definido. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros); Amarillo, para 8 a 20; Naranja, para 20 a 50; Rojo, para 50 a 100. Se sigue con Marrón, Azul y Violeta. La más exhaustivamente documentada es Crystal Clear (CC), y es la que se ha de describir a continuación.

Crystal Clear consiste en valores, técnicas y procesos.

Los siete valores o propiedades de Crystal Clear[10]son:

1) Entrega frecuente. Consiste en entregar software a los clientes con frecuencia, no solamente en compilar el código. La frecuencia dependerá del proyecto, pero puede ser diaria, semanal, mensual o las que sean necesarias.
2) Comunicación osmótica. Todos juntos en el mismo cuarto. Una variante especial es disponer en la sala de un diseñador Senior; eso se llama Experto al Alcance de todos. Una reunión separada para que los concurrentes se concentren mejor.
3) Mejora reflexiva. Tomarse un pequeño tiempo (unas pocas horas cada alguna semana o una vez al mes) para pensar bien qué se está haciendo.
4) Seguridad personal. Hablar cuando algo ande mal: decirle amigablemente al manager que la agenda no es realista, o a un colega que su código necesita mejorarse. Esto es importante porque el equipo puede descubrir y reparar sus debilidades.
5) Foco. Saber lo que se está haciendo y tener la tranquilidad y el tiempo para hacerlo.
6) Fácil acceso a usuarios expertos. Un encuentro semanal o semi-semanal con llamados telefónicos adicionales parece ser una buena pauta. Otra variante es que los programadores se entrenen para ser usuarios durante un tiempo. El equipo de desarrollo, de todas maneras, incluye un Experto en Negocios.

7) Ambiente técnico con prueba automatizada, management de configuración e integración frecuente. No es más que la compilación y la integración varias veces al día.

Características

- Crystal aconseja que el tamaño del equipo sea reducido (Pocos componentes).
- La mejora de la comunicación entre los miembros del equipo del proyecto
- Utilización de políticas diferentes para equipos diferentes.
- Codificación por colores de Cristal.

Roles

- Patrocinador. Produce la Declaración de Misión con Prioridades de Compromiso, consigue los recursos y define la totalidad del proyecto.
- Usuario Experto. Junto con el Experto en Negocios produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe familiarizarse con el uso del sistema, sugerir atajos de teclado, modos de operación, información a visualizar simultáneamente, navegación, etcétera.
- Diseñador Principal. Produce la descripción arquitectónica. Se supone que debe ser al menos un profesional de Nivel En Metodologías Ágiles se definen tres niveles de experiencia: Nivel 1 es capaz de “seguir los procedimientos”; Nivel 2 es capaz de “apartarse de los procedimientos específicos” y encontrar otros distintos; Nivel 3 es capaz de manejar con fluidez, mezclar e inventar procedimientos). El Diseñador Principal tiene roles de coordinador, arquitecto, mentor y programador más experto.
- Diseñador-Programador. Produce, junto con el Diseñador Principal, los Borradores de Pantallas, el Modelo Común de Dominio, las Notas y Diagramas de Diseño, el Código Fuente, el Código de Migración, las Pruebas y el Sistema Empaquetado. Cockburn no distingue entre diseñadores y programadores. Un programa en CC es “diseño y programa”; sus programadores son diseñadores-programadores. En CC un diseñador que no programe no tiene cabida.
- Experto en Negocios. Junto con el Usuario Experto produce la Lista de Actores-Objetivos y el Archivo de Casos de Uso y Requerimientos. Debe conocer las reglas y políticas del negocio.

- Coordinador. Con la ayuda del equipo, produce el Mapa de Proyecto, el Plan de Entrega, el Estado del Proyecto, La Lista de Riesgos, el Plan y Estado de Iteración y La Agenda de Visualización.
- Verificador. Produce el Reporte de Bugs. Puede ser un programador en tiempo parcial, o un equipo de varias personas.
- Escritor. Produce el Manual de Usuario.

Proceso

Las distintas versiones de Crystal cuentan con un código de color para señalar la complejidad de una determinada metodología: mientras más oscuro un color, más pesado es el método. Cuanto más crítico es un sistema, más rigor se requiere. El código cromático se aplica a una forma tabular creada por Alistair Cockbum que se usa en muchos métodos ágiles para situar el rango de complejidad al cual se aplica una metodología.

Prácticas

- Entregas frecuentes
- Mejoras Reflexivas
- Seguridad Personal
- Fácil acceso a los usuarios expertos
- Ambiente técnico.

1.6.5. DSDM (Método de Desarrollo de Sistemas Dinámicos)

Define el marco para desarrollar un proceso de producción de software. Fue creada en 1994 por Arie Van Bennekum con el objetivo de crear una metodología RAD unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: Estudio de viabilidad, Estudio del negocio, Modelado funcional, Diseño y construcción, y finalmente Implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases.

Características

- Debe tener: Son los requerimientos fundamentales del sistema. De éstos, el subconjunto mínimo ha de ser satisfecho de forma íntegra.
- Debería tener: Son requerimientos esenciales para los que habrá una solución en corto plazo.
- Podría tener: Podrían quedar eliminados del sistema si no hay otra solución.
- Se desea que cuente, pero no lo tendrá en esta versión: Son los requerimientos valorados pero pueden esperar.

Roles

Los más importantes son: Programadores y Programadores Senior. Son los únicos roles de desarrollo. Ambos títulos cubren todos los roles de desarrollo, incluyendo analistas, diseñadores, programadores y verificadores.

- Coordinador técnico. Define la arquitectura del sistema y es responsable por la calidad técnica del proyecto, el control técnico y la configuración del sistema.
- Usuario Embajador. Brinda al proyecto conocimiento de los usuarios y disemina información sobre el avance del sistema a otros usuarios.
- Visionario. No es más que un usuario participante que tiene la percepción exacta de los objetivos del proyecto. Su misión principal es que el proyecto vaya en la dirección adecuada de los clientes.
- Patrocinador Ejecutivo. Es la persona que muestra autoridad y responsabilidad financiera y es el responsable de las decisiones más importantes.
- Facilitador. Encargado de administrar el avance del taller y la guía de la preparación y la comunicación.
- Escriba. Registra los requerimientos, acuerdos y decisiones tomadas en las reuniones, talleres y sesiones de prototipado.

Proceso

El ciclo de desarrollo de DSDM está compuesto de 5 fases, precedidas de un pre-proyecto y un post-proyecto.

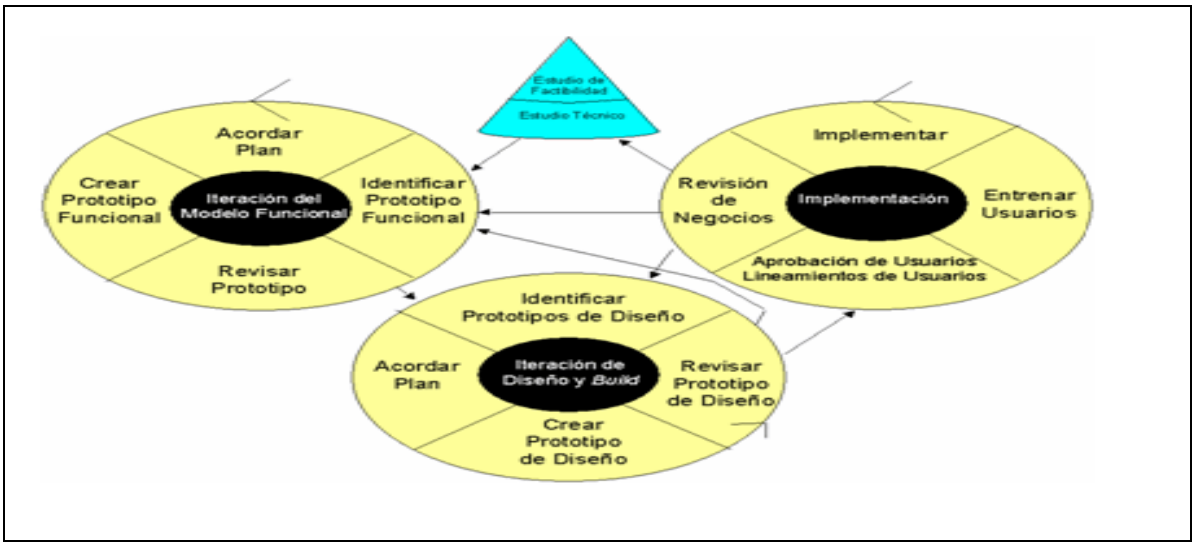
1) Estudio de viabilidad.

2) Estudio de negocio.
3) Iteración de modelado funcional.
4) Iteración de diseño y desarrollo.
5) Despliegue.

El método empieza con un estudio de viabilidad y negocio. El estudio de viabilidad considera si DSDM es apropiado para el proyecto. El estudio de negocio es una serie corta de talleres para entender el área de negocio donde tiene lugar el desarrollo. La iteración de modelado funcional es donde se plantea el contenido y la estrategia, se realiza la iteración y se analizan los resultados. Luego en la iteración de diseño y desarrollo es donde se construye la mayor parte del sistema y finalmente en el despliegue el sistema se transfiere del ambiente de desarrollo al de producción.

Las últimas tres fases son iterativas e incrementales. De acuerdo con la iniciativa de mantener el tiempo constante, las iteraciones de DSDM son cajas de tiempo. La iteración acaba cuando el tiempo se consume. Se supone que al final de la iteración los resultados están garantizados. Una caja de tiempo puede durar de unos pocos días a unas pocas semanas.

Gráfico que muestra el ciclo de vida de DSDM



Prácticas

- Es imperativo el compromiso activo del usuario.
- Los equipos DSDM deben tener el privilegio de tomar decisiones.

- El foco radica en la frecuente entrega de productos.
- Se requiere desarrollo iterativo e incremental.
- Todos los cambios durante el desarrollo son irreversibles.
- La línea base de los requerimientos es de alto nivel.
- La prueba está integrada a través de todo el ciclo de vida.
- Es esencial una estrategia colaborativa y cooperativa entre todos los participantes.

1.6.6. FDD (Desarrollo Conducido por Rasgos)

Define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Fue creada por Jeff De Luca y Peter Coad en el año 1998.

FDD es un método ágil, iterativo y adaptativo. A diferencia de otras Metodologías Ágiles, no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción y se considera adecuado para proyectos mayores y de misión crítica.

FDD no requiere un modelo específico de proceso y se complementa con otras metodologías. Enfatiza cuestiones de calidad y define claramente entregas tangibles y formas de evaluación del progreso. Su implementación de referencia, análogo al C3 (primer proyecto creado en XP), fue el Singapore Project; DeLuca había sido contratado para salvar un sistema muy complicado para el cual el contratista anterior había producido, luego de dos años, 3500 páginas de documentación y ninguna línea de código. Naturalmente, el proyecto basado en FDD fue todo un éxito, y permitió fundar el método en un caso real de misión crítica.

Roles

Hay tres categorías de roles en FDD: roles claves, roles de soporte y roles adicionales. Los roles claves son:

- Administrador del proyecto, quien tiene la última palabra en materia de visión, cronograma y asignación del personal.
- Arquitecto jefe.
- Manager de desarrollo.

- Programador jefe, que participa en el análisis de los requerimientos y selecciona rasgos del conjunto a desarrollar en la siguiente iteración.
- Propietarios de clases, que trabajan bajo la guía del Programador jefe en diseño, codificación, prueba y documentación.
- Experto de dominio, que puede ser un cliente, patrocinador, analista de negocios o una mezcla de todo lo anterior.

Los roles de soporte son:

- Administrador de entrega, que controla el progreso del proceso revisando los reportes del programador jefe y manteniendo reuniones breves con él.
- Abogado de lenguaje, es aquel que conoce a la perfección el lenguaje y la tecnología que será usada.
- Ingeniero de construcción, es el encargado del control de versiones y pública la documentación.
- Herramentista (toolsmith), es el responsable de mantener las bases de datos y sitios Web.
- Administrador del sistema, que controla el ambiente de trabajo.

Los tres roles adicionales son los de verificadores, encargados del despliegue y escritores técnicos.

Proceso

FDD consiste en cinco procesos secuenciales durante los cuales se diseña y construye el sistema. La parte iterativa soporta desarrollo ágil con rápidas adaptaciones a cambios en requerimientos y necesidades del negocio. Cada fase del proceso tiene un criterio de entrada, tareas, pruebas y un criterio de salida. Típicamente, la iteración de un rasgo insume de una a tres semanas.

Las fases son:

Desarrollo de un modelo general: Cuando comienza el desarrollo, los expertos de dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir. A esta altura se espera que existan requerimientos tales como casos de uso o especificaciones funcionales. Los expertos de dominio presentan un ensayo más en el que los miembros del equipo y el arquitecto jefe se informan de la descripción de alto nivel del sistema. El dominio general se subdivide en áreas más específicas y se define un ensayo más detallado para cada uno de los miembros del dominio. Luego un equipo de

desarrollo trabaja en pequeños grupos para producir modelos de objetos de cada área de dominio. Simultáneamente, se construye un gran modelo general para todo el sistema.

Construcción de la lista de rasgos: Los ensayos, modelos de objeto y documentación de requerimientos proporcionan la base para construir una amplia lista de rasgos. Estos rasgos son pequeños ítems útiles a los ojos del cliente. La lista de rasgos es revisada por los usuarios y patrocinadores para asegurar su validez y exhaustividad, los rasgos que requieran de más de diez días se descomponen en otros más pequeños.

Planeamiento por rasgos: Incluye la creación de un plan de alto nivel, en el que los conjuntos de rasgos se ponen en secuencia conforme a su prioridad y dependencia, y se asigna a los programadores jefes.

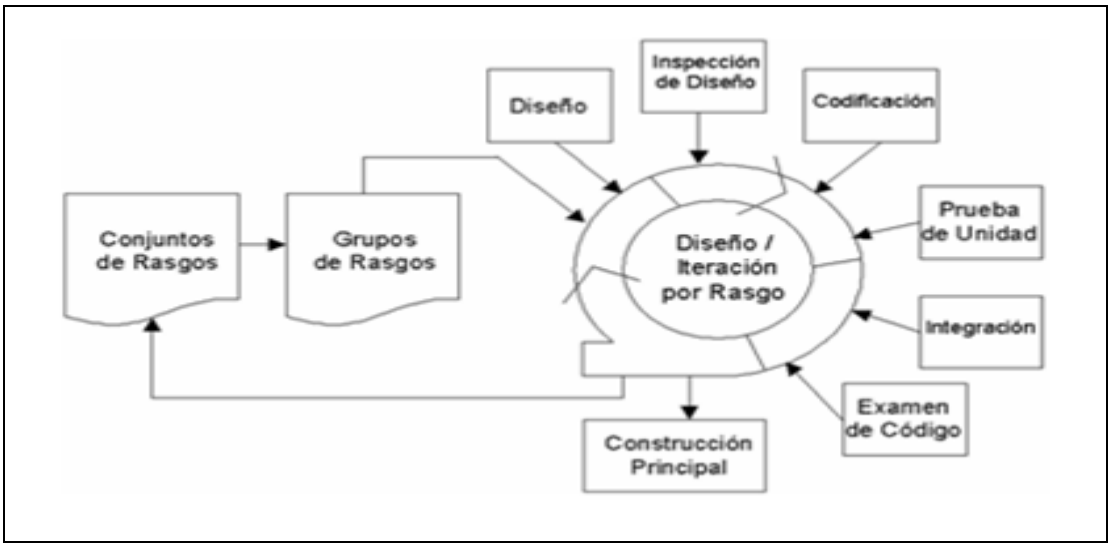
Diseño por rasgos y Construcción por rasgos: Se selecciona un pequeño conjunto de rasgos del conjunto, y los propietarios de clases seleccionan los correspondientes equipos dispuestos por rasgos. Se procede luego iterativamente hasta que se producen los rasgos seleccionados. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.

Prácticas

- Modelados de objetos de dominio. Consiste en el desglose de un problema en otros menores, el diseño y la implementación de cada clase u objeto se convierte en un problema sencillo a resolver. Cuando se integran las clases completas, forman la solución al problema mayor. Un aspecto particular de la técnica es el modelado en colores, que agrega una dimensión adicional de visualización.
- Desarrollo por Rasgos. Simplemente lograr que las clases y objetos funcionen. El control del progreso se realiza mediante pequeños exámenes de funcionalidad y descompuesto en funciones valoradas por el cliente. Un rasgo en FDD es una función pequeña expresada en la forma <acción> <resultado> <por | para | a> <objeto>.
- Propiedad individual de clases. Cada clase en específico tiene una sola persona asignada como responsable por su consistencia.

- Equipos de rasgos, pequeños y dinámicamente formados. La formación de un equipo garantiza que un conjunto de mentes se les apliquen a cada decisión y se tomen en cuenta las alternativas pertinentes.
- Inspección. Uso de los mejores mecanismos de detección.
- Builds regulares. Estos forman la base a partir de la cual se van agregando nuevos rasgos.
- Administración de configuración. Permite realizar seguimiento histórico de las últimas versiones completas de código fuente.
- Reporte de progreso. Comunicación a todos los niveles organizacionales que los requieran.

La siguiente gráfica muestra el ciclo de vida de FDD



1.6.7. MSF (Marco de Trabajo de Soluciones de Microsoft)

Visión General de MSF:

Microsoft Solution Framework es una metodología para el desarrollo de software para la planificación, desarrollo y gestión de proyectos tecnológico. Creada en el año 1994 por Microsoft. Se centra en el modelo de procesos y de equipo dejando los demás aspectos en segundo plano. Está compuesto de varios modelos que se encargan de cada una de las fases del desarrollo de un proyecto: Modelo de Arquitectura del proyecto, Modelo de Equipo, Modelo de Procesos, Modelo de Gestión de Riesgo, Modelo de Diseño de procesos y Modelo de Aplicación.

Características

- Adaptable: Es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un lugar específico.
- Escalable: Puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- Flexible: Es utilizada en el ambiente de desarrollo de cualquier cliente.
- Tecnología Agnóstica: Porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el Modelo de Aplicación.

- Modelo de Arquitectura del Proyecto*:* Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.
- Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.
- Modelo de Proceso: Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de Equipo.
- Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.
- Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño

eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

- Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

MSF está compuesta por las siguientes fases:

Fase 1 Estrategia y alcance: En esta fase el equipo y el cliente definen los requerimientos del negocio y los objetivos generales del proyecto. La fase culmina con el hito Visión y Alcance aprobados.

- Elaboración y aprobación del documento de alcances del proyecto.
- Formación del equipo de trabajo y distribución de competencias y responsabilidades.
- Elaboración del plan de trabajo.
- Elaboración de la matriz de riesgos y plan de contingencia.

Fase 2 Planificación y prueba de concepto: Durante la fase de planeación el equipo crea un borrador del plan maestro del proyecto, además de un cronograma del proyecto y de la especificación funcional del proyecto. Esta fase culmina con el hito Plan del proyecto aprobado.

- Documento de planificación y diseño de arquitectura.
- Documento de plan de laboratorio (son las pruebas de conceptos).

Fase 3 Desarrollo: Esta fase involucra una serie de releases internos del producto, desarrollados por partes para medir su progreso y para asegurarse que todos sus módulos o partes están sincronizados y pueden integrarse. La fase culmina con el hito Alcance completo.

Fase 4 Estabilización: Esta fase se centra en probar el producto. El proceso de prueba hace énfasis en el uso y el funcionamiento del producto en las condiciones del ambiente real. La fase culmina con el hito Release Readiness aprobado.

- Selección del entorno de pruebas piloto.
- Gestión de incidencias.

- Revisión de la documentación final de la arquitectura.
- Elaboración de plan de despliegue.
- Elaboración del plan de formación.

Fase 5 Despliegue: En esta fase el equipo implanta la tecnología y los componentes utilizados por la solución, estabiliza la implantación, apoya el funcionamiento y la transición del proyecto, y obtiene la aprobación final del cliente. La fase termina con el hito Implantación completa.

- Registro de mejoras y sugerencias.
- Revisión de las guías y manuales de usuario
- Entrega del proyecto y cierre del mismo.

Roles

MSF valora mucho el trabajo en equipo, tal es el caso que define un grupo de equipos para el desarrollo eficiente del proyecto, los siguientes se relacionan a continuación:

- Vista del modelo de equipo, Administración del producto, Administración del programa, equipo de Desarrollo, Equipo de Pruebas, Equipo encargado de las experiencias con el usuario y por último el Administrador de Versiones. Con el objetivo de asegurar la eficiencia del trabajo de estos equipos define los siguientes roles.
- Administrador del Producto: Su misión fundamental es manejar el marketing y las relaciones públicas, determinar el valor del negocio que no es más que definir y mantener la justificación para el negocio, también es el encargado de planear el producto, aquí es donde se analizan y priorizan los requerimientos de los clientes, determinan las métricas del negocio y los criterios de éxitos del proyecto.
- Arquitecto de Soluciones: Su misión es manejar todos los posibles diseños de solución, manejar las especificaciones funcionales y manejar el alcance de la solución y acuerdos de decisión críticos.
- Soporte: Es el encargado de desarrollar procedimientos de recuperación ante las fallas que puedan surgir en producto, así como proveer soporte al cliente y proveer soluciones para dar respuestas rápidas a los clientes de cualquier tipo de incidente ocurrido.

Roles que intervienen dentro del proceso MSF.

- Analista de Negocios.
- Jefe de Proyecto.
- Arquitecto.
- Desarrollador.
- Tester.
- Administrador de Releases.

Modelo de Procesos de MSF:

El modelo de procesos del MSF describe una secuencia de actividades de alto nivel para la construcción y desarrollo de soluciones de tecnologías de la información (IT por las siglas en inglés de Information Technology).

Los principios del modelado de procesos del MSF son:

- Trabajar con una visión en común.
- Mantenerse ágiles, esperando los cambios imprevistos que puedan surgir.
- Concentrarse en la entrega de valores de negocios.
- Fomentar la comunicación abierta.

Prácticas

- Dividir los proyectos grandes en partes más pequeñas manejables.
- Uso de prototipos.
- Uso de implementaciones frecuentes y pruebas rápidas.
- Ciclos rápidos.
- Estimación ascendente.

1.7. Estado Actual de los Procesos de Evaluación de Metodologías de Desarrollo de Software.

En la actualidad los procesos de evaluación relacionados con las Metodologías de Desarrollo de Software se enfocan en la descripción y comparación de diversos modelos de proceso de desarrollo de

software, en su gran mayoría lo que tratan de hacer estos procesos o guías para la evaluación de metodologías es tratar de responder a una serie de preguntas relacionadas con el tema como es el caso de: ¿Cuáles de todas las tendencias sobre metodologías de desarrollo de software es la más apropiada para mi organización?, ¿Cuál es la diferencia entre las metodologías ágiles y las conocidas como "tradicionales"?, ¿Qué características particulares poseen?, ¿Pueden combinarse?, ¿Qué ciclo de vida de desarrollo del software se propone?, ¿Qué requerimientos de adopción poseen? entre otras. Sin embargo no entran en los detalles de cada metodología como seria una evaluación cuantitativa para los elementos que de se deben producir y tener en cuenta para el desarrollo de un software.

Con el surgimiento de las Metodologías Ágiles se han realizados algunos trabajos relacionados con la evaluación de metodologías de desarrollo de software, con el propósito de establecer criterios comparativos: algunos de ellos se relacionan a continuación: La empresa Pragma Consultores realizo una interesante trabajo relacionado con el tema llamado ¿Agile o Unified ?disponibles en[11], el marco de trabajo (framework) 4- DAT[12] y Balanceo de Metodologías Orientadas al Plan y Ágiles[13].

1.8. Framework de Evaluación de Metodologías de Desarrollo de Software.

1.8.1. 4-DAT[12]

Es un framework diseñado para determinar el grado de agilidad de una determinada metodología. En esencia su función principal está en realizar una valoración de los principales elementos que dan la agilidad a los procedimientos ágiles tal es el caso de: la velocidad a la hora de realizar el producto, la flexibilidad, el aprendizaje y la sensibilidad.

Descripción del Framework

El análisis lo sustenta sobre las siguientes metodologías: SCRUM, XP, DSDM, Crystal, FDD y ASD.

Está estructurado en 4 dimensiones:

1) Alcance del Método. Aquí se incluye: Tamaño del Proyecto, Tamaño del equipo de desarrollo, Estilo de desarrollo, Estilo del código, Ambiente Tecnológico, Ambiente Físico, Cultura de Negocios y Mecanismos de Abstracción.
2) Caracterización de la agilidad: Aquí se incluye: Flexibilidad, Velocidad, Aprendizaje y Responsabilidades.

3) Valores Ágiles. Aquí se incluye: Los individuos e iteraciones por encima de los procesos y las herramientas, El software activo por encima de la documentación comprensiva, La negociación del cliente sobre la negociación del contrato, La respuesta al cambio sobre el seguimiento de un plan, Mantener los procesos ágiles y Mantener efectividad en los costos.
4) Caracterización del proceso de software. Aquí se incluye: Proceso de Desarrollo, Proceso de gerencia del proyecto, Configuración y control del proceso de software.

Después de hacer el análisis de cada una de las Metodologías Ágiles en cuanto a los parámetros establecidos se procede a calcular el grado de agilidad de cada procedimiento mediante la siguiente ecuación:

DA (Objeto) = (1/m) ∑m DA (Objeto, Fases ó Prácticas).

Donde DA es el grado de agilidad.

1.9. Marco de desarrollo nacional

En Cuba no existió cultura de producción de software hasta hace no más de una década, lo que trajo consigo que no se fabricaran software comercial sino artesanal por completo. Antes del año 2000, solo utilizaban metodologías de desarrollo de software en Cuba los centros adscriptos a universidades o las propias universidades como la CUJAE que utilizaban: ADESA, ADOOSI para sistemas de gestión estructurados u Orientados a Objetos y MULTIMET para sistemas de hipermedia.

Fue solo a partir del surgimiento de la metodología RUP (Por su impacto a nivel internacional) que se comenzó a producir software con la diversificación en ese sentido. Contribuyó a ello el interés político del país de pasar de artesanos a industriales en la producción de software lo que logró que nacionalmente se comenzaran a interesar por este tema aquellos que se dedicaban a la producción de software.

Por tales motivos se puede asegurar que las empresas adscriptas al MIC (Ministerio de las Informáticas y las Comunicaciones) lo que más usan es RUP (Rational Unified Process, en español Proceso Unificado de Rational), aunque con ciertas modificaciones para aterrizarlo en sus intereses. Además en las universidades cubanas la formación docente en la materia incluye a las metodologías de desarrollo que están sustentadas sobre la base de RUP por ser las más completa, factible, robusta y las más conocidas de todas.

A nivel nacional, la metodología de desarrollo más conocida de las relacionadas anteriormente es XP por ser precisamente las más reconocida a nivel internacional, lo que no implica que sea la mejor .Esto nos da la medida de la poca usabilidad y conocimiento de procedimientos ágiles que se tiene en Cuba.

Después de haber caracterizado las metodologías ágiles más reconocidas a nivel internacional. Se ve la necesidad imperante de lograr establecer un método para que realice el cálculo de la agilidad de estas, ya que en la mayoría de la bibliografía relacionada con este tema los autores se limitan a decir que tan ágil es una u otra metodología, pero sin tener en concreto los valores que puedan guiar a los que se dispongan hacer uso de estas importantes metodologías de desarrollo de software. Por tales motivos se procederá en el siguiente capítulo a brindar un método para calcular la agilidad de estas con el fin de valorar que tan rápidas son para el proceso de software. Sin estar enmarcado en una valoración cualitativa de los aspectos más relevantes de estos procedimientos de desarrollo de software.

2. CAPÍTULO 2: DESARROLLO DEL MÉTODO CUANTITATIVO PARA LA EVALUACIÓN DE METODOLOGÍAS ÁGILES.

2.1. Introducción

Investigar metodologías de desarrollo de software implica, sin lugar a dudas, el estudio y análisis de diversos parámetros que según el tema en cuestión puedan aportar resultados tangibles para evaluar o concluir sobre algo, en este caso (Con el objetivo de realizar un método cuantitativo para la evaluación de Metodologías Ágiles). Basado en lo anterior, el principal objetivo de este capítulo, es realizar un método para la evaluación cuantitativa de los procedimientos ágiles, basado en 5 dimensiones. Estas brindarán como resultado un valor, que mientras mayor sea, más ágil es la metodología, ya que cumpliría con gran parte de la cantidad de elementos definidos por cada dimensión.

Con la aplicación de este método a los procedimientos ágiles, se lograría obtener que tan ágil son estos, en cuanto a los elementos principales que fueron definidos para el desarrollo de un software con la menor cantidad de artefactos producidos en cada fase, así como el cumplimiento que les dan las Metodologías Ágiles a otros aspectos que son de vital importancia a la hora de realizar software con un límite de tiempo restringido y sobre todo que sea indispensable la entrega del producto funcional lo antes posible al cliente.

La importancia del valor que pueda arrojar una determinada metodología ágil después de vérsele aplicado el método esta substancialmente enfocado a la hora de la posible selección de una de estas para la realización de un determinado producto de software que requiera de un rápido desarrollo.

2.2. Dimensiones

Las dimensiones establecidas para obtener el grado de agilidad en los aspectos concernientes al desarrollo de software de forma ágil, se basan esencialmente en el análisis y estudio realizado al framework 4- DAT, además del aporte realizado en la investigación en cuanto a: la agregación de la quinta dimensión relacionada con los roles, la determinación de los indicadores medibles para la dimensión vinculada con los parámetros de la agilidad, la inclusión de todos los elementos que podían ser cuantificados para las dimensiones definidas así como la determinación de aquellos artefactos que se consideren necesarios obtener en cada una de las etapas básicas del software.

El nivel de significación de cada una de las dimensiones establecidas responde a la verificación de la metodología en el cumplimiento de la filosofía y la esencia de las Metodologías Ágiles en lo referente a las fases, las prácticas, los valores ágiles y la velocidad con que se debe desarrollar el producto de software.

Dimensión 1: Alcance de la metodología**:** Esta dimensión está basada en la determinación de los Modelos Esenciales que se necesitan generar en cada etapa básica del software con el fin de lograr obtener un sistema funcional, con la mayor agilidad posible, sin tener que enmarcarse en los rigurosos procesos que incluyen las metodologías tradicionales.

Dimensión 2: Atributos relacionados con la agilidad de las Metodologías Ágiles en las distintas etapas del software: En esta dimensión se definen 5 atributos, que en gran medida brindan la agilidad de los procedimientos ágiles. A partir de estos se definen indicadores medibles con el fin de garantizar un cálculo cuantitativo de la agilidad de los mismos.

Dimensión 3: Valores Ágiles: Aquí se definen los valores ágiles establecidos por el Manifiesto Ágil (Ver Epígrafe 1.2.1). La función de esta dimensión es realizar un cálculo de forma cuantitativa para ver el cumplimiento de los procedimientos ágiles teniendo en cuenta estos valores, ya que constituyen la base del enfoque liviano.

Dimensión 4: Caracterización del proceso de software: En esta dimensión se definen tres componentes que caracterizan el proceso de software, como es el caso del Desarrollo del Proceso, Proceso de Administración del Proyecto y el Proceso de Control de Configuración del Software. Cada uno de estos, tiene elementos importantes a tener en cuenta para la realización de un software ágil que cumpla con las expectativas y funciones del proceso de software de forma general.

Dimensión 5: Roles: En esta dimensión se definen los roles básicos para la realización de cualquier tipo de software. Con el fin de realizar una evaluación cuantitativa para esta dimensión, se analizarán los roles que definen las Metodologías Ágiles en correspondencia con los definidos como básicos.

2.3. Cálculo Cuantitativo del Grado de Agilidad.

Con el propósito de obtener una evaluación cuantitativa del grado de agilidad de las Metodologías Ágiles en la presente tesis se define la siguiente ecuación:

GAM =1/5 ∑ (VAD)

Donde GAM es el grado de agilidad de la metodología en específico analizada y VAD es el valor que indica cuan ágil es la metodología en la dimensión evaluada.

El GAM va ser igual a la sumatoria del valor del grado de agilidad de las 5 dimensiones analizadas divida por la cantidad de dimensiones (5).

El objetivo del valor que arroja esta ecuación es para tener en concreto que tan ágil es una determinada metodología de desarrollo de software. Partiendo de la premisa que mientras más grande sea el valor obtenido más ágil será el procedimiento, ya que cumplirá con la mayor cantidad de los elementos establecidos como esenciales a producir en cada una de las dimensiones definidas.

Los valores obtenidos por cada dimensión serán dados en dos cifras significativas.

Penalización = p

No es más que aplicarle una penalización a la dimensión que se este analizando, en caso de que la metodología que se este evaluando defina más actividades de la que se establecen como elementos fundamentales para ser ágil. El valor establecido para la penalización es de 0.1 y en el caso de la dimensión relacionada con la caracterización del proceso de software se le aplicara una penalización 0.2 a la metodología que no tenga establecidas las practicas suficientes para cubrir el ciclo de vida del software de forma integra.

2.4. Descripción para la obtención del valor cuantitativo en el alcance de la metodología.

Cabe destacar el hecho de que no se trata de simplificar las etapas de las metodologías, ya que si bien un número reducido de artefactos en una fase imprime una mayor rapidez a la hora del desarrollo de la misma, no debe realizarse un análisis trivial en este sentido, pues de incurrirse en la pérdida de información esencial para el desarrollo del proyecto, carecería de sentido dicha modelación. En esta etapa inicial se justifican los artefactos que serán denominados esenciales para el cumplimiento a cabalidad de las etapas del ciclo de vida de la aplicación. Para este proceso se asume que el ciclo de vida básico de un proyecto de software implica las fases de análisis, diseño, implementación y prueba.

Etapas básicas para la construcción de un software

Cuando se dispone a realizar un producto de software, hay que analizar que tan ágil se es en los aspectos básicos de esté, o sea en las etapas esenciales definidas para la realización del mismo como es el caso del análisis, el diseño, la implementación y las pruebas. Por ende es la selección de estas

etapas para medir el grado de agilidad de las Metodologías Ágiles, para reconocer que tan eficiente se puede ser con un determinado procedimiento ágil.

Análisis: El análisis en la realización de un software es un proceso de descubrimiento, refinamiento, modelado y especificación. Aquí se refinan en detalle los requisitos del sistema y el papel asignado al software. El análisis es la primera fase técnica del proceso de ingeniería de software. Es donde se realiza la declaración general del ámbito del software en una especificación concreta que se convierte en la base de todas las actividades relacionadas con la ingeniería de software. Este se enfoca en el dominio de la información funcional y del comportamiento del problema.

Diseño: El diseño de software es tanto un proceso como un modelo. El proceso de diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va construir. Sin embargo, es importante destacar que el proceso de diseño simplemente no es un recetario, sino más bien un conocimiento creativo, una experiencia en el tema, un sentido de lo que hace que un software sea bueno, y un compromiso general con la calidad, son factores críticos de éxito para un diseño competente. El diseño del software es un proceso iterativo mediante el cual los requisitos se traducen en un plano para construir el software. El diseño del software se encuentra en el núcleo técnico de la ingeniería del software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño del software es la primera de las tres actividades técnicas: diseño, generación de código y pruebas que se requieren para construir y verificar el software. Los requisitos del software, manifestados por los modelos de datos funcionales y de comportamiento, alimentan las siguientes tareas de diseño: diseño de datos, diseño arquitectónico, diseño de la interfaz y el diseño a nivel de componentes.

Implementación: En la fase de desarrollo se implementa el sistema, es decir, se crea el código correspondiente al resultado de la fase de diseño, siguiendo los patrones y la arquitectura escogida. El objetivo principal de la etapa de implementación es desarrollar la arquitectura y el sistema como un todo. De forma más específica, los propósitos de la implementación son: Definir la organización del código, planificar las integraciones del sistema necesarias en cada iteración e implementar las clases y subsistemas encontrados durante el diseño.

Pruebas: Es la etapa encargada de comprobar que el software realice correctamente las tareas indicadas en las especificaciones.

La agilidad en esta dimensión esta sujeta a un análisis realizado a partir de los elementos esenciales que se definen en cada etapa, que son valorados como los necesarios y con capacidad para

representar lo que realmente se desea en cada etapa, con el fin lograr una máxima agilidad en el proceso de software.

Modelos Esenciales que se deben desarrollar en cada etapa.

Análisis: Captura de Requisitos Funcionales, Captura de Requisitos NO Funcionales, Diseño de Algoritmos y Definición de la Arquitectura.

Diseño: Diagrama de Clases de Diseño.

Implementación: Diagrama de Componentes, Modelo de Implementación, Subsistema de Implementación y Diagrama de Despliegue.

Pruebas: Casos de Pruebas y Plan de Pruebas.

Ecuación para la obtención del grado de agilidad de esta Dimensión.

VAD (Alcance de la metodología) = ∑ coeficiente de completitud de cada etapa - p

El coeficiente de completitud de una etapa es la sumatoria del valor que tienen asignados los elementos de dicha etapa.

2.4.1. Descripción para la obtención del coeficiente de completitud para la etapa de análisis.

	Elementos	Valor Asociado
	Requisitos Funcionales.	0.40
Análisis	Requisitos NO Funcionales.	0.20
	Diseño de Algoritmos.	0.10
	Definición de la Arquitectura	0.30

Los elementos definidos en la etapa de análisis son considerados como los necesarios a cumplir, ya que de una forma ágil se obtendría lo que realmente se necesita para lograr los objetivos de la etapa, con el fin de lograr representar los Artefactos Esenciales realmente útiles para la obtención de un software funcional que cumpla con las expectativas del cliente en menor tiempo posible siempre y cuando cumpla con la calidad requerida.

Una metodología que en la etapa de análisis realice actividades que no se encuentren entre las definidas, estaría un poco disociada de lo que realmente proponen los enfoques ágiles, ya que se vería afectado el tiempo disponible para realizar la etapa de análisis. Básicamente este enfoque esta diseñado para responder a la tarea primordial de obtener el software funcional lo antes posible.

Además de que se estaría preparando para los futuros cambios que ocurrirían dentro del proceso del software que son inevitables y el contar con muchos artefactos en esta etapa atrasaría considerablemente los tiempos estimados para la obtención de los objetivos establecidos para esta etapa.

La consideración para ser ágil en la etapa de análisis es la captura de Requisitos Funcionales que es donde se describen las funcionalidades o los servicios que se esperan que el sistema provea, sus entradas y salidas, excepciones, etc. Requisitos No Funcionales, se refieren a las propiedades emergentes del sistema como la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, la capacidad de los dispositivos de entrada/salida, y la representación de datos que se utilizan en las interfaces del sistema. Diseño de Algoritmos; se refiere al análisis que se debe realizar con el fin de obtener en esta etapa los algoritmos necesarios para algunos casos donde se requieran por la complejidad que encierre la aplicación. Para lograr una máxima eficiencia en esta etapa se deben diseñar estos algoritmos para cuando pasen a la siguiente fase no se detenga el desarrollo del software.

La selección de la definición de la arquitectura en esta etapa, se basa en la importancia que tiene esta a la hora de crear una estructura lógica y física del sistema.

Cuando se logra establecer una arquitectura se pueden obtener los siguientes resultados:

Comunicación Mutua, Decisiones tempranas de Diseño, Restricciones Constructivas y Reutilización o abstracciones transferibles de un sistema entre otras. Estos elementos que brindan la arquitectura constituyen la base del fundamento de la inclusión de esta como un elemento de suma importancia a tener presente en el software. Adicionándole a estos resultados lo que se puede obtener con las conocidas 4 +1 vistas arquitectónicas relacionadas a continuación:

Vista de Casos de Usos: Esta vista muestra la funcionalidad del sistema como es percibida desde el exterior. Así como también describe un conjunto de escenarios y casos de uso que tienen una cobertura arquitectónicamente significativa o que ilustran un punto específico de la arquitectura. Es un subconjunto del Modelo de Casos de Uso y además su realización es obligatoria.

Vista Lógica: Describe el diseño más importante de las clases y su organización en paquetes y subsistemas, y la organización de éstos en capas. También contiene algunas realizaciones de casos de uso. Esta muestra cómo la funcionalidad es diseñada en el interior del sistema, en términos de la

estructura estática y comportamiento dinámico del sistema. Es un subconjunto del Modelo de Casos de Uso y su realización es obligatoria.

Vista de Procesos: Los procesos más importantes y las clases que interviene.

Vista de Implementación: Esta vista muestra la organización del código y el código actual de ejecución. Contiene una visión general del Modelo de Implementación y su organización en términos de módulos en paquetes y capas. También se describe la asignación de paquetes y clases de la Vista Lógica a los paquetes y módulos de la Vista de Implementación. Es un subconjunto del Modelo de Implementación.

Vista de Despliegue: Muestra la distribución física del sistema (ordenadores, dispositivos) y sus conexiones.

El valor asociado a cado uno de los elementos que fueron definidos en la etapa de análisis, estuvo basado en la descripción de la importancia de cada uno de estos.

2.4.2. Descripción para la obtención del coeficiente de completitud para la etapa de diseño.

	Elementos	Valor Asociado
Diseño	Diagrama de Clases del Diseño.	1

Partiendo de la concepción de que en la etapa de diseño se encuentra el núcleo técnico de la ingeniería del software. La fundamentación básica de la selección de este diagrama en la etapa de diseño, se basa, en que con la realización del Diagrama de Clases del Diseño se puede representar de forma objetiva la esencia del diseño. Ya que en el diagrama de clases se especifican las clases involucradas en el software con sus atributos, la relación entre ellas y los métodos que garantizan el funcionamiento del software, así como la representación de la parte estática del sistema. A través de este diagrama es que se pueden convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado, además es donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación.

Por tales motivos se define que para lograr el máximo de agilidad en esta etapa, se debe realizar solamente este elemento, esta valoración esta sujeta a que el equipo de desarrollo que se disponga a realizar un software bajo filosofías ágiles debe tener su personal totalmente capacitado para lograr que con el Diagrama de Clases del Diseño se logre representar lo que realmente sea significativo para la etapa de implementación.

2.4.3. Descripción para la obtención del coeficiente de completitud para la etapa de implementación

	Elementos	Valor Asociado
	Diagrama de Componentes.	0.25
Implementación	Modelo de Implementación.	0.25
	Subsistema de Implementación.	0.25
	Diagrama de Despliegue	0.25

En la etapa de implementación el objetivo principal es el desarrollo de la arquitectura y el sistema como un todo, específicamente se refiere a implementar las clases y subsistemas encontrados durante el diseño. Con el fin de garantizar el objetivo esencial de la etapa, se define: El Diagrama de Componentes, El Modelo de Implementación, Los Subsistema de Implementación y el Diagrama de Despliegue.

La fundamentación esencial para la determinación de estos cuatros elementos estuvo basada, en el resultado concreto que se obtendrían con ellos ya que al finalizar esta etapa el software estaría listo para la explotación y para ser probadas las funcionalidades implementadas. Esta determinación puede resultar ilógica para muchos, pero lo que si es cierto es que para la obtención rápida de software, solo se debe producir y hacer énfasis en lo que realmente pueda aportar resultados funcionales y entregables para los clientes, la actualidad del software no requiere de ceremonias en las fases sino de objetivos concretos.

El coeficiente de completitud responde a la importancia de ambos en esta etapa.

Diagrama de Componentes

Es de vital importancia la generación de este diagrama ya que garantizaría una representación de forma de grafo de los componentes del software unidos por medio de relaciones de dependencia (compilación, ejecución), mostrándose las interfaces que estos soportan.

Modelo de Implementación.

Su importancia se basa en describir como los elementos del modelo de diseño, Ejemplo: las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen de los componentes unos de otros.

Subsistema de implementación.

Es de gran importancia generar este artefacto en esta fase ya que es el encargado de proveer una colección de componentes y otros subsistemas de implementación usados para estructurar el modelo de implementación y dividirlos en pequeñas partes que pueden ser integradas y probadas de forma separada.

Los subsistemas de implementación incluyen dependencias y otras informaciones. También podrían incluir modelos claves del subsistema (diagramas de componentes, modelo de despliegue).

Diagrama de Despliegue.

El diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software (procesos y objetos que se ejecutan en ellos).

Después de ver descrito la esencia de cada unos de los artefactos que se definen como necesarios a cumplir en la etapa de implementación. Se esta en condiciones de apreciar la importancia de la definición de estos, este análisis básicamente responde a la filosofía de las Metodologías Ágiles relacionadas con esta etapa y que no es más que la codificación sin entrar en detalles de la producción de estos tan esenciales artefactos. El análisis de la selección de los artefactos definidos para la etapa de Implementación también esta relacionado con que no todos los equipos de desarrollo cuentan con la destreza de codificar a partir de lo obtenido en el diseño.

2.4.4. Descripción para la obtención del coeficiente de completitud para la etapa de pruebas.

	Elementos	Valor Asociado
Pruebas	Casos de Pruebas.	0.50
	Plan de Pruebas.	0.50

Si el objetivo principal de las pruebas es evaluar que el software haga realmente lo que solicito el cliente, no es recomendable para los enfoques ágiles realizar todos los artefactos que se pueden generar en esta tan importante etapa del software según RUP (Proceso Unificado del Rational), solo se debe basar en la creación de los casos de pruebas y en el plan de pruebas, ya que estos garantizarían que se les realicen las pruebas pertinentes de funcionalidad al software bajo una estrategia bien definida.

Según el excelente libro sobre las pruebas del software de Glen Myers [14], establece normas que pueden servir acertadamente como objetivos de las pruebas:

- La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una buena prueba tiene éxito si descubre un error no detectado hasta entonces.

Con el fin de cubrir estos objetivos es importante establecer una estrategia de pruebas para los casos de pruebas definidos en el software y así responder de una forma ágil a las tareas relacionadas con esta tan importante etapa del software sin tener que incurrir en otros rigurosos procesos que están definidos para la realización de las pruebas del producto de software realizado.

Casos de Pruebas

La selección de los Casos de Pruebas como un artefacto esencial a cumplir dentro del desarrollo de software ágil estuvo basado en su misión fundamental, que es determinar que un requisito sea completamente satisfactorio. Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber al menos un caso de prueba para cada requisito a menos que un requisito tenga requisitos secundarios. En ese caso, cada requisito secundario deberá tener por lo menos un caso de prueba. Algunas metodologías como RUP recomiendan crear al menos dos casos de prueba para cada requisito. Uno de ellos debe realizar la prueba positiva de los requisitos y el otro debe realizar la prueba negativa.

Plan de Pruebas

Un plan de pruebas del software integra las técnicas de diseño de casos de pruebas en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. El plan de pruebas proporciona un mapa que describe los pasos que hay que llevar a cabo como parte de la prueba, cuándo se deben planificar y realizar esos pasos, cuánto esfuerzo, tiempo y recurso se van a requerir. Por tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de pruebas, la ejecución de las pruebas y la agrupación y evaluación de los datos resultantes.

El propósito del plan de pruebas es explicitar el alcance, enfoque, recursos requeridos, calendario, responsables y manejo de riesgos de un proceso de pruebas.

Un Plan de Pruebas es un instrumento muy útil para dar estructura tanto al proceso de pruebas como a la documentación de las mismas.

Se distinguen diferentes tipos, que pueden variar desde un plan general, aplicable a todas las pruebas, hasta un plan detallado que se utiliza para un tipo de pruebas específico, como la prueba de aceptación, la prueba de interfaz, etcétera.

La integridad y la importancia del plan de pruebas se puede ver reflejado en los elementos que incluye como es el caso de: indicador del plan, alcance, elementos a probar, estrategia, categorización de la caracterización, productos tangibles (documentos a entregarse), procedimientos especiales, recursos, calendario, manejo de riesgos y responsables.

Un buen plan de pruebas ayuda a la organización del proyecto completo, a la estimación de los recursos necesarios y a la gestión de la calidad en general.

Valor Asociado

Es el valor correspondiente a cada uno de los elementos esenciales definidos en cada una de las etapas. El significado de cada valor esta relacionado con la importancia que les fue atribuida a los elementos esenciales.

2.5. Descripción para la obtención del valor cuantitativo en los parámetros relacionados con la agilidad de las metodologías ágiles en las distintas etapas del software

El grado de agilidad de esta dimensión esta basado en la cuantificación de los indicadores definidos para los parámetros que brindan la agilidad de un determinado procedimiento.

Parámetros Establecidos

Velocidad: V: Elementos asociado con los procedimientos ágiles a la hora de realizar el software, como es el caso de la cantidad de roles, ya que las metodologías ágiles basan parte de su filosofía en este aspecto. Así como ser lo más subjetivo y eficiente a la hora de realizar las iteraciones en el tiempo definido.

Dinamismo: D: Es la velocidad con que se producen los cambios en los requerimientos. Los cambios de contexto, tanto sea de negocios, de la organización, técnicos, etc. Estos son considerados desde el punto de vista de cambios de requerimientos.

Aprendizaje: A: El indicador relacionado con este atributo se basa en el nivel de asimilación del conocimiento anterior actualizado y la experiencia de crear un ambiente de aprendizaje. El análisis de este indicador esta basado en la cooperación y aporte que pueden existir entre las distintas fases de los métodos ágiles.

Ligereza: L: Es la capacidad del equipo de desarrollo en producir lo más rápido posible y con la mejor calidad los aspectos relacionados con el lenguaje de modelado, o sea tener buena habilidad para obtener la modelación de lo que la metodología ágil en cuestión desee modelar. Esto en esencia se refiere a la capacidad y el adiestramiento de las personas encargadas en la tarea.

Flexibilidad: F: Se refiere a la capacidad de los procedimientos ágiles de enfrentarse a los cambios y ver a estos como algo común dentro del desarrollo de software y así dar la posibilidad al cliente de cambiar los requisitos cuando el desee. Este parámetro se puede sintetizar en uno de los principios básicos de las metodologías ágiles que no es más que responder al cambio antes de seguir estrictamente un plan, esto no quiere decir que no se planifique, pero si estar listos para enfrentar cualquier cambio en los requisitos, nuevas peticiones o cambios en general, para responder a ellos y rehacer los planes ya que lo importante es satisfacer al cliente y cumplir con sus necesidades.

Indicadores atribuidos a cada parámetro.

	Parámetros				
	Velocidad	**Dinamismo**	**Aprendizaje**	**Ligereza**	**Flexibilidad**
Indicadores	Duración de las iteraciones Roles involucrados.	Modificaciones de los Requisitos	Nivel de asimilación.	Complejidad del lenguaje de modelado.	Nivel de adecuación frente a cambios.

Cantidad de Roles por Metodologías Ágiles.

Es importarte aclarar que cada uno de estos roles en las metodologías ágiles puede tener asociado varios trabajadores en función de lograr las actividades asignadas para cada uno de estos. La gran mayoría de estos roles están relacionados con programadores “Senior”, que son personas con capacidad para cubrir todos los roles de desarrollo, incluyendo análisis, diseño, programación y pruebas.

Tiempo de duración de las iteraciones.

Se refiere a el énfasis que hacen los procedimientos ágiles en la duración de las iteraciones con el fin de ser lo más rápido posible, la gran mayoría de estos procedimientos definen una duración estable en cuanto a semana, y siguen el trabajo así haya o no cumplido la función la iteración realizada y los problemas existentes durante este proceso son tratados de resolver en la próxima iteración con el fin de ganar tiempo y no demorar más de lo establecido para cada iteración según la metodología.

- **Rango para los indicadores de la velocidad:** Este significa que mientras más cerca se este de uno, más ágil es la metodología.

- Rango para la cantidad de Roles.

El número a la izquierda representa la cantidad de roles y el de la derecha el valor asociado con el rango.

Cantidad de Roles	5	6	7	8	9	10	11	12	13	Mayor 14
Valor Asociado	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1

El análisis esta basado en una de las características básicas de las metodologías ágiles y no es más que la definición de poco roles, pero algo importante de valorar en este indicador es que ellas que definen pocos roles por el simple hecho de que tienen que ser personas con amplio desempeño en las distintas etapas del software, esto puede tener varios enfoques y distintos puntos de vistas, pero la realidad es que mientras en un equipo de desarrollo se tenga la menor cantidad posible de roles mejor será el desempeño de las tareas asignadas, ya que habrá mejor control y mayores exigencias, esto hay que tenerlo presente partiendo de la premisa de que el proyecto que se valla a desarrollar no sea de dimensiones muy extensas.

- Rango para la cantidad de iteraciones por semana.

Es un poco difícil establecer un rango para evaluar la cantidad de semanas que demoran las iteraciones en las Metodologías Ágiles, ya que algunas no establecen la duración de las iteraciones debido a que este tiempo es definido por el equipo de trabajo y no es algo que venga asociado con el procedimiento,

Con el fin de lograr la evaluación cuantitativa correspondiente a este indicador se define el siguiente rango:

Duración de las Iteraciones	1 - 2	3 - 4	5 - 6	> = 7
Valor Asociado	1	0.75	0.50	0.1

A la hora de asignarles a los distintos indicadores de la velocidad un determinado número del rango establecido entre 0 y 1, esto no fue tomado a la ligera existió un estudio basado en las premisas de que las Metodologías Ágiles en su filosofía se basan en la definición de pocos roles y la mínima duración en cuanto a semanas para las iteraciones, por tales motivos es que en la presente tesis se asume que mientras mayor se el valor de estos indicadores más cerca está el procedimiento de la agilidad. Ya que los roles que definen las Metodologías Ágiles satisfacen las necesidades del equipo de desarrollo, lo importante a valorar en este indicador es que casi en la totalidad de las Metodologías Ágiles definen roles de programadores Senior, que son aquellas personas con grandes capacidades a la hora de programar e incluso llegan a realizar tareas de diseño y de pruebas.

En cuanto a las iteraciones ellas definen un número de semanas para realizar esta tarea y su filosofía se basa en que cuando menos tiempo dediquen a las iteraciones más cerca se está en avanzar en las siguientes etapas, con el fin de avanzar las Metodologías Ágiles en su mayoría cuando realizan las iteraciones las realizan en el número de semanas establecido, y las cosas pendientes tratan de resolverlas en la próxima iteración, lo que no le da cavidad a detenerse por un determinado problema.

El valor para el parámetro relacionado con la velocidad, no será valorado en ninguna etapa solo que se obtendría la suma del valor asociado con la cantidad de roles y la duración de las iteraciones y este seria el que aparecerá el la agilidad total correspondiente a este parámetro.

- **Rango para el indicador de la ligereza.**

Este indicador se refiere a la complejidad del lenguaje de modelado. En esencia este indicador se basa en asignarle una evaluación de bien, regular ó mal de acuerdo con las habilidades del personal asignado para esta tarea, que no es más que la realización de todos los aspectos relacionados con la visualización, especificación, construcción y documentación de todos los artefactos que involucra un sistema de software.

Este indicador se tomaría en base a la semántica propia del lenguaje. Ya que se va del alcance de este trabajo de tesis, por lo que se de deja a criterio del evaluador, proponiéndole una escala de 0, 0.25, 0.50 0.75, 1.

- **Rango para el indicador de la flexibilidad.**

En este parámetro es analizado el nivel de adecuación de las metodologías ágiles frente a los cambios, esto no tiene mucha explicación ya que los procedimientos ágiles para todas las etapas definidas para la construcción de software enfrentan los cambios dándole la bienvenida esa es su

filosofía, a partir de que ellas se rigen por unos de los principios de los métodos ágiles que no es más que responder a los cambios más que seguir estrictamente un plan. A raíz de un análisis del enfoque que les dan estos procedimientos a los cambios se llego a la conclusión de no establecer valores para el rango y simplemente se toma este indicador con valor 1 o sea que todos estos procedimientos van a ser ágiles en lo referente al nivel de adecuación frente a los cambios.

- **Rango para los indicadores del aprendizaje**

La esencia del análisis realizado, se basa en el nivel de asimilación de las distintas fases. Para lograr obtener un valor asociado con el rango definido para determinar el grado de agilidad se analiza a cada uno de los procedimientos ágiles en cuanto a las etapas básicas de la construcción de un software. Para ello se consideran las fases que rigen a cada uno de estos procedimientos.

En las etapas básicas para la construcción del software definidas se analizan las fases relacionadas con cada una y a partir de ahí, se realizó la asignación del valor asociado. Este análisis estuvo basado teniendo en cuenta si la fase tenía presente el indicador analizado y de ahí se procedió a determinar el por ciento de garantía del indicador en cada fase y a este se le asignó un valor del rango.

Ejemplo: En una metodología que se definan 4 fases para una determinada etapa y las cuatros cumplen con el nivel de asimilación entonces representa el 100 % y este por consiguiente tiene una óptima agilidad y toma el mayor valor del rango, o sea 1. En caso de que aparezca definido el valor 0 en la tabla es que las fases relacionadas con las etapas no cumplen con el indicador.

- **Rango para el indicador del Dinamismo**

Este rango está basado esencialmente para el por ciento de modificaciones de los requisitos. Las metodologías ágiles todas garantizan de un 50 % a un 100 % de modificaciones de estos ya que la realidad objetiva del desarrollo de software actual está dada en cambios constante en los requisitos, porque en muchos de los casos los clientes inicialmente no saben lo que verdaderamente desean y el nivel de inestabilidad cambia de un 50 % a un 100 %, debido a que la filosofía de los procedimientos ágiles se enfoca en la interacción constante con el cliente. Los procedimientos ágiles en la etapa de análisis definen el transcurso del desarrollo del sistema como algo cambiante, pero no todas con el mismo nivel de aceptación. Por tales motivos se tomo del libro" Ágil Software Development Ecosystems" de Highsmith el valor que le dan a cada uno de estos procedimientos en base a una escala de 5 puntos, partiendo del enfoque que les dan los procedimiento ágiles analizados al sistema como algo cambiante en la etapa de análisis, o sea el cambio constante de requisitos que ocurrirán en

el transcurso del proyecto y de ahí se realizó en análisis para obtener el rango correspondiente**.** **(Ver anexo 1).**

5 = 100% = 1 4 = 80% = 0, 8 3 = 60 % = 0, 6.

El valor obtenido en base al rango de [0,1], está dado por el por ciento que representa el indicador en base a 5 según el análisis de Highsmith[9].

Este indicador solo es valorado en la etapa de análisis. Ya que es donde se definen los requisitos y se valora el transcurso del sistema como algo cambiante.

Casos en que no se puedan evaluar los indicadores

Habrán indicadores que no podrán ser evaluados por la falta de documentación con que cuentan las Metodologías Ágiles de forma general, ya que mucha de las bibliografías disponibles en Internet están bajo elevados precios, esto se debe en gran medida a que la información concreta de algunas de estas metodologías que han sido aplicadas algún determinado tipo de software son tema de confidencialidad de las empresas que las aplican. Para estos casos el método plantea la evaluación de 0 o sea que no será tenido en cuenta a la hora obtener el grado agilidad de la dimensión, o se dejan a consideraciones puntuales de los evaluadores. Un caso particular es el de la evaluación del dinamismo que solo será analizado en la etapa de análisis y las restante aparecerán con el valor 0.

Ecuación para la obtención del grado de agilidad de esta Dimensión.

VAD= Valor del Grado de Agilidad en la Dimensión y AT=Agilidad Total en la etapa.

VAD (Parámetros relacionados con la agilidad) = ∑ AT (Análisis, Diseño, Implementación y Pruebas) / n x m.

Donde *n* es la cantidad de etapas del desarrollo de software y *m* es la cantidad de parámetros definidos, ambas variables constituyen una constante ya que las etapas y los parámetros son fijos para todas las metodologías ágiles que le sea aplicado el método.

AT = Total de la suma de los valores asociados a cada rango de las etapas definidas. El caso de la agilidad total para los indicadores relacionados con el parámetro velocidad se obtiene mediante la suma del valor asociado a la metodología analizada en lo referente a la cantidad de roles y la duración de las iteraciones. Mientras que para los restantes parámetros la agilidad total se obtiene a través de la suma de los valores asociados a cada uno de los indicadores en las etapas definidas para la obtención de la agilidad en esta dimensión.

Después de obtener el resultado de la suma en cada etapa se realiza una sumatoria total y se divide por la constante definida para la cantidad total de etapas (m) y como resultado final para el grado de agilidad se suma el resultado de todas las etapas y se divide por el producto de las etapas (m) y la cantidad de parámetros definidos que siempre será una constante (5) que son los definidos.

El caso donde aparece el valor 0 es que no está definido el valor del indicador para ese parámetro.

En la siguiente tabla se muestra como será realizado el procedimiento para obtener el grado de agilidad de las metodologías ágiles definidas.

	V	S	D	L	F
Etapas					
Análisis	-	[0;1]	[0;1]	[0;1]	[0;1]
Diseño	-	[0;1]	[0;1]	[0;1]	[0;1]
Implementación	-	[0;1]	[0;1]	[0;1]	[0;1]
Pruebas	-	[0;1]	[0;1]	[0;1]	[0;1]
Total	AT	AT	A T	A T	A T

2.6. Descripción para la obtención del valor cuantitativo en los valores ágiles

Para cada uno de los valores ágiles se definieron elementos con el fin de analizar las prácticas establecidas por las Metodologías Ágiles existentes que garanticen a cada uno de estos valores.

Ecuación para la obtención del grado de agilidad de esta Dimensión.

VAD (Valores Ágiles) = ∑ coeficiente de completitud de cada valor ágil - p.

El coeficiente de completitud no es más que el valor asociado a los elementos que se consideran necesarios en los valores ágiles.

Definición de Elementos Esenciales para los valores ágiles.

Individuos e iteraciones por sobre los procesos y las herramientas:

	Elementos	Valor Asociado
Individuos e iteraciones por sobre los procesos y las herramientas.	Mayor valor al equipo que al entorno.	0.50
	Realización de iteraciones en todas las etapas.	0.50

La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

El análisis para la determinación de los elementos necesarios para darle cumplimiento a este valor del Manifiesto Ágil, estuvo basado propiamente en lo que define él como importante y es significativo la definición de que la metodología de desarrollo de software que haga otras cosas o simplemente no cumpla con los elementos establecidos no estaría en condiciones de cumplir este valor y por tanto sería afectado su proceso de desarrollo ya que perdería tiempo en tratar de realizar otras tareas, como es el caso de adaptar al equipo al entorno que se cree.

Los elementos establecidos para este valor tienen un significado importante por eso la asignación de 0.50 para ambos.

Software activo por encima de la documentación comprensiva:

	Elementos	**Valor Asociado**
Software activo por encima de la documentación comprensiva.	Producción de documentos necesarios.	0.20
	Entregas frecuentes.	0.40
	Pruebas de funcionamiento al software	0.40

La regla a seguir es no producir documentos a menos que sean necesarios de forma inmediata para tomar una decisión importante. Estos documentos deben ser cortos y centrarse en lo fundamental.

Realmente lo que desea la mayoría de los clientes en la actualidad es un software funcional lo antes posible, por eso es que los enfoques metodológicos ágiles tratan de satisfacer al cliente lo más rápido posible con entregas funcionales del producto. Sin embargo hay metodologías que por tratar de desempeñar un riguroso proceso se tardan en satisfacer al cliente con lo que realmente le interesa, que es ver lo antes posible el resultado de su petición. Por eso es la consideración que se le debe dar a este valor establecido por El Manifiesto Ágil. En el elemento relacionado con la producción de documentos, se basa objetivamente en promover la documentación, pero no cientos de páginas en tomos nunca mantenidos y rara vez usados.

Los elementos establecidos, sin dudas garantizan la función de un software activo a través de las prácticas que se establecen en los procedimientos ágiles. Pero los más importantes son las entregas

frecuentes y las pruebas de funcionamiento del software, que sin dudas con estos dos elementos se puede obtener un software para ser entregado al cliente, por tales razones es que ambos tienen el mayor valor de rango. Mientras que la producción de la documentación necesaria ocuparía un plano menos significativo, esto es basado en el enfoque que les dan las metodologías ágiles a la obtención de documentos, que solo se limitan a producir los necesarios.

La colaboración con el cliente más que la negociación de un contrato:

	Elementos	Valor de Rango
La colaboración con el cliente más que la negociación de un contrato.	Frecuente relación con el cliente.	1

Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

Esto realmente es un tema espinoso para algunas empresas dedicadas a la producción de software, ya que muchos de los clientes a los que se disponen a realizarles un producto están lejos de ellos o no disponen el tiempo requerido para ser los clientes parte del equipo de trabajo, pero es importante aclarar que si se trata de un enfoque ágil hay que tener bien cerca al cliente ya que con su ayuda se logrará gran parte del éxito del producto, por la relación constante y la satisfacción que tendrá cada vez que estime conveniente un cambio dentro del sistema.

La determinación de solamente un elemento para garantizar el cumplimiento de este valor establecido en El Manifiesto Ágil, está fundamentado en la importancia que se debe tener en la frecuente relación con el cliente y por tanto se concluye que un procedimiento que tenga bien definido y priorizado la relación con cliente estará en condiciones de ser ágil en lo referente a este aspecto.

Responder a los cambios más que seguir estrictamente un plan:

	Elementos	Valor Asociado
Responder a los cambios más que seguir estrictamente un plan.	Estrategia para responder a los cambios.	0.75
	Planificación flexible.	0.25

La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

El cambio es algo complicado en la producción actual de software, por eso no se concibe una metodología de desarrollo de software que no tenga trazada una estrategia para responder a estos y tratar de solucionar los problemas que puedan traer consigo un determinado cambio de requisitos en una etapa avanzada de producto.

La definición de los dos elementos establecidos para garantizar una respuesta ante los cambios, responde básicamente a la estrategia que se debe tener para ser ágil en este aspecto. Sin embargo de los dos establecidos el de mayor peso corresponde al de la estrategia que deben tener las metodologías ágiles para enfrentarse a estos. Por tales razones no se considera ágil un procedimiento que no tenga las prácticas necesarias para darle la bienvenida al cambio y no posea una planificación flexible.

2.7. Descripción para la obtención del valor cuantitativo en la caracterización del proceso de software

El proceso de desarrollo de software es un conjunto estructurado de actividades para desarrollar un sistema de software. Con el fin de garantizar el éxito requerido tanto por la parte del cliente como la del equipo de desarrolladores es de vital importancia contar con prácticas que garanticen que el transcurso del software se desarrolle bajo guías basadas en garantizar todos los aspectos concernientes con el ciclo de vida del producto, así como contar con un proceso de administración y de configuración que respalde las expectativas de lo que se desea lograr al final de terminado el software, que no es más que un sistema funcional que responda a los requisitos establecidos por el cliente. Con el propósito de cuantificar las prácticas involucradas en el proceso del software fueron definidos los siguientes aspectos: Proceso de Desarrollo, Proceso de Administración y Control de Configuración.

Ecuación para la obtención del grado de agilidad de esta Dimensión.

VAD (Caracterización del proceso de software) = ∑ coeficiente de completitud de los elementos que conforma el proceso de software - p.

	Elementos	Valor Asociado
Proceso de Desarrollo	Prácticas para cubrir el ciclo de vida completo del software.	1

Un procedimiento metodológico que cuente con prácticas para cubrir el ciclo de vida completo de un software se considera ágil, ya que proporciona una guía para desarrollar todas las actividades que se

involucran en las distintas etapas. Sin embargo una metodología que tenga límites de prácticas para alguna etapa específica del ciclo de vida se considera lenta en ese aspecto ya que tendrá que prescindir de la preparación del equipo de desarrollo para cubrir esas prácticas.

En el caso de que la metodología de desarrollo de software analizada no cuente con las prácticas necesarias para cubrir de forma integra el ciclo de vida de un software será penalizada con 0.20, o sea se le restará al valor uno esta penalización. El fundamento de este análisis estuvo centrado en lograr una mayor agilidad a través de las prácticas, ya que estas serían un punto de guía de lo que se debe producir bajo los enfoques ágiles.

	Elementos	**Valor Asociado**
Proceso de Administración del Proyecto.	Planificación del proyecto	0.75
	Realización de reuniones frecuentemente.	0.25

El proceso de administración del proyecto son las actividades que permiten asegurar que el software se lleve a cabo a tiempo y de acuerdo con la planificación. Así como de acuerdo con los requisitos del software. La administración del proyecto tiene varios objetivos, pero los primordiales a cumplir son la planificación del proyecto y las reuniones para ver el estado de desarrollo, así como los problemas que estén afectando el desarrollo. La fundamentación de la selección de solo estos dos elementos como los básicos a tener en cuenta, se basa en tratar de ser lo más óptimo posible a la hora de administrar el software. La evaluación para los elementos definidos dentro del Proceso de Administración del Software, están sujeto, a si la metodología analizada cuenta con las prácticas o las fases pertinentes para garantizar los elementos definidos.

Se considera que una metodología de desarrollo de software que realice una buena planificación del proyecto cumpliría de forma ágil la esencia del proceso de administración. Por eso es que se le asocia el mayor valor, mientras que a las reuniones se le da un valor un poco más discreto pero si vital a la hora de realizar este proceso.

	Elementos	**Valor Asociado**
Control de Configuración del Software	Administración de Configuración.	1

Gestión de Configuración es el proceso de identificar y definir los elementos en el sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando y reportando el estado de los

elementos y las solicitudes de cambio, y verificando que los elementos estén completos y que sean los correctos. Dentro de este vital proceso se encuentra el de la Administración de la Configuración que la esencia de sus objetivos se basa en permitir a los equipos de desarrollo capturar, controlar y administrar de forma segura los cambios y activos del software. La selección de este elemento dentro del Control y Configuración del Software esta basado en la importancia de este a la hora de manejar los inevitables cambios que puedan surgir en el transcurso del desarrollo del software.

2.8. Descripción para la obtención del valor cuantitativo en la definición de roles

Es importante la valoración que para el desarrollo de software con Metodologías Ágiles, se deben tener en cuenta la definición de pocos roles como parte de la esencia de la filosofía de las metodologías livianas, la evaluación cuantitativa de esta dimensión está basado en ver si los roles definidos por estas metodologías se corresponden con los establecidos como básicos para cualquier proyecto. La fundamentación de la definición de los roles seleccionados como los esenciales estuvo basado en los resultados objetivos que se obtendrían con estos, ya que se lograría obtener un software funcional, sin tener que incurrir en la definición de un grupo mayor de roles que entorpecería el proceso de desarrollo de un software ágil y se estaría incumpliendo con una de las características básicas de las Metodologías Ágiles que no es más que la definición de pocos roles. El establecimiento de estos roles como esenciales también estuvo sujeto a la valoración de las experiencias con que deben contar cada uno de los miembros del equipo que se disponga a desarrollar un determinado software bajo estas metodologías.

Ecuación para la obtención del grado de agilidad de esta Dimensión.

VAD (Roles) = ∑ coeficiente de completitud de cada rol - p.

El coeficiente de completitud no es más que el valor asociado a los roles de las metodologías ágiles que cumplen con los establecidos.

Roles Básicos.	Valor Asociado
Líder del Proyecto.	0.16
Analista	0.16
Diseñador	0.16
Programador	0.16
Arquitecto	0.16
Probador	0.16

La distribución del rango no es exactamente 1. Por lo que se decidió sumarle al resultado final de la suma de todos los roles 0.04.

La definición de roles según las metodologías tradicionales como es el case de RUP (Proceso Unificado del Rational). Se enfocan en la determinación de grandes cantidades roles esto en esencia puede ser muy importante para proyectos de gran envergadura o para equipos que no tengan conocimientos profundos sobre el desarrollo de software. Pero lo que si es importante aclarar que las metodologías ágiles como parte de su filosofía tratan de definir la menor cantidad de roles posibles, ya que esto le garantizaría mayor control de las actividades a los responsables, así como menos generación de gastos por concepto de personas incluidas en el desarrollo de software, por estos motivos estas metodologías tratan de realizar el software con personal altamente calificado y de varios años de experiencias. Si embargo es razonable aclarar que la gran mayoría de los que se disponen a realizar software bajo metodologías ágiles son programadores Senior, que no es más que aquellas personas que dominan el trabajo desde las fases de análisis hasta la implementación y pueden realizar cualquier tarea que se le asigne.

El fundamento de la selección de estos roles básicos para garantizar agilidad, partió de la premisa de que con ellos se puede realizar un software de forma integra, siempre y cuando cuenten con los conocimientos necesarios. Por la importancia de cada uno se le asignó de forma equitativa el mismo valor de rango.

Al finalizar este capítulo se logra concretar un método de evaluación cuantitativa para las metodologías ágiles existentes y las que puedan surgir con el de cursar de los años. La definición genérica del método permite una evaluación para obtener el grado de agilidad tanto de los enfoques ágiles como tradicionales, a partir de las dimensiones definidas para el desarrollo de una aplicación de software que requiera de una rápida realización.

3. CAPÍTULO 3: APLICACIÓN DEL MÉTODO CUANTITATIVO PARA LA EVALUACIÓN DE METODOLOGÍAS ÁGILES.

3.1. Introducción

La esencia de este capítulo esta enfocada en aterrizar de forma objetiva el método realizado en el capítulo anterior a las Metodologías Ágiles más reconocidas a nivel internacional. Con el fin de obtener el grado de agilidad de estas para establecer criterios comparativos después de ver obtenido su grado de agilidad y tener una guía de valores cuantificados a la hora de seleccionar cual de los enfoques analizados se ajustan más a las características del equipo de desarrollo, partiendo del cumplimiento que le dan a las dimensiones establecidas que en gran medida encierran las características básicas para el desarrollo de un software ágil. Además será analizada RUP desde su enfoque tradicional para observar que tan ágil pude ser al aplicarle el método.

En la actualidad existe lo que se puede llamar una revolución de Metodologías Ágiles, pero las de más impacto y las que más reporte de aplicación en proyectos tienen son XP, DSDM y SCRUM.

Con el objetivo de mostrar como sería el procedimiento de la aplicación del método, fueron seleccionadas las metodología antes mencionadas y RUP esta última con el fin de de analizar si puede arrojar valores definidos en los límites del método.

3.2. Aplicación del Método a las Principales Metodologías Ágiles

3.2.1. Aplicación del Método a XP (Programación Extrema)

Dimensión 1 Alcance de la metodología

En XP el proceso de captura de requisitos funcionales y no funcionales se logra a través de las Historias de Usuarios. Mientras que la definición de la arquitectura constituye unos de los objetivos esenciales dentro de fase de exploración, esta es realizada a partir de las Historias de Usuarios.

Historia de Usuarios (User Stories) **(Ver anexo 2)**

El concepto de las historia de usuarios tiene algo que ver con los famosos (“casos de uso”) utilizados en el ciclo incremental de desarrollo de software. Pero esta similitud se basa en que su cometido es el mismo, sirven para hacer las mismas cosas, pero no son lo mismo. Permiten sustituir unos largos requerimientos por una serie de historia de usuarios y además permiten hacer una estimación en el tiempo para la reunión de lanzamientos de las futuras versiones del sistema.

Además de esto, las Historias de Usuarios ayudan a crear tests de aceptación. Estos son pruebas que se aplicarán al sistema para ver si cumplen una determinada “Historia del Usuario”, lo cuál viene a ser lo mismo que cumplir un determinado requisito en otros modelos de desarrollo. Las Historias de Usuarios son realizadas por el propio cliente en forma de 3 sentencias de texto, en las que describe necesidades del sistema con la propia terminología del negocio, sin hacer uso de vocabulario técnico. Aquí también están reflejados los requisitos no funcionales donde se incluyen términos de la seguridad del sistema.

Como XP tiene definido para la etapa de análisis el cumplimiento de tres elementos establecidos para el cumplimiento del objetivo de forma ágil de la etapa analizada, le corresponde la evaluación asociada a esos elementos. Sin embargo esta metodología no tiene contemplado en la etapa de análisis el diseño de algoritmos, lo que obtendrá el valor 0 en la evaluación de este elemento. Vale aclarar que XP no será penalizado ya que no incluye la realización de otras tareas que no sean las establecidas.

XP	Elementos	Valor Asociado
	Requisitos Funcionales.	0.40
Análisis	Requisitos No Funcionales.	0.20
	Diseño de Algoritmos.	0
	Definición de la Arquitectura	0.30

El coeficiente de completitud para esta etapa es 0.90.

XP para la etapa de diseño define una práctica que es conocida como Diseño Simple que es donde 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 deben ser removidos 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.

La etapa de diseño en XP se basa esencialmente en el diseño de las cosas más simple que puedan funcionar y el uso de las tarjetas CRC.

Tarjetas CRC **(Ver anexo 3)**

Las clases pueden ser descubiertas usando tarjetas de Colaboración – Responsabilidad - Clases (CRC por sus siglas en inglés). Fueron introducidas por Ward Cunningham y Kent Beck en 1989. Una

tarjeta CRC es una tarjeta indexada de 3 x 5 que muestra: El nombre de la clase y su descripción, La responsabilidad de la clase, Conocimiento interno de la clase, Servicios brindados por la clase y Los colaboradores para las responsabilidades (Un colaborador es una clase cuyos servicios son necesarios para una responsabilidad).

- Una sección de tarjeta CRC incluye:

- Un grupo de personas escogidas para personificar un escenario.
- La creación de una tarjeta para cada objeto del escenario.
- La asignación de tarjetas a los participantes.
- Anotación de las responsabilidades y la colaboración en las tarjetas.
- Creación de nuevas tarjetas para los nuevos objetos descubiertos.

- Beneficios de las tarjetas CRC.

- A medida que más y más escenario son completados, emergen rastros de colaboración.
- Las tarjetas pueden ser físicamente arregladas para representar esas colaboraciones.
- Esto puede ayudar a representar jerarquías generalización/especificación, o jerarquías de agregación entre las clases.

Las tarjetas CRC son más efectivas para grupos novatos en técnicas de OO (Orientadas a Objetos) porque ellas: Provienen la focalización en los detalles de la OOP (Programación Orientada a Objetos) y la generalización.

XP en la etapa de diseño cumple con el elemento establecido como básico dentro de esta etapa del software, a través de las tarjetas CRC, por tanto obtiene el máximo valor asociado a la etapa del diseño y no esta sujeta a ninguna penalización.

XP	Elementos	Valor Asociado
Diseño	Diagrama de Clases del Diseño.	1

El coeficiente de completitud para esta etapa es 1.

XP en la etapa de implementación, solamente realiza la codificación o sea la implementación del diseño de clases que fue obtenido en la etapa diseño, por tanto no recibirá ningún valor de los asignados para los elementos que fueron seleccionados para esta etapa.

Los objetivos de esta etapa XP lo complementa mediante la siguiente práctica:

Refactorización: 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.

	Elementos	Valor Asociado
	Diagrama de Componentes.	0
Implementación	Modelo de Implementación.	0
	Subsistema de implementación.	0
	Diagrama de Despliegue	0

El coeficiente de completitud para esta etapa es 0.

XP para la etapa de pruebas cumple con los dos elementos establecidos para la etapa a cumplir para ser ágil en tal importante proceso, que es el de realizarles las pruebas pertinentes al producto para comprobar la eficiencia de lo requerido por el cliente. La estrategia seguida por XP para cumplir con lo definido como esencial en esta etapa esta basada en la producción de código dirigido 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 usuarios que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es de vital importancia.

En lo relacionado con el proceso de pruebas en XP es importante aclarar que las pruebas unitarias se deben escribir antes que el código y son realizadas por los programadores, mientras que las funcionales son hechas por los clientes y responden básicamente a la realización de los casos de

pruebas para verificar que los requisitos establecidos en las historias de usuarios sean completamente satisfactorios.

Casos de Pruebas (ver anexo 4)

	Elementos	Valor Asociado
Pruebas	Casos de Pruebas.	0.50
	Plan de Pruebas.	0.50

El coeficiente de completitud para esta etapa es 1.

Valor de la agilidad de XP en la dimensión Alcance de la metodología.

VAD (Alcance de la metodología) = ∑ coeficiente de completitud de cada etapa - p.

VAD (Alcance de la metodología) = (0.90 + 1 + 0 + 1) – 0

VAD (Alcance de la metodología) = 2.90

Dimensión 2 (Atributos para obtener la agilidad)

Los valores asociados a los indicadores de cada parámetro fueron obtenidos a partir del siguiente análisis.

Para el caso de los indicadores relacionados con la velocidad se analizaron la cantidad de roles que define XP y la duración de las iteraciones.

Cantidad de Roles: 7: Programador, Cliente, Encargado de Pruebas, Encargado de seguimiento, Entrenador, Consultor y Gestor.

Duración de las iteraciones: 2 semanas.

Según la puntuación de Highsmith para el indicador relacionado con el dinamismo, es de 5 puntos lo que representa el 100 % del enfoque que la da XP al sistema como algo cambiante en la etapa de análisis que es donde se analiza este indicador.

Para el caso de la evaluación del indicador del aprendizaje se obtuvo el siguiente resultado:

XP para las etapas definidas propone las siguientes fases:

Análisis: Exploración y Planeamiento ambas crean un ambiente de aprendizaje para ser utilizado por las siguientes fases por tanto en esta etapa se obtiene el 100% del cumplimiento del indicador.

Diseño: XP no define fases relacionadas con la etapa de diseño. Por tanto tiene el valor 0.

Implementación: XP define para esta etapa la producción y cumple con el indicador.

Pruebas: En la etapa de pruebas XP incluye una fase que se denomina mantenimiento y muerte del proyecto pero solamente la primera cumple con el indicador. Por ende solo recibe la mitad del valor asociado a esta etapa.

En el caso de la ligereza solo se tuvo presente para el desarrollo de la aplicación del método, el caso en que las habilidades del equipo de desarrollo para modelar los artefactos que se requieran sean de buenas. Por eso es que aparece el valor 1 en todas las etapas menos en la de pruebas, ya aquí ahí no se modela nada.

Para la flexibilidad se le asignó el valor 1 por la definición que se tomo para este indicador en el método por la filosofía que adoptan la mayoría de los procedimientos ágiles en cuanto al nivel de adecuación frente a los cambios.

	V	D	A	L	F
Etapas					
Análisis		1	1	1	1
Diseño		0	0	1	1
Implementación		0	1	1	1
Pruebas		0	0.50	0	1
Total	1.8	1	2.50	3	4

VAD (Parámetros relacionados con la agilidad) = ∑ AT (Análisis, Diseño, Implementación y Pruebas) / n x m.

Al aplicarle la fórmula correspondiente a esta dimensión se obtiene que su agilidad sea de 0.62.

Dimensión 3 (Valores Ágiles).

XP para garantizar los elementos definidos para el primer valor ágil definido, propone las siguientes prácticas:

El juego de la planificación. Hay una comunicación frecuente 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.

Propiedad colectiva del código: Cualquier programador puede cambiar cualquier parte del código en cualquier momento.

Cliente in-situ: 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.

Sin embargo no propone ninguna práctica en especifico para garantizar las iteraciones en todas las fases, pero si las realización porque eso constituye uno de los fundamentos de su filosofía.

Por las razones expuestas XP tiene el valor asociado a cada elemento.

	Elementos	**Valor Asociado**
Individuos e iteraciones por sobre los procesos y las herramientas.	Mayor valor al equipo que al entorno.	0.50
	Realización de iteraciones en todas las etapas.	0.50

Coeficiente de completitud para este valor es 1.

XP para garantizar los elementos definidos para el segundo valor ágil, propone las siguientes prácticas:

Entregas pequeñas: Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema.

Pruebas: La producción de código está dirigida por las pruebas unitarias. Estas son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

En cuanto la producción de documentos necesario no designa ninguna práctica, pero sin embargo ellos definen para la documentación solo generar los documentos que sean necesarios, como por ejemplo los casos de pruebas.

Como XP define prácticas para garantizar los elementos establecidos obtiene los respectivos valores asociados a cada elemento.

	Elementos	**Valor Asociado**
Software activo por encima de la documentación comprensiva.	Producción de documentos necesarios.	0.20
	Entregas frecuentes.	0.40
	Pruebas de funcionamiento al software	0.40

Coeficiente de completitud para este valor es 1.

XP con el fin de garantizar el cumplimiento del tercer valor ágil define las siguientes:

El juego de la planificación y el cliente in-situ. La definición de ambos quedo expuesta en el análisis del primer valor ágil visto en la presente dimensión. Como define prácticas para el cumplimiento de este valor obtiene la máxima agilidad en este elemento.

	Elementos	**Valor de Rango**
La colaboración con el cliente más que la negociación de un contrato.	Frecuente relación con el cliente.	1

Coeficiente de completitud para este valor 1.

XP con el fin de garantizar el primer elemento definido en el cuarto valor ágil propone las siguientes prácticas:

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 actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).

Diseño simple: Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.

Sin embargo no propone una planificación flexible, ya que tiene las cosas bien establecidas y no admite modificaciones. Por tanto XP en la evaluación de este valor solo obtiene el valor asociado al primer elemento.

	Elementos	**Valor Asociado**
Responder a los cambios más que seguir estrictamente un plan.	Estrategia para responder a los cambios.	0.75
	Planificación flexible.	0

Coeficiente de completitud para este valor 0.75.

VAD (valores ágiles) = ∑ coeficiente de completitud de cada valor ágil - p.

VAD (valores ágiles) = (1 +1 + 1 + 0.75) = 3.75 – 0 = 3.75.

Dimensión 4 (Caracterización del proceso de software).

XP para el proceso de desarrollo del software tiene establecido prácticas que garantizan de forma integra el ciclo de vida del software, las prácticas son las siguientes:

Entregas pequeñas, Metáfora, Diseño simple, Refactorización, Programación en parejas, Propiedad colectiva del código, Integración continua, Cliente y Estándares de programación.

Al poseer prácticas para cubrir el ciclo de vida completo de un software obtiene el valor máximo para este elemento.

	Elementos	**Valor Asociado**
Proceso de Desarrollo	Prácticas para cubrir el	1

	ciclo de vida completo del software.	

El coeficiente de completitud de XP para este elemento del proceso de software es 1.

Con el fin de garantizar el primer elemento del segundo aspecto del desarrollo de software XP define la siguiente práctica: El juego de la planificación.
Sin embargo XP no define la realización constante de reuniones.
Al cumplir con solamente el primero de los elementos establecidos para la evaluación cuantitativa del proceso de administración del proyecto, obtiene solamente el valor asociado con este elemento.

	Elementos	**Valor Asociado**
Proceso de Administración del Proyecto.	Planificación del proyecto.	0.75
	Realización de reuniones frecuentemente.	0

El coeficiente de completitud de XP para este elemento del proceso de software es 0.75.
XP no tiene definida ninguna práctica con el fin de garantizar el control de configuración del software.

	Elementos	**Valor Asociado**
Control de Configuración del Software	Administración de Configuración.	0

El coeficiente de completitud de XP para este elemento del proceso de software es 0.

VAD (caracterización del proceso de software) = ∑ coeficiente de completitud de coeficiente de completitud de aspecto que conforma el proceso de software - p

VAD (caracterización del proceso de software) = (1 + 0.75 + 0) =1.75 – 0 = 1.75

Dimensión 5 (Roles).

XP no define todos los roles que fueron establecidos como estándar para cualquier tipo de software. Pero algo interesante resulta el desempeño y la misión que tienen los programadores de resolver las tareas relacionadas con el análisis y diseño. Entre los roles que define XP fueron encontrados 4 que no tienen relación con los establecidos como básicos, así que estará sujeta a una penalización de

0,4.En esta dimensión cave recordar que al valor total de la suma de roles encontrados se le suma el complemento de esta dimensión que es de 0.04.

Roles Básicos.	Valor Asociado
Líder del Proyecto.	0.16
Analista	0.16
Diseñador	0.16
Programador	0.16
Arquitecto	0
Probador	0.16

VAD (Roles) = ∑ coeficiente de cada rol - p.

VAD (Roles) = 0.80 – 0.40 + 0.04 = 0.44

En el desarrollo del método se decidió aplicarle una penalización a las metodologías de desarrollo de software que establecieran más elementos de los considerados como básicos para ser ágiles. Como XP define 4 roles que no están relacionados con los establecidos, debe ser penalizada con 0.40 o sea 0.10 por cada rol de más que define.

Después de verse aplicado la penalidad a esta dimensión el grado de agilidad seria 0.09.

Agilidad Total de XP.

GAM (XP) = 1/5 ∑ VAD (Dimensión **i**)

GAM (XP) = 1/5 * (2.90 +0.62 + 3.75+ 1.75 + 0.44)

GAM (XP) = 1.89

3.2.2. Aplicación del Método a DSDM (Método de Desarrollo de Sistemas Dinámicos)

Dimensión 1 Alcance de la metodología

En la etapa de análisis DSDM, realiza varias tareas como es el caso de la obtención de un prototipo funcional que este es el resultado de la fase “Iteración del Modelo Funcional” que también lleva explicito la definición de la arquitectura que es desarrollada por el Coordinador Técnico. Los siguientes artefactos también son creados en la etapa de análisis: acuerdo de un plan, listado de riesgos, lista de funciones priorizadas, sin embargo el único de los mencionados que va de acuerdo con los elementos establecidos por el método de cálculo cuantitativo de la agilidad de las metodologías ágiles es la captura de requisitos funcionales y no funcionales que se logra a través de la Lista de Funciones Priorizadas.

Es importante hacer referencia a la forma típica de esta metodología para tratar los requerimientos.

- Debe tener: Son los requerimientos fundamentales del sistema. De estos, el subconjunto mínimo ha de ser satisfecho de forma integra.

- Debería tener: Son requerimientos esenciales para los que habrá una solución en corto plazo.

- Podría tener: Podrían quedar eliminados del sistema si no hay otra solución.

- Se desea que cuente, pero no lo tendrá en esta versión: Son los requerimientos valorados pero pueden esperar.

De los artefactos esenciales definidos para DSDM en la etapa de análisis el único que no se cumple es el Diseño de Algoritmos

Sin embargo debe ser penalizada ya que incluye la realización de tres artefactos que no se contemplan en el método como elementales para ser ágil.

DSDM	Elementos	Valor Asociado
	Requisitos Funcionales.	0.40
Análisis	Requisitos No Funcionales.	0.20
	Diseño de Algoritmos.	0
	Definición de la Arquitectura	0.30

EL coeficiente de completitud para la etapa de análisis en DSDM es 0.90 menos 0.30 por la realización de tres artefactos no contemplados entre los definidos.

En la etapa de diseño DSDM, desarrolla un prototipo de diseño y se acuerda un plan. Básicamente en el prototipo de diseño lo que se obtiene es el diseño de clases.

Al cumplir esta metodología con el elemento básico del diseño obtiene el mayor valor relacionado con la agilidad para esta etapa. Sin embargo como define algo que no está establecido para ser ágil es penalizado con 0.1. Específicamente se refiere a la elaboración del plan que se realiza en esta etapa según la filosofía de DSDM.

	Elementos	Valor Asociado
Diseño	Diagrama de Clases del Diseño.	1

El coeficiente de completitud para esta etapa es de 1 menos la penalización que es de 0.1.

La etapa de implementación en DSDM solo se ve sujeta a la obtención del sistema implementado, o sea que solo se dedican a la codificación y no hacen énfasis en la obtención de ningunos de los

modelos esenciales definidos como elementales para obtener en esta etapa. Por tanto no tendrá ninguna evaluación en esta etapa.

	Elementos	Valor Asociado
	Diagrama de Componentes.	0
Implementación	Modelo de Implementación.	0
	Subsistema de implementación.	0
	Diagrama de Despliegue	0

El coeficiente de completitud para esta etapa es de 0.

Para la etapa de pruebas, DSDM produce el manual de usuario y un reporte de revisión del sistema, este último puede ser valorado como un plan de pruebas ya que se basa en resolver los problemas que no fueron detectados durante la etapa de diseño e implementación. El objetivo de este reporte es establecer cuatro cursos de acción posible a tomar: Si el sistema satisface todos los requerimientos, el desarrollo termina, Si quedan muchos requerimientos sin resolver, se puede empezar el sistema desde el principio, Si se ha dejado alguna prestación no critica, el proceso puede correr desde la iteración funcional del modelo en adelante y por último si algunas cuestiones técnicas se pudieron resolver se puede iterar desde la fase de diseño. Es obvio que todas estas funciones del reporte de revisión del sistema constituyen la base de un plan de pruebas para resolver las cosas pendientes que quedaron durante la elaboración del producto.

Con el fin de evaluar todas las funcionalidades previstas para el producto de software final, en la fase de pos proyecto se realiza una verificación del cumplimiento de todos los requisitos establecidos y esto es realizado a través de casos de pruebas.

Después de la valoración hecha para la etapa de pruebas en DSDM se esta en condiciones de evaluar los elementos establecidos por el método para el proceso de pruebas.

Vale aclarar que por el simple hecho de DSDM producir el manual de usuario será sujeto a una penalización, ya que no está contemplado entre los elementos que se definen en esta etapa.

	Elementos	Valor Asociado
Pruebas	Casos de Pruebas.	0.50
	Plan de Pruebas.	0.50

El coeficiente de completitud para esta etapa es 1 y la penalización es de 0.1.

Valor de la agilidad de DSDM en la dimensión Alcance de la metodología.

VAD (Alcance de la metodología) = ∑ coeficiente de completitud de cada etapa - p

VAD (Alcance de la metodología) = (0.60 + 0.90 + 0 + 0.90) – 0 = 2.40

Dimensión 2 (Parámetros relacionados con la agilidad)

Los valores asociados a los indicadores de cada parámetro fueron obtenidos a partir del siguiente análisis.

Cantidad de Roles: 15:

Duración de las iteraciones: 2 semanas.

Según la puntuación de Highsmith para el indicador relacionado con el dinamismo, es de 3 puntos lo que representa el 60 % del enfoque que le da DSDM al sistema como algo cambiante en la etapa de análisis que es donde se analiza este indicador.

Para el caso de la evaluación del indicador del aprendizaje se obtuvo el siguiente resultado:

DSDM para las etapas definidas propone las siguientes fases:

Análisis: Pre proyecto, Estudio de factibilidad, Estudio del negocio e Iteración del modelo funcional. De todas estas fases la única que cumple con el indicador es la última. Por tanto solo obtiene 25 % que representa un 0.25 del valor asociado.

Diseño: DSDM para esta etapa define la fase diseño y construcción de la iteración la cual cumple con el indicador y obtiene el valor 1.

Implementación: DSDM define para esta etapa la fase de implementación y esta cumple con el indicador y obtiene el valor 1.

Pruebas: En esta etapa se define la fase del pos proyecto, que es donde se verifica las funcionalidades del software, y esta cumple con el indicador, por tanto obtiene el valor 1.

En el caso del indicador de la ligereza se tomo solamente el caso, en que el equipo de desarrollo tenga buenas habilidades. Por eso es la toma del valor 1 para el modelado de los modelos esenciales en todas las etapas definidas.

La flexibilidad aparece con el valor 1 en todas las etapas, por la definición arbitraria que se tomo para evaluar las metodologías ágiles en cuanto a su filosofía sobre el nivel de adecuación frente a los cambios.

	V	D	A	L	F
Etapas					
Análisis		0.6	0.25	1	1
Diseño		0	1	1	1
Implementación		0	1	1	1
Pruebas		0	1	0	1
Total	1.1	0.6	3.85	3	4

VAD (Parámetros relacionados con la agilidad) = ∑ AT (Análisis, Diseño, Implementación y Pruebas) / n x m.

Al aplicarle la fórmula correspondiente a esta dimensión se obtiene que su agilidad sea de 0.63.

Dimensión 3 (Valores Ágiles).

DSDM para garantizar los elementos definidos para el primer valor ágil, propone las siguientes prácticas:

Estrategia colaborativa y cooperativa entre todos los participantes.

Es imperativo el compromiso activo del usuario.

Se requiere desarrollo iterativo e incremental.

Después de ver mencionado las prácticas que garantizan el cumplimiento del valor ágil analizado, se puede concluir con la asignación del valor correspondiente para cada elemento.

	Elementos	Valor Asociado
Individuos e iteraciones por sobre los procesos y las herramientas.	Mayor valor al equipo que al entorno.	0.50
	Realización de iteraciones en todas las etapas.	0.50

Coeficiente de completitud para este valor es 1.

Con el fin de garantizar los elementos establecidos para el segundo valor ágil, DSDM define las siguientes prácticas, una para cada elemento.

Desarrollo iterativo e incremental.

Integración de pruebas a través de todo el ciclo de vida.

Entregas frecuentes del producto.

	Elementos	Valor Asociado
Software activo por encima de la documentación comprensiva.	Producción de documentos necesarios.	0.20
	Entregas frecuentes.	0.40
	Pruebas de funcionamiento al software	0.40

Coeficiente de completitud para este valor es 1.

DSDM define para el cumplimiento del tercer valor ágil la siguiente práctica:

Estrategia colaborativa y cooperativa entre todos los participantes. Por tanto tiene el valor asociado a este elemento.

	Elementos	Valor de Asociado
La colaboración con el cliente más que la negociación de un contrato.	Frecuente relación con el cliente.	1

Coeficiente de completitud para este valor es 1.

DSDM define para el cumplimiento del cuarto valor ágil la siguiente práctica:

La línea de base de los requerimientos es de alto nivel. Esto permite que los requerimientos de detalle se cambien según se necesite y que los esenciales se capten tempranamente. Por tanto tiene el valor asociado a este elemento. Sin embargo no cuenta con una práctica para garantizar una planificación flexible.

	Elementos	Valor Asociado

Responder a los cambios más que seguir estrictamente un plan.	Estrategia para responder a los cambios.	0.75
	Planificación flexible.	0

Coeficiente de completitud para este valor es 0.75.

VAD (valores ágiles) = ∑ coeficiente de completitud de cada valor ágil - p.

VAD (valores ágiles) = 3.75 – 0 = 3.75

Dimensión 4 (Caracterización del proceso de software).

Con el fin de cubrir el ciclo de vida del software de forma completa DSDM, propone las siguientes prácticas:

- Es imperativo el compromiso activo del usuario.
- Los equipos DSDM deben tener el privilegio de tomar decisiones.
- El foco radica en la frecuente entrega del producto.
- Se requiere desarrollo iterativo e incremental.
- Todos los cambios durante el desarrollo son irreversibles.
- La línea base de los requerimientos es de alto nivel.
- La prueba está integrada a través de todo el ciclo de vida.
- Es esencial una estrategia colaborativa y cooperativa entre todos los participantes.

Al poseer prácticas para cubrir el ciclo de vida del software, obtiene el máximo valor definido para este elemento.

	Elementos	Valor Asociado
Proceso de Desarrollo	Prácticas para cubrir el ciclo de vida completo del software.	1

El coeficiente de completitud de este elemento es 1.

DSDM no define una práctica específica para la planificación del proyecto, sin embargo tiene una constante frecuencia de reuniones y talleres donde básicamente se capturan los requisitos y se analizan todos los problemas relacionados con el producto.

	Elementos	Valor Asociado
Proceso de Administración del Proyecto.	Planificación del proyecto.	0
	Realización de reuniones frecuentemente.	0.25

El coeficiente de completitud para los elementos establecidos dentro del proceso de administración del proyecto es de 0.25

DSDM no propone prácticas para el elemento establecido en el control de configuración del software.

	Elementos	Valor Asociado
Control de Configuración del Software	Administración de Configuración.	0

VAD (caracterización del proceso de software) = ∑ coeficiente de completitud de los aspecto que conforma el proceso de software - p

VAD (caracterización del proceso de software) = 1.25 – 0 = 1.25

Dimensión 5 (Roles).

DSDM define todos los roles definidos como básicos, es importante aclarar en esta dimensión que todos los roles de desarrollo son cubiertos por programadores Senior, son aquellos con capacidad para trabajar tanto en el diseño, el análisis, en las pruebas y las tareas de programación. Sin embargo La evaluación de esta dimensión esta sujeta a una penalización tan grande que deja sin valor a esta dimensión.

Roles Básicos.	Valor Asociado
Líder del Proyecto.	0.16
Analista	0.16
Diseñador	0.16
Programador	0.16
Arquitecto	0.16
Probador	0.16

El coeficiente de completitud es 0. Al restar 1 que es el valor obtenido por el cumplimiento de roles básicos menos las penalizaciones da un valor negativo, pero este carece de sentido ya solo el método esta realizado para tener como mínimo valor cero.

VAD (Roles) = ∑ coeficiente de cada rol - p

VAD (Roles) = 1 – 1.50 = 0

Agilidad Total de DSDM.

GAM (DSDM) = 1/5 ∑ VAD (Dimensiones)

GAM (DSDM) = 1/5 * (2.40 + 0.63 + 3.75+ 1.25 +0)

GAM (DSDM) = 1.61

3.2.3. Aplicación del Método a SCRUM

Antes de iniciarse con la aplicación del método a SCRUM, vale la pena aclarar que esta metodología no esta diseñada para cubrir una producción de un producto de software de forma integra, más bien es usado para gestionar proyectos de software.

SCRUM en la etapa de análisis lo único que produce es El Backlog del Producto.

Backlog del Producto: Es el inventario de funcionalidades, mejoras, tecnología y corrección de errores que deben incorporarse al producto de software a través de las sucesivas iteraciones. Representa todo aquello que esperan los clientes, usuarios y en general los interesados en el producto. Después de la descripción de las funcionalidades del Backlog del Producto se esta en condiciones de asegurar que este garantiza la obtención de los requisitos funcionales y no funcionales del sistema así como una definición de la arquitectura, ya que se deben tener bien definidos todo los aspectos relacionados con la tecnología y un esbozo de las interfaces por la que estará compuesta el sistema. Al cumplir con la definición de estos elementos que son considerados por el método de evaluación. Entonces obtiene

el valor asociado con cada uno de estos y no esta sujeta a ninguna penalización ya que cumple estrictamente con lo establecido para la etapa de análisis menos el diseño de algoritmos.

El coeficiente de completitud de la etapa de análisis para SCRUM es 0.90.

	Elementos	Valor Asociado
	Requisitos Funcionales.	0.40
Análisis	Requisitos No Funcionales.	0.20
	Diseño de Algoritmos.	0
	Definición de la Arquitectura.	0.30

SCRUM para la etapa de Diseño no define ningún artefacto. Por tanto el coeficiente de completitud es 0.

	Elementos	Valor Asociado
Diseño	Diagrama de Clases del Diseño.	0

Al igual que en la etapa de diseño SCRUM no realiza ninguno de los elementos establecidos como fundamentales para ser ágil en la implementación. Solo se enfoca en la codificación a través del Sprint Backlog que es la lista que descompone las funcionalidades del Backlog del Producto que serán implementadas.

El coeficiente de completitud para la etapa de implementación en SCRUM es 0.

	Elementos	Valor Asociado
	Diagrama de Componentes.	0
Implementación	Modelo de Implementación.	0
	Subsistema de implementación.	0
	Diagrama de Despliegue	0

SCRUM para la etapa de pruebas no define los elementos que son considerados en esta etapa. Las pruebas al software la trata de realizar en la realización del Sprint y las constantes iteraciones que se realizan a lo largo del proyecto. Por tanto el coeficiente de completitud para la etapa de pruebas es 0.

	Elementos	Valor Asociado
Pruebas	Casos de Pruebas.	0
	Plan de Pruebas.	0

Valor de la agilidad de SCRUM en la dimensión Alcance de la metodología.

VAD (Alcance de la metodología) = ∑ coeficiente de completitud de cada etapa - p.

VAD (Alcance de la metodología) = (0.90 + 0 + 0 + 0) – 0

VAD (Alcance de la metodología) = 0.90

Dimensión 2 (Atributos para obtener la agilidad)

Los valores de los indicadores relacionados con la velocidad fueron obtenidos a partir del siguiente análisis:

Cantidad de Roles: 5: Scrum Master (Jefe del Equipo), Product Owner (Usuario interno o gerente), Scrum Team (Equipo), Customer (Cliente) y Management (Gerente).

Duración de las iteraciones: 4 semanas.

La puntuación que le asigna Highsmith a SCRUM en lo referente al sistema como algo cambiante es de 5 puntos lo que representa el 100 % y el valor del grado de agilidad para la etapa de análisis es 1.

Para el indicador relacionado con el aprendizaje solo cumplen con lo establecido por el método para este parámetro la etapa de análisis que define en SCRUM el Pre - Juego y en la etapa de Implementación define el desarrollo, ambas cumplen con la creación de un ambiente de aprendizaje y para el caso del desarrollo actualiza el conocimiento anterior. Por tales motivos solo reciben el valor asociado para este indicador las etapas antes mencionadas.

El caso del indicador relacionado con la ligereza está evaluado a consideración del evaluador y este caso se determino el valor 1.

Para la flexibilidad se tomo el valor definido por el método para este parámetro en todas las etapas definidas.

	V	D	A	L	F
Etapas					
Análisis		1	1	1	1
Diseño		0	0	1	1
Implementación		0	1	1	1
Pruebas		0	0	0	1
Total	2	1	2	3	4

VAD (Parámetros relacionados con la agilidad) = ∑ AT (Análisis, Diseño, Implementación y Pruebas) / n x m.

Al aplicarle la fórmula correspondiente a esta dimensión se obtiene que su agilidad sea de 0.6.

Dimensión 3 (Valores Ágiles).

Con el fin de garantizar un mayor valor al equipo que al entorno de trabajo SCRUM propone las siguientes prácticas: Planificación del Sprint y las Reuniones Diarias y para la realización de iteraciones en todas las etapas lo define en su filosofía de iterar cada vez que se obtenga un Sprint del producto. Al garantizar prácticas para los dos elementos definidos en este valor ágil, obtiene el valor asociado a ambos.

	Elementos	Valor Asociado
Individuos e iteraciones por sobre los procesos y las herramientas.	Mayor valor al equipo que al entorno.	0.50
	Realización de iteraciones en todas las etapas.	0.50

SCRUM para garantizar los elementos definidos para el segundo valor ágil, propone las siguientes prácticas:

Sprint y las Revisiones del Sprint estas son para la entrega frecuente y las pruebas de funcionamiento del software. Sin embargo no propone prácticas que garanticen la producción de los documentos necesarios, pero si en su filosofía define la obtención de estos.

Al garantizar los elementos definidos para este valor, obtiene el máximo coeficiente de completitud que es 1.

	Elementos	Valor Asociado
Software activo por encima de la documentación comprensiva.	Producción de documentos necesarios.	0.20
	Entregas frecuentes.	0.40
	Pruebas de funcionamiento al software	0.40

La frecuente relación con el cliente en SCRUM se logra a través de las siguientes prácticas:

Backlog del Producto.

El coeficiente de completitud para este valor ágil es 1.

	Elementos	Valor de Rango
La colaboración con el cliente más que la negociación de un contrato.	Frecuente relación con el cliente.	1

La practica encargada de responder a los cambios El Planeamiento del Sprint. Para el segundo elemento no define prácticas, por tanto solo obtiene el valor asociado con el primero y su coeficiente de completitud es 0.75.

	Elementos	Valor Asociado
Responder a los cambios más que seguir estrictamente un plan.	Estrategia para responder a los cambios.	0.75
	Planificación flexible.	0

VAD (valores ágiles) = ∑ coeficiente de completitud de cada valor ágil - p.

VAD (valores ágiles) = 3.75 – 0 = 3.75

Dimensión 4 (Caracterización del proceso de software).

Practicas definidas por SCRUM para cubrir el ciclo de vida del software:

Equipo SCRUM, Backlog del Producto, Sprint y Revisión del Sprint.

Como cumple con el elemento definido para el proceso de desarrollo obtiene el valor 1.

	Elementos	Valor Asociado
Proceso de Desarrollo	Prácticas para cubrir el ciclo de vida completo del software.	1

SCRUM define para la planificación del proyecto la fase de Planeamiento y el segundo elemento la práctica de Realización Diarias de reuniones.

Al cumplir con los dos elementos establecidos para el Proceso de Administración, la evaluación es la máxima o sea 1.

	Elementos	Valor Asociado
Proceso de Administración del Proyecto.	Planificación del proyecto.	0.75
	Realización de reuniones frecuentemente.	0.25

Para el control de configuración no define ninguna práctica, por tanto el valor establecido para el correspondiente a este aspecto es 0.

	Elementos	Valor Asociado
Control de Configuración del Software	Administración de Configuración.	0

VAD (caracterización del proceso de software) = ∑ coeficiente de completitud de los aspecto que conforma el proceso de software - p

VAD (caracterización del proceso de software) = 2 – 0 = 2

Dimensión 5 (Roles).

En el análisis realizado para la verificación de los roles definidos por SCRUM en cuanto a los establecidos como básicos para el desarrollo de un software, fueron identificados las siguientes relaciones:

SCRUM Master cumple la función del Líder de Proyecto.

Producto Owner cumple la función de Analista.

Algo típico de esta metodología es la definición del equipo Scrum el cual lo integran Programadores, Diseñadores de Interfaz y Probadores es importante aclarar que la composición del equipo puede estar formada de 5 a 10 integrantes. Entre los roles no definidos como básicos se encuentran el Cliente y el Gerente lo cual provocarían una penalización de 0.20 entre ambos.

Roles Básicos.	Valor Asociado
Líder del Proyecto.	0.16
Analista	0.16
Diseñador	0.16
Programador	0.16
Arquitecto	0
Probador	0.16

VAD (Roles) = ∑ coeficiente de cada rol - p.

VAD (Roles) =0.80 – 0.20 + 0,04

Agilidad Total de SCRUM.

GAM (SCRUM) = 1/5 ∑ VAD (Dimensiones)

GAM (SCRUM) = 1/5 * (0.90+0.60 + 3.75+ 2 + 0, 64)

GAM (SCRUM) = 1.58

3.2.4. Aplicación del Método a RUP (Proceso Unificado del Rational).

Dimensión 1 Alcance de la metodología

En el riguroso proceso que define RUP, para la etapa de análisis se puede incluir la fase de inicio donde se producen los siguientes artefactos: Modelado del Negocio, Modelo de Casos de Uso del Negocio, Modelo de Objetos, Objetos del Negocio, Requisitos, Glosario de Términos, Documento Visión, Modelos de Casos de Usos, Especificación de los Casos de Uso, Reglas del negocio, Listado de Riesgos y Diagrama de Actividades. En lo referente con la arquitectura se enfoca a la vista de casos de usos en esta etapa.

Entre todos estos artefactos solo pueden ser evaluados según los elementos que establece el método para la etapa de análisis; el de los Requisitos que es donde se incluyen los Funcionales y los No Funcionales y la definición de la arquitectura. Por tanto al seleccionar los elementos a evaluar en la dimensión 1, solo será tenido en cuenta la definición de requisitos y por la generación de los demás artefacto será sujeto a la penalización cada uno de ellos.

	Elementos	Valor Asociado
	Requisitos Funcionales.	0.40
Análisis	Requisitos No Funcionales.	0.20

	Diseño de Algoritmos.	0
	Definición de la Arquitectura	0.30

El coeficiente de completitud de RUP para esta es 0.90 pero como esta sujeta a tantas penalizaciones se queda con el valor 0.

En la etapa de diseño RUP define los siguientes artefactos: Modelo de Anales/Diseño y el Modelo de Datos. Por lo que obtendrá solamente la evaluación del primero, ya que el segundo no esta contemplado en el elemento que define el método para esta etapa.

	Elementos	**Valor Asociado**
Diseño	Diagrama de Clases del Diseño.	1

El coeficiente de completitud para esta etapa en RUP es de del máximo valor, o sea 1 sin embargo esta sujeta a una penalización de 0.1 por la generación del Modelo de datos que no esta valorado como elemental para ser ágil en la diseño. Por tanto su coeficiente final para esta etapa es 0.90

Para la implementación RUP cumple con todos los artefactos definidos en el método. Por tanto obtiene el valor asignado a cada elemento.

	Elementos	**Valor Asociado**
	Diagrama de Componentes.	0.25
Implementación	Modelo de Implementación.	0.25
	Subsistema de implementación.	0.25
	Diagrama de Despliegue	0.25

En esta etapa el coeficiente de completitud para RUP es de 1. Ya que cumple con todos los elementos establecidos para la etapa y no esta sujeta a ninguna penalización.

En la etapa de pruebas RUP desarrollo los siguientes artefactos:

- El plan de Pruebas.
- La estrategia de prueba.
- Los casos de pruebas.
- Los Procedimientos de prueba.

- El listado de ideas de prueba.
- El listado de datos de prueba.
- El resumen de la evaluación de las pruebas.

Dentro del riguroso proceso que realiza para la etapa de pruebas encontramos los dos artefactos que fueron definidos por el método para la agilidad en esta etapa es el caso de: El plan de pruebas y Los casos de pruebas. Los demás artefactos estarán sujetos a la penalización

	Elementos	Valor Asociado
Pruebas	Casos de Pruebas.	0.50
	Plan de Pruebas.	0.50

El coeficiente de RUP en la etapa de prueba es el máximo, sin embargo es penalizada con 0.50 por el hecho de incluir 5 artefactos que no son contemplados como elementales dentro de lo que establece esta dimensión del método. Por tanto obtiene un coeficiente de completitud de 0.50 como final.

Valor de la agilidad de RUP en la dimensión Alcance de la metodología.

VAD (Alcance de la metodología) = ∑ coeficiente de completitud de cada etapa - p

VAD (Alcance de la metodología) = (0 + 0.90+ 1 + 0.50) = 2.4

Dimensión 2 (Parámetros relacionados con la agilidad)

Los valores asociados a los indicadores de cada parámetro fueron obtenidos a partir del siguiente análisis.

Análisis para la obtención de los valores del parámetro relacionado con la velocidad en RUP:

RUP define 14 roles ellos son: Analista de procesos del negocio, Diseñador del negocio, Revisor del modelo del negocio (arquitecto), Analista del Sistema, Arquitecto, Especificador, Diseñador de Interfaces, Implementador, Integrador, Revisor, Analista de prueba., Diseñador de prueba., Administrador de prueba y Probador.

Como define 14 roles el valor asociado para esta cantidad es 0.1

El otro indicador del parámetro de la velocidad es, la duración de las iteraciones, pero RUP no define estrictamente la duración de estas y solo lo deja a consideración del equipo de desarrollo. Por tanto no se le define valor a este indicador.

A la sumatoria de los valores obtenidos en cada etapa se le suma el valor asociado a la cantidad de roles que en el caso de RUP es 0.1 y la duración de las iteraciones que no se define.

El caso del dinamismo no se tendrá presente, ya que el análisis realizado para este parámetro estuvo basado sobre la puntuación que le da Highsmith al sistema como algo cambiante pero solo tuvo en cuenta las metodologías ágiles.

Para el caso del aprendizaje se decide darle el máximo valor en cada etapa, ya que RUP provee un detallado proceso de en cada etapa y esto como resulta aporta un nivel de asimilación y aprendizaje constante entre las fases.

El indicador relacionado con la flexibilidad esta evaluado de 0 en todas las etapas, ya que RUP no tiene en su filosofía darle la bienvenida a los cambios, a partir de que ella trata de definir con exactitud lo que realmente desea el cliente en la etapa de análisis y después se le hace un poco tedioso recibir los cambios que quiera realizar el cliente en el transcurso del desarrollo del software.

	V	D	A	L	F
Etapas					
Análisis		0	1	1	0
Diseño		0	1	1	0
Implementación		0	1	1	0
Pruebas		0	1	1	0
Total	0.1	0	4	4	0

VAD (Parámetros relacionados con la agilidad) = ∑ AT VAD (Análisis, Diseño, Implementación y Pruebas) / n x m.

Al aplicarle la fórmula correspondiente a esta dimensión se obtiene que su agilidad sea de 0.41.

Dimensión 3 (Valores Ágiles).

RUP un cumple con ninguno de los valores ágiles definidos en esta dimensión. Ya que estos fueron definidos para el cumplimiento de las metodologías ágiles. Por tanto el grado de agilidad en esta dimensión es 0.

Dimensión 4 (Caracterización del proceso de software).

RUP para el proceso de desarrollo de software tiene garantizada las prácticas suficientes para cubrir de forma integra el proceso de software completo, las siguientes son:

Desarrollo de Software Iterativo

En función de la cada vez mayor complejidad solicitada para los sistemas de software, ya no es posible trabajar secuencialmente: definir primero el problema completo, luego diseñar toda la solución, construir el software y finalmente, testear el producto. Es necesario un enfoque iterativo, que permita una comprensión creciente del problema a través de refinamientos sucesivos, llegando a una solución efectiva luego de múltiples iteraciones acotadas en complejidad.

RUP utiliza y soporta este enfoque iterativo que ayuda a atacar los riesgos mediante la producción de raleases ejecutables progresivos y frecuentes que permiten la opinión e involucramiento del usuario.

A través de las iteraciones que generan raleases ejecutables, se logra detectar en forma temprana los desajustes e inconsistencias entre los requerimientos, el diseño, el desarrollo y la implementación del sistema, manteniendo al team de desarrollo focalizado en producir resultados.

Administrar los Requerimientos

Los requerimientos son las condiciones o capacidades que el sistema debe conformar. La Administración de Requerimientos es un enfoque sistemático para hallar, documentar, organizar y monitorear los requerimientos cambiantes de un sistema.

Arquitectura basada en Componentes

El proceso de software debe focalizarse en el desarrollo temprano de una arquitectura robusta ejecutable, antes de comprometer recursos para el desarrollo en gran escala. RUP describe como diseñar una arquitectura flexible, que se acomode a los cambios, comprensible intuitivamente y promueve una más efectiva reutilización de software. Soporta el desarrollo de software basado en componentes: módulos no triviales que completan una función clara. RUP provee un enfoque sistemático para definir una arquitectura utilizando componentes nuevos y preexistentes.

Modelado Visual del Software

RUP muestra como modelar el software visualmente para capturar la estructura y comportamiento de arquitecturas y componentes. Las abstracciones visuales ayudan a comunicar diferentes aspectos del software; comprender los requerimientos, ver como los elementos del sistema se relacionan entre sí,

mantener la consistencia entre diseño e implementación y promover una comunicación precisa. El estándar UML (Lenguaje de Modelado Unificado), creado por Rational Software, es el cimiento para una modelización visual exitosa.

Verificación de la Calidad del Software

Es necesario evaluar la calidad de un sistema respecto de sus requerimientos de funcionalidad, confiabilidad y performance. La actividad fundamental es el testing, que permite encontrar las fallas antes de la puesta en producción. RUP asiste en el planeamiento, diseño, implementación, ejecución y evaluación de todos estos tipos de testing.

El aseguramiento de la calidad se construye dentro del proceso, en todas las actividades, involucrando a todos los participantes, utilizando medidas y criterios objetivos, permitiendo así detectar e identificar los defectos en forma temprana.

Control de los Cambios del Software

La capacidad de administrar los cambios es esencial en ambientes en los cuales el cambio es inevitable. RUP describe como controlar, rastrear y monitorear los cambios para permitir un desarrollo iterativo exitoso. Es también una guía para establecer espacios de trabajo seguros para cada desarrollador, suministrando el aislamiento de los cambios hechos en otros espacios de trabajo y controlando los cambios de todos los elementos de software (modelos, código, documentos, etc.). Describe como automatizar la integración y administrar la conformación de raleases.

Después de ver explicado cada una de las práctica que provee RUP para el ciclo de vida completo de software se esta en condiciones de asignarle el valor máximo para el elemento definido en el proceso de desarrollo.

	Elementos	Valor Asociado
Proceso de Desarrollo	Prácticas para cubrir el ciclo de vida completo del software.	1

En RUP la planificación del proyecto es más eficiente que en la mayoría de las metodologías de desarrollo de software existentes, a pesar no establecer prácticas para esta tan importante tarea. Sin embargo garantiza tres fases en el nivel 2 de madurez de CMMI, estas son: Establecer las Estimaciones, Desarrollar el Plan del Proyecto y Obtener el compromiso del Plan. Por riguroso énfasis

que realiza RUP para la planificación del proyecto se le asigna el valor correspondiente a este elemento.

El otro elemento analizado en el proceso de administración del proyecto es la realización constante de reuniones, pero RUP no tiene definido en su estrategia la realización de reuniones como otras metodologías como es el caso de SCUM y XP, por tanto el valor para este elemento es 0.

	Elementos	Valor Asociado
Proceso de Administración del Proyecto.	Planificación del proyecto.	0.75
	Realización de reuniones frecuentemente.	0

A pesar de ser unos de los flujos de trabajo menos implementados de RUP, la administración de configuración tiene los siguientes propósitos: Soportar métodos de desarrollo, mantener la integridad del producto, Asegurar que el producto configurado esté completo y correcto, Restringir los cambios a los artefactos basados en la política del proyecto y por ultimo proveer pistas de auditoria de cambios a los artefactos por qué, cuándo y por quién. Por el énfasis de que realiza RUP para garantizar el control de configuración del software se le asigna el valor correspondiente al elemento establecido, en este caso 1.

	Elementos	Valor Asociado
Control de Configuración del Software	Administración de Configuración.	1

VAD (caracterización del proceso de software) = ∑ coeficiente de completitud de coeficiente de completitud de aspecto que conforma el proceso de software - p

VAD (caracterización del proceso de software) = (1 + 0.75 + 1) =2.75 – 0 = 2.75

Dimensión 5 (Roles).

RUP define todos roles que se definen como elementales a tener presente en el desarrollo de de cualquier tipo de software, sin embargo debe ser penalizada por la definición de otros roles que no son tenidos en cuente a la hora de desarrollar un software bajo filosofías ágiles. Estos roles adicionales son 8 lo cual se le aplicará una penalización de 0.1 a cada uno. El valor 0.04 corresponde al complemento de 1.

Roles Básicos.	Valor Asociado
Líder del Proyecto.	0.16
Analista	0.16

Diseñador	0.16
Programador	0.16
Arquitecto	0.16
Probador	0.16

VAD (Roles) = ∑ coeficiente de cada rol - p.

VAD (Roles) = 0.96 – 0.8 = 0.16 + 0.04 = 0.2

Agilidad Total de RUP.

GAM (RUP) = 1/5 ∑ VAD (Dimensión **i**)

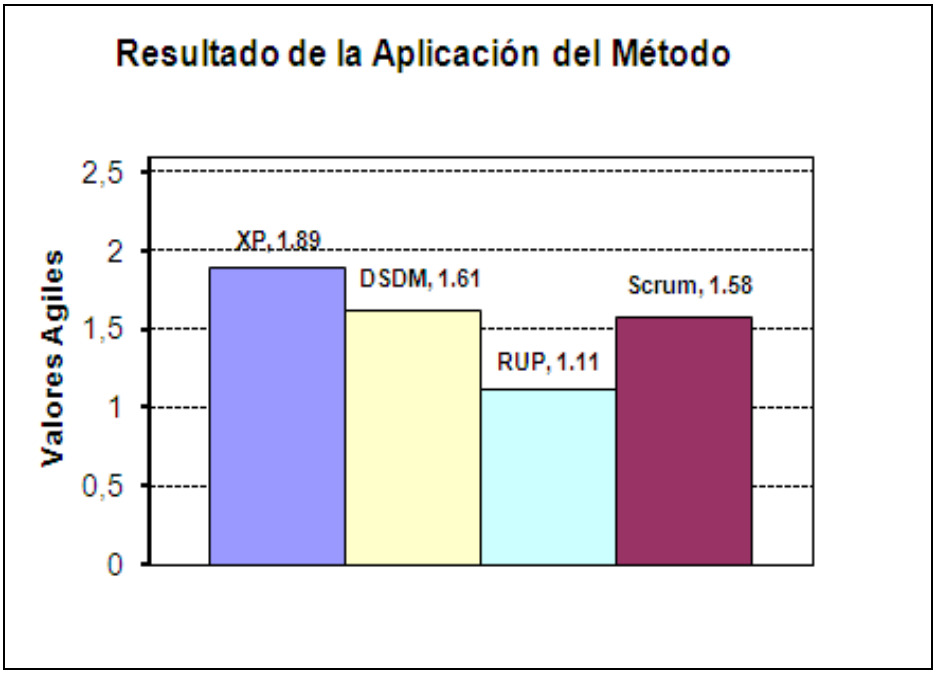
GAM (RUP) = 1/5 * (2.4 +0.41 + 0+ 2.75 + 0.2)

GAM (RUP) = 1.11

3.2.5. Límites del Método Cuantitativo para La Evaluación de Metodologías Ágiles

El máximo valor que se puede obtener con la aplicación del método es 2.6, este se basa en la metodología ideal según los elementos establecidos en cada una de las dimensiones del método. Mientras que el menor resultado que puede arrojar el método es 0, esto implica que mientras más cerca este el procedimiento metodológico de este, más riguroso será su proceso de software.

3.2.6. Resultados de la aplicación del método



3.3. Valoración de los resultados de la aplicación del método.

De los valores obtenidos con la aplicación del método vale la pena resaltar el de RUP, visto desde un enfoque ágil. Es importante la valoración de lo ajustable que puede convertirse esta metodología pesada a los entornos de desarrollo de software de forma ágil, solo que el equipo de software debe ajustarlo a sus expectativas y olvidarse de la producción de muchos artefactos que realiza y que pueden entorpecer un desarrollo rápido de software.

En lo referente a las Metodologías Ágiles que le fue aplicado el método, es importante la reflexión de lo alejadas que se encuentran del límite máximo definido para el método ya que su proceso de desarrollo en el software se ve ajustado a la determinación de pocos artefactos en las etapas esenciales del ciclo de vida del producto de software. Es obvio que esto puede ser parte del fracaso de un sistema si se opta por una de estas metodologías sin tener previsto la esencia para la cuales fueron diseñadas.

Los valores obtenidos en el método también pueden servir de base para la adopción de las prácticas y las fases que realmente le convengan a un determinado equipo de desarrollo de software de una metodología en específico que responda a la agilidad que realmente se desee de una determinada etapa del ciclo de vida del software.

Una evaluación interesante de los resultados obtenidos, sería el análisis de los valores arrojados por las metodologías evaluadas en cada dimensión. El contar con el valor de la metodología más ágil es de vital importancia desde la perspectiva de que se analiza el proceso de software completo desde el punto de vista de los elementos que conforman a cada una de las dimensiones evaluadas. Pero algo interesante resulta el saber valorar los distintos valores obtenidos en las dimensiones, ya que pueden servir de punto de partida para la selección de una determinada dimensión para ser aplicada al entorno típico de cada cual.

La valoración de los resultados obtenidos en cada una de las metodologías analizadas, también estuvo sujeto a un análisis estadístico a través de un software especializado SPSS (Statistical Product and Service Solutions) por sus siglas en ingles. La tabla de salida resultante después de la corrida de los datos obtenidos en cada una de las dimensiones arroja que no hay diferencias significativas entre las muestras, la discriminación más significativa entre las Metodologías Ágiles y las Tradicionales se

encuentran en la dimensión 3 según los algoritmos estadísticos con los que cuenta SPSS. (Ver Anexo 5) donde se muestran los resultados obtenidos después de la corrida de los datos.

Para ver los resultados obtenidos en cada una de las dimensiones para las metodologías evaluadas (Ver Anexo 6).

Como se ha visto en este capítulo, la aplicación del método para la evaluación cuantitativa de metodologías de desarrollo de software ágiles, resulta un punto de partida para un posible análisis a la hora de seleccionar el procedimiento metodológico para un proyecto. Los valores obtenidos, en gran medida brindan la posibilidad de examinar a profundidad los elementos que realmente se deben tener en cuenta para el desarrollo ágil de un determinado producto de software.

Otro aspecto importante que se logra con la aplicación del método, es que brindaría la posibilidad de zafarse de algunas ataduras relacionadas con las concepciones propias de los autores que abordan el tema de de las Metodologías Ágiles de una forma cualitativa o sea que solo se limitan a escribir y opinar de estas desde el punto de vista filosófico sin aterrizar los verdaderos elementos que caracterizan un proceso de software ágil.

CONCLUSIONES

En la presente tesis se realizó un método para la evaluación cuantitativa de Metodologías Ágiles, el cual se propone que sea tenido en cuenta para definir que tan ágil es una metodología de desarrollo de software a partir de las dimensiones establecidas, que en esencia evalúan la agilidad en los aspectos más significativos relacionados con el desarrollo de un software de forma rápida, lográndose además:

- Caracterizar las Metodologías Ágiles más reconocidas a nivel internacional.
- La definición de las 5 dimensiones para calcular el grado de agilidad de las Metodologías Ágiles.
- La definición de ecuaciones basadas en los coeficientes de completitud de cada dimensión.
- La aplicación del método a las Metodologías Ágiles más referenciadas y aplicadas en la actualidad.
- Una valoración de los resultados obtenidos con la aplicación del método.

RECOMENDACIONES

A partir de los resultados obtenidos, en la presente tesis se recomienda:

- Darle un seguimiento a está investigación, para una posible profundización del método obtenido con el fin de incorporarle otros elementos que sean de consideración para un enfoque ágil entre ellos definir con exactitud los valores asociados al indicador relacionado con el parámetro ligereza de la dimensión 2, partiendo de la semántica propia del lenguaje de modelado que usen en su entorno de desarrollo.

- Se le recomienda a La Universidad de las Ciencias Informáticas y en especial a La Facultad 3, que diseñen cursos optativos donde puedan ser evaluadas las metodologías ágiles de desarrollo de software a partir del método realizado, para que los estudiantes y profesores interesados en el tema puedan observar de forma objetiva que tan ágil son los procedimientos metodológicos existentes.

REFERENCIA BIBLIOGRAFICA

BIBLIOGRAFÍA

1. Alianza Agil. 2004 [cited; Available from: www.agileallince.org.
2. El Manifiesto Agil [cited; Available from: http://www.sdmagazine.com.
3. Petroski, H., La Evolucion de las Cosas Ultiles. 1994: Ventage Books.
4. Gallo, E., M.V... 2006.
5. Scrum Gathering. 2005 [cited; Available from: http://www.agileadvice.com/archives/2005/05/scrum_gathering.html.
6. Beck, K., “Extreme Programming Explained. Embrace Change“. 1999.
7. Schwaber K., B.M., Martin R.C. , “Agile Software Development with SCRUM”. . 2001.
8. Pekka Abrahamsson, O.S., Jussi Ronkainen y Juhani Warsta. , Agile Software Development Methods. 2002.
9. Highsmith, J., "Agile Software Development Ecosystems". . 2002.
10. Cockburn, A., Crystal Clear. 2002.
11. Consultores, P. ¿Agile o Unified? 2004 [cited; Available from: http://www-2.dc.uba.ar/materias/isoft2/invitados/sansano.pdf.
12. A.Quemer, B.H. (2007) An evaluation of the degree of agility in six agile methods and its applicability for method engineering. **Volume**,
13. Juan Gardini, L.C. Balanceo de Metodologías Orientadas al Plan y Ágiles 2004 [cited; Available from: http://www.fi.uba.ar/materias/7546/material/BalanceoAgilPlan.pdf.
14. Myers, G., The Art of Software Testing 1979.

BIBLIOGRAFIA

Reynoso, C. “Introducción a la Arquitectura de Software”.Marzo 2004. Disponible en : http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx.

Reynoso, C. “Métodos Heterodoxos en Desarrollo de Software”. Abril 2004. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/heterodox.mspx.

Palacio, J. “Flexibilidad con Scrum”. Disponible en: http://www.navegapolis.net/files/Flexibilidad_con_Scrum.pdf.

Beck, J., Andres, C.” Extreme Programming Explanided”. 2004.

Schwaber, K. “Agile Project Management with Scrum”. 2004.

Pressman, R. “Ingeniería de Software un Enfoque Practico”. Disponible en: http://bibliodoc.uci.cu/pdf/reg02689.pdf.

Fowler, M. “Is design Dead?”2001. Disponible en: www.martinfowler.com/article/designDead.html.

Cockbun, A., Williams, L. “The Cost and Benefits of Pair Programming”. 2000.

Petroski, H.”La Evolución de las Cosas Útiles”. 1994.

Dyba, T., Dingseyr, T. “Empirical studies of agile software development: A systematic review”. Enero 2008. Disponible en: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0B-4RRFN8D-1&_user=2342189&_origUdi=B6V0B-4N1JRNN-3&_fmt=high&_coverDate=02%2F02%2F2008&_rdoc=1&_orig=article&_acct=C000056883&_version=1&_urlVersion=0&_userid=2342189&md5=8ca61dea607a5d5a246588c87359caa4.

Quemer, A., Henderson, B.” An evaluation of the degree of agility in six agile methods and its applicability for method engineering”. 2007. Disponible en : http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0B-4N1JRNN-3&_user=2342189&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000056883&_version=1&_urlVersion=0&_userid=2342189&md5=a71f2dbbe8e8b1d071e68220f9b19c14#secx13.

Reynoso, C.”Métodos Heterodoxos en Desarrollo de Software”. Abril 2004. Disponible en: http://ima.udg.edu/Docencia/3105200728/SurveyMetAgil.pdf.

BIBLIOGRAFIA ONLINE

www.controlchaos.com

www.crystalmethodologies.org

www.dsdm.org

www.adaptivesd.com

www.featuredrivendevelopment.com

www.poppendieck.com

www.sei.cmu.edu/cmm/cmm.html

www.extremeprogramming.org,

www.xprogramming.com, c2.com/cgi/wiki?ExtremeProgramming

http://www.topdownsoftware.com/Practitioners/Life_Cycle_Models/AgileMethods.html

ANEXOS

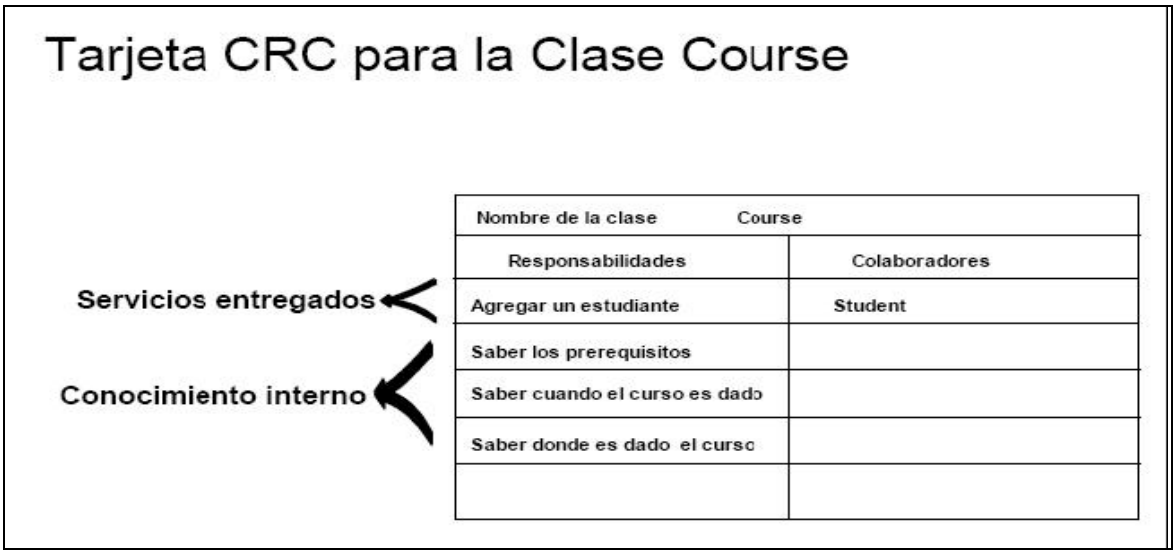
Anexo 1 Evaluación de Highsmith para el sistema como algo cambiante.

	CMM	ASD	Crystal	DSDM	FDD	LD	Scrum	XP
Sistema como algo cambiante	1	5	4	3	3	4	5	5

Anexo 2 Plantilla de Historia de Usuarios

Historia de Usuario	
Número:	Nombre Historia de Usuario:
Modificación de Historia de Usuario Número:	
Usuario:	Iteración Asignada:
Prioridad en Negocio: (Alta / Media / Baja)	Puntos Estimados:
Riesgo en Desarrollo: (Alto / Medio / Bajo)	Puntos Reales:
Descripción:	
Observaciones:	

Anexo 3 Tarjeta CRC (Ejemplo)



Anexo 4 Plantilla de Caso de Prueba

Caso de Prueba de Aceptación	
Código Caso de Prueba:	Número Historia de Usuario:
Descripción de la Prueba:	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución:	
Resultado Esperado:	
Evaluación de la Prueba:	

Anexo 5 (Salida de SPSS)

NPar Tests

Notes

Output Created		23-MAY-2008 09:18:44
Comments		
Input	**Filter**	<none>
	Weight	<none>
	Split File	<none>
	N of Rows in Working Data File	4
Missing Value Handling	**Definition of Missing**	User-defined missing values are treated as missing.
	Cases Used	Statistics for each test are based on all cases with valid data for the variable(s) used in that test.
Syntax		NPAR TESTS /M-W= dimension1 dimension2 dimension3 dimension4 dimension5 BY Tipo(1 2) /MISSING ANALYSIS /METHOD=MC CIN(99) SAMPLES(10000).
Resources	**Elapsed Time**	0:00:00.14
	Number of Cases Allowed(a)	47662
	Time for Exact Statistics	0:00:00.12

a Based on availability of workspace memory.

Mann-Whitney Test

Ranks

	Tipo	N	Mean Rank	Sum of Ranks
dimension1	1.00	3	2.33	7.00
	2.00	1	3.00	3.00
	Total	4		
dimension2	1.00	3	3.00	9.00
	2.00	1	1.00	1.00
	Total	4		
dimension3	1.00	3	3.00	9.00
	2.00	1	1.00	1.00
	Total	4		
dimension4	1.00	3	2.00	6.00
	2.00	1	4.00	4.00
	Total	4		
dimension5	1.00	3	2.67	8.00
	2.00	1	2.00	2.00
	Total	4		

Test Statistics(c)

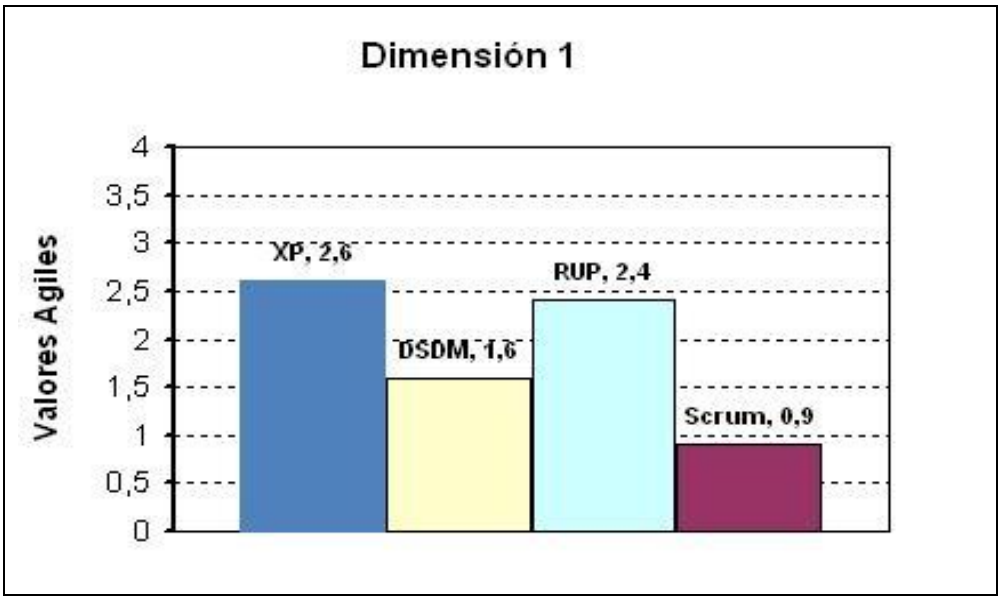
			dimension1	dimension2	dimension3	dimension4	dimension5
Mann-Whitney U			1.000	.000	.000	.000	1.000
Wilcoxon W			7.000	1.000	1.000	6.000	2.000
Z			-.447	-1.342	-1.732	-1.414	-.447
Asymp. Sig. (2-tailed)			.655	.180	.083	.157	.655
Exact Sig. [2*(1-tailed Sig.)]			1.000(a)	.500(a)	.500(a)	.500(a)	1.000(a)
Monte Carlo Sig. (2-tailed)	Sig.		1.000(b)	.501(b)	.252(b)	.252(b)	1.000(b)
	99% Confidence Interval	Lower Bound	1.000	.489	.240	.241	1.000
		Upper Bound	1.000	.514	.263	.263	1.000
Monte Carlo Sig. (1-tailed)	Sig.		.504(b)	.247(b)	.252(b)	.252(b)	.496(b)
	99% Confidence Interval	Lower Bound	.491	.236	.240	.241	.483
		Upper Bound	.517	.258	.263	.263	.509

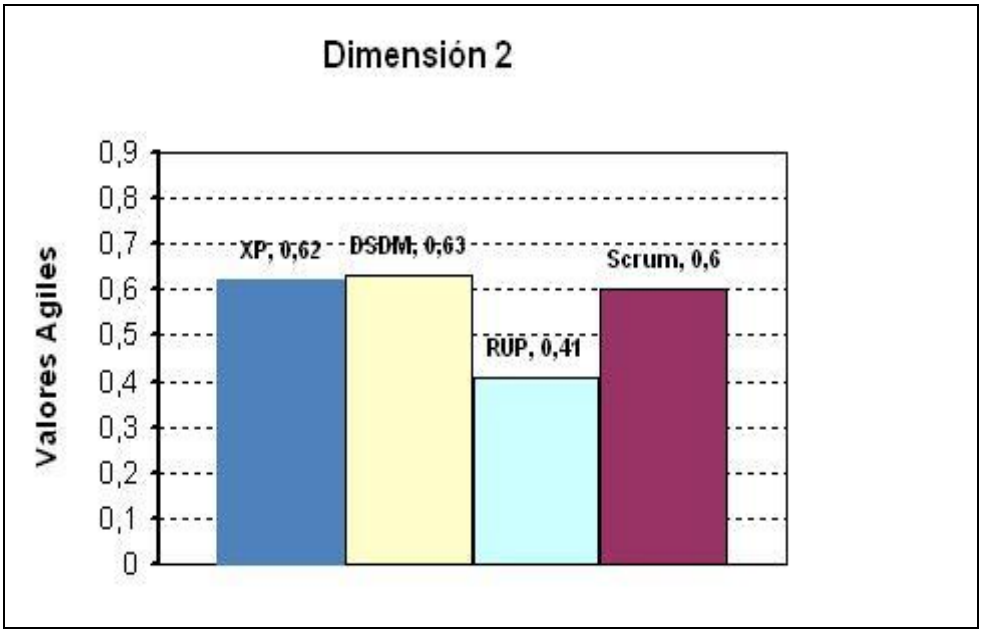
a Not corrected for ties.

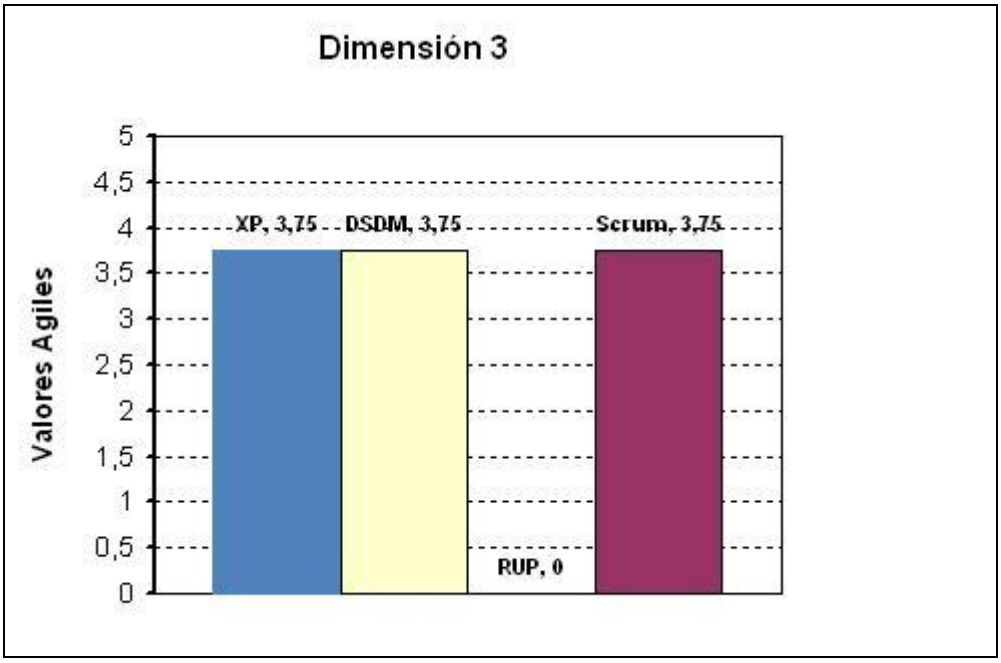
b Based on 10000 sampled tables with starting seed 334431365.

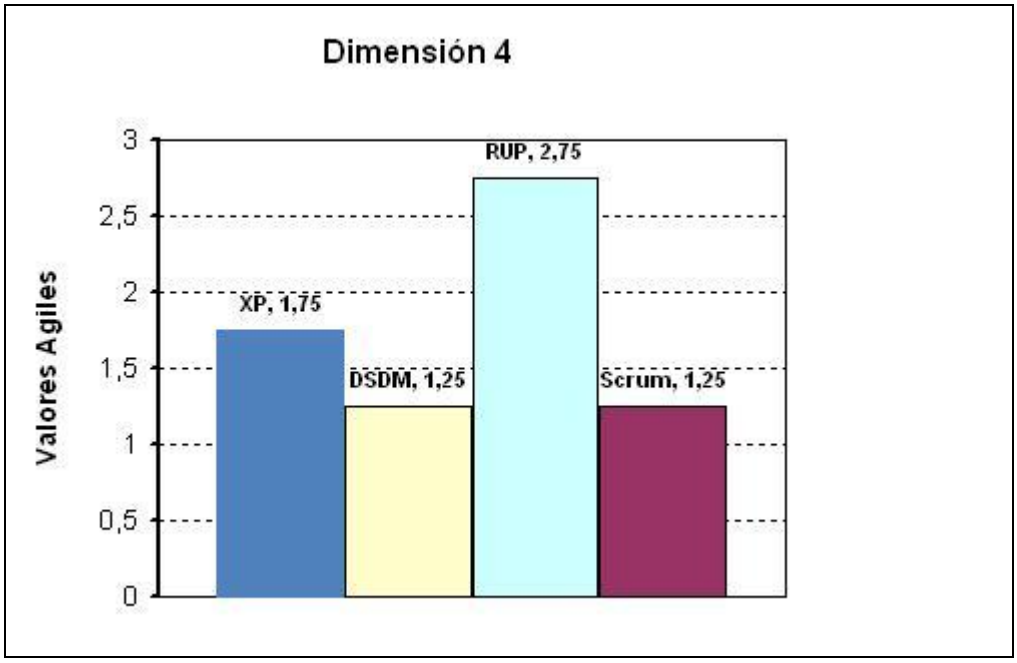
c Grouping Variable: Tipo

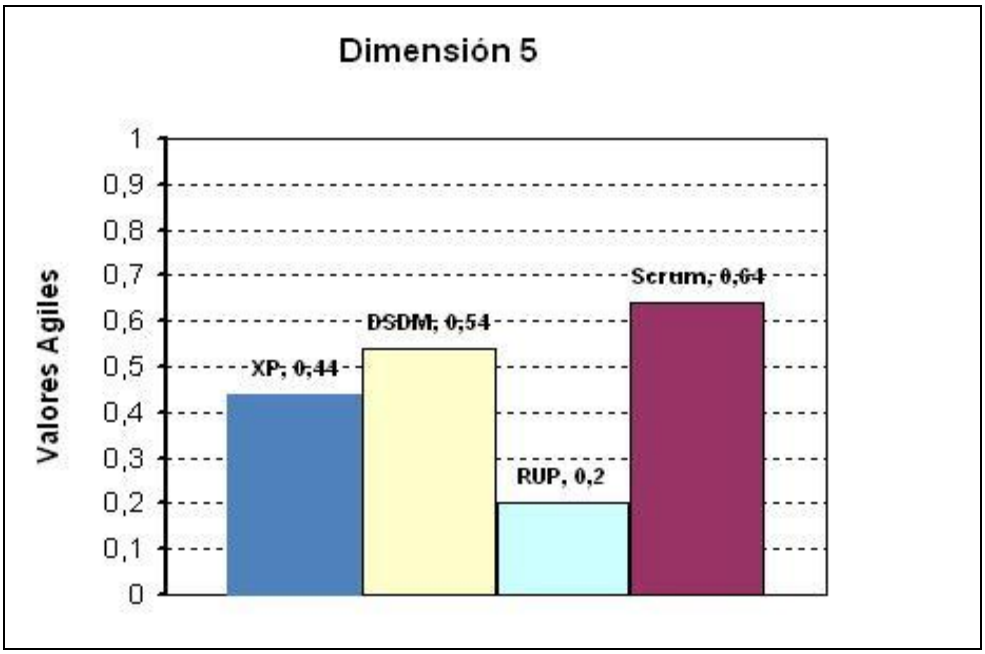
Anexo 6 (Valor obtenido en cada dimensión)






Dimensión 3
Valores Agiles
5
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
XP, 3,75
DSDM, 3,75
RUP, 0
Scrum, 3,75


Dimensión 4
Valores Agiles
3
2,5
2
1,5
1
0,5
0
XP, 1,75
DSDM, 1,25
RUP, 2,75
Scrum, 1,25


Dimensión 5
Valores Agiles
1
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
XP; 0;44
DSDM; 0;54
RUP, 0,2
Scrum; 0;64

GLOSARIO DE TERMINOS

Ingeniería de Software: Es una tecnología multicapa en la que, se pueden identificar: los métodos (indican cómo construir técnicamente el software), el proceso (es el fundamento de La Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos)

Metodologías: Se refiere a los métodos de investigación en una ciencia. Se entiende como la parte del proceso de investigación que permite sistematizar los métodos y las técnicas necesarios para llevarla a cabo. Define Quién debe hacer, Qué, Cuándo y Cómo debe hacerlo.

Metodologías de Desarrollo: Se define como un conjunto de filosofías, etapas, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas de información.

Metodología Ágil: Constituyen un nuevo enfoque en el desarrollo de software, mejor aceptado por los desarrolladores de proyectos que las metodologías convencionales debido a la simplicidad de sus reglas y prácticas, su orientación a equipos de desarrollo de pequeño tamaño, su flexibilidad ante los cambios y su ideología de colaboración .

Agilidad: Ligero, pronto, expedito, se dice de la persona que mueve o utiliza sus miembros con soltura. También se pude describir a la agilidad como una combinación de flexibilidad, velocidad y elasticidad. En el entorno de la investigación es la habilidad de responder de forma versátil al cambio para maximizar los beneficios.

Ciclo de Vida: Es un proceso por el cual los analistas de sistemas, los ingenieros de software, los programadores y los usuarios finales elaboran sistemas de información y aplicaciones informáticas.

Gurús: Une la experiencia de una veintena de profesionales de la comunicación, tan conocedores como apasionados del mundo de las nuevas tecnologías de la información e Internet.

Iteraciones: En el contexto de un proyecto de software se refieren a la técnica de desarrollar y entregar componentes incrementales de funcionalidades del negocio. Una iteración resulta en uno o más paquetes atómicos y completos del trabajo del proyecto que pueda realizar alguna función tangible del negocio. Múltiples iteraciones contribuyen a crear un producto completamente entregable.

Stakeholders: Partes involucradas en el proceso de desarrollo de un software que esperan recibir un beneficio del mismo. (Usuarios, Clientes, Managers).

Release: Lanzamiento de la versión definitiva de un producto final a menos que aparezcan errores que lo impidan.

Refactoring. Técnica que mantiene intacto el funcionamiento del software mejorando la estructura interna del mismo.

Etapas Básicas: Entiéndanse por estas aquellas fases esenciales del ciclo de vida de un software como es el caso del análisis, diseño, implementación y pruebas.

Modelos Esenciales: Se refiere a los artefactos esenciales que se deben obtener en cada etapa básica del proceso de software.