Cuatro enfoques metodológicos para el desarrollo de Software RUP – MSF – XP - SCRUM

Oiver Andrés Pérez A.

Recibido el 1 de abril de 20111. Aprobado el 10 de junio de 2011

Resumen

El presente artículo aborda el proceso de desarrollo de software desde cuatro enfoques metodológicos: RUP, MSF, XP, SCRUM, en los cuales se exponen las características, estructura, proceso, principios, ciclo de vida, artefactos y roles entre otras características propias de cada metodología. La investigación se realizó a partir de la revisión de literatura en la cual se analizaron y seleccionaron cuatro enfoques metodológicos de desarrollo de software de gran uso, reconocimiento y vigencia en la industria del desarrollo de software. En este sentido, el propósito del artículo es aportar una guía práctica que permita conocer y aplicar las diferentes estrategias metodológicas proporcionadas por estos cuatro enfoques metodológicos para el desarrollo de software.

Palabras clave

Ingeniería de Software, Metodologías de desarrollo, Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), Extreme Programming (XP), SCRUM.

Abstract

This paper takes into account the process of development of software from four different methodological focus, in which are considered the characteristics, structure, process, principles, life cycle, artifacts and roles among other characteristics proper to each methodology. This research has been made revising some literature from which four methodological focus of development of software have been analyzed and selected since they are very useful, recognized and prevailing to the development of the software industry. So that, the purpose of this paper is just to provide a practical guide that helps to know and to apply the different methodological strategies provided by these four methodological focus for the development of software.

Keywords

Software engineering, Software Development Process, Rational Unified Process (RUP), Microsoft Solutions Framework (MSF), Extreme programming (XP), SCRUM.

I. Introducción

En los años 80 se propuso que la mejor forma de desarrollar un sistema software era por medio de una planificación rígida y meticulosa del proyecto, soportada por herramientas CASE (Ingeniería de Software Asistida por Computador) y algunos procesos de desarrollo rigurosos y altamente controlados, que eran sinónimo de garantía y calidad en el software. Estas metodologías tenían una carga de trabajo pesada en planificación, diseño y documentación, absorbiendo gran parte del tiempo destinado al desarrollo del sistema.

Al implementar estas metodologías en proyectos pequeños o medianos con mayores exigencias en los tiempos de respuesta, se obtuvo como resultado la ineficacia en los procesos, debido a que se pasaba más tiempo pensando el sistema y al momento de liberar el producto se hacía casi imposible realizar cambios en las especificaciones, puesto que se debía empezar desde cero con el análisis y la documentación, haciendo del desarrollo de software un proceso improductivo e ineficiente. Aun así, estas metodologías se siguen implementando en determinados proyectos que no requieren de resultados rápidos pero sí de procesos críticos.

En los años 90 se comenzaron a proponer métodos ágiles para el desarrollo de software, que permitieran a los desarrolladores concentrarse en el software y no totalmente en el diseño y documentación del mismo. Éstas metodologías tienen un enfoque iterativo para la especificación, el desarrollo y la entrega del producto, teniendo como principio que los requerimientos podían cambiar permanentemente y durante el proceso de desarrollo, entregando sistemas funcionales más rápidamente con la posibilidad de agregar nuevos cambios en las especificaciones.

En la actualidad, el proceso de desarrollo de software ha sido abordado desde diferentes metodologías, las cuales tienen diferentes enfoques para la captura de requerimientos y el proceso de desarrollo del sistema software, algunas de ellas se basan en analizar y documentar rigurosamente las especificaciones del sistema, para luego realizar un desarrollo y posteriormente efectuar las pruebas. Otros métodos proponen centrarse en la organización del equipo de trabajo, incluir al cliente activamente y en arrojar resultados satisfactorios más rápidamente. Sea cual fuere la metodología es conveniente saber que éstas se eligen e implementan de acuerdo a la naturaleza del proyecto, llegando incluso a combinarse entre sí para lograr mejores resultados.

El objetivo de éste trabajo es analizar las cuatro metodologías propuestas, caracterizarlas y proporcionar un guía que permita entender de una manera general el enfoque de cada una, así como sus ventajas y desventajas, permitiendo a los estudiantes y desarrolladores tener un panorama global sobre las tendencias actuales en procesos de desarrollo de software.

El contenido del artículo está organizado de la siguiente manera: En la sección II se presenta la metodología Rational Unified Process (RUP) de IBM. En la sección III se presenta la metodología Microsoft Solutions Framework (MSF) de Microsoft. En la sección IV se presenta la Extreme Programming (XP) basada en los métodos ágiles y propuesta por Kent Beck. En la sección V se presenta la disciplina SCRUM basada en la rapidez y flexibilidad de métodos de desarrollo avanzados probados en la industria de productos comerciales y finalmente en la sección VI se presentan las conclusiones y una matriz comparativa de las cuatro metodologías.

II. RATIONAL UNIFIED PROCESS (RUP)

RUP es una metodología que tiene como objetivo ordenar y estructurar el desarrollo de software, en la cual se tienen un conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema Software (Amo, Martínez y Segovia, 2005). Inicialmente fue llamada UP (Unified Process) y luego cambió su nombre a RUP por el respaldo de Rational Software de IBM. Ésta metodología fue lanzada en 1998 teniendo como sus creadores a Ivar Jacobson, Grady Booch y James Rumbaugh. El RUP nació del UML (Unified Modeling Language) y del UP (Sommerville, 2005).

Características del RUP

El RUP es un proceso basado en los modelos en Cascada y por Componentes, el cual presenta las siguientes características: Es dirigido por los casos de uso, es centrado en la arquitectura, iterativo e incremental (Booch, Rumbaugh y Jacobson, 2000), lo cual es fundamental para el proceso de desarrollo de software. A continuación se explican las tres características de RUP:

a) **Casos de Uso**: Describe un servicio que el usuario requiere del sistema, incluye la secuencia completa de interacciones entre el usuario y el sistema.

b) **Centrado en la arquitectura:** Comprende las diferentes vistas del sistema en desarrollo, que corresponden a los modelos del sistema: Modelos de casos de uso, de análisis, de diseño, de despliegue e implementación. La arquitectura del software es importante para comprender el sistema como un todo y a la vez en sus distintas partes (Abrahamsson, Salo, Ronkainen y Warsta, 2002), sirve para organizar el desarrollo, fomentar la reutilización de componentes y hacer evolucionar el sistema, es decir, agregarle más funcionalidad (Pressman y Murrieta, 2006)

En la figura 1 se aprecia la forma en que los modelos de la arquitectura se completan en cada ciclo, ejemplo: se ve en la "línea base de la arquitectura" que la barra que denota el modelo de despliegue está clara e incompleta, evidenciándose una im-

plementación parcial del sistema, lo cual mostraría solo algunas funciones y propiedades del software en construcción. A esta parcialidad en la implementación se le conoce como arquitectura ejecutable. En la misma gráfica que se encuentra arriba se ve la misma barra pero un poco más oscura, lo cual muestra que el modelo se ha estado mejorando progresivamente, mostrando que durante la construcción los diferentes modelos se van desarrollando hasta completarse.

De igual forma se aprecia la misma barra en la "línea base al final de la construcción" en la cual se ve la barra del modelo de despliegue completa y con un color más oscuro, esto obedece a los refinamientos sucesivos que hace la metodología RUP a la arquitectura ejecutable, proporcionando de esta manera un prototipo evolutivo y funcional. De la misma manera la arquitectura como tal no cambia drásticamente pues gran parte de la arquitectura se definió durante la fase de elaboración, pero puede agregar modelos así como lo muestra la gráfica con la adición del modelo de pruebas a la misma arquitectura:

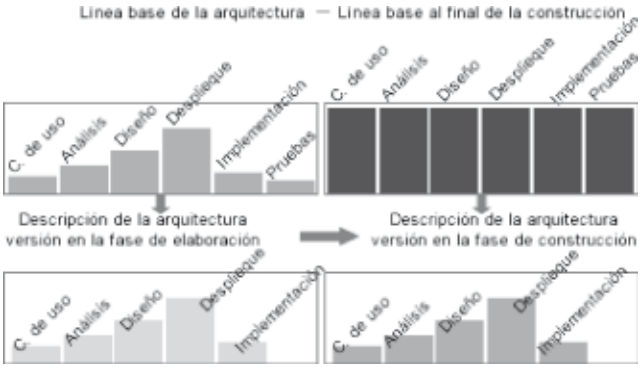

Figura 1. *Arquitectura RUP. Fuente: Adaptado de RUP (Booch, Rumbaugh y Jacobson, 2000)*

c) Iterativo e Incremental: Significa que la aplicación se divide en pequeños proyectos, los cuales incorporan una parte de las especificaciones, y el desarrollo de la misma es una iteración que va incrementando la funcionalidad del sistema de manera progresiva (Silva, Barrera, Arroyave y Pineda, 2007)

Tal como lo muestra la figura 2, una iteración está compuesta por los requisitos, análisis, diseño, implementación y pruebas; pero dicha iteración sólo entrega una parte pequeña pero funcional del sistema, de tal forma que los requisitos y demás modelos no se desarrollan en una sola iteración sino progresivamente, ello con la finalidad de poder garantizar entregas funcionales e iterativas y de tal forma ir completando el sistema software paso a paso. Cabe aclarar que una iteración también incluye otros artefactos que no están explícitamente en la gráfica, tales como la planificación y el análisis de la iteración, entre otras actividades específicas concebidas dentro de esa iteración.

Figura 2. *Las Iteraciones en RUP. Fuente: Adaptado de RUP (Booch, Rumbaugh y Jacobson, 2000)*

Estructura del RUP

El proceso del RUP se ejecuta en tres perspectivas: La perspectiva dinámica, la cual contiene las fases del modelo sobre el tiempo; la estática que muestra las actividades del proceso y la práctica, que muestra las buenas prácticas durante el proceso del RUP (IBM, s. f.)

La figura 3 muestra la estructura de RUP y la forma en que se relacionan sus tres perspectivas. En ésta se aprecia la forma en que las disciplinas se aplican a cada una de las fases hasta lograr su completitud, y a su vez, cómo cada fase se completa de forma iterativa para así avanzar a la fase siguiente. De igual forma se aprecia que la perspectiva de buenas prácticas está en un eje "z" que es transversal a las perspectivas dinámica "x" y estática "y", funcionando de manera permanente en el proceso de desarrollo de software.

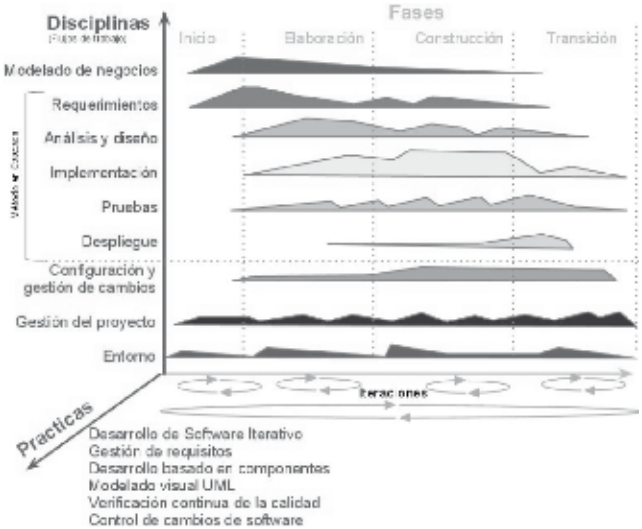

Figura 3. *Estructura de RUP. Fuente: Adaptado de RUP (IBM, s. f.)*

Para aclarar esta relación, a continuación se presenta una descripción de las tres perspectivas:

a) La perspectiva dinámica se compone por las fases de Inicio, Elaboración, Construcción y Transición,

cada fase se subdivide en iteraciones (Rational Software Corporation, 1998) y comprenden los siguientes objetivos:

- Fase de inicio: Su objetivo es la comunicación con el cliente y las actividades de planeación. Se establece el caso del negocio para el sistema, así como la identificación de todas las entidades externas que interactúan con el sistema y sus respectivas iteraciones.

- Fase de elaboración: Tiene como fin desarrollar un entendimiento del dominio del problema, crear un marco de trabajo arquitectónico para el sistema, desarrollar el plan del proyecto e identificar los riesgos claves. Al finalizar esta fase se debe tener el modelo de requerimientos del sistema (UML), una arquitectura y un plan de desarrollo.

- Fase de construcción: Su objetivo es el diseño del sistema, la programación, las pruebas y la integración de todas las partes del sistema software. Al final de esta fase se debe tener un software operativo con su respectiva documentación.

- Fase de transición: En esta fase el sistema software se entrega a los usuarios finales para sus respectivas pruebas en un entorno real. Al terminar esta fase se debe tener un software documentado y funcionando correctamente.

b) La perspectiva estática define dentro del proceso de desarrollo de software el quién hace qué, cómo y cuándo (Booch, Jacobsony Rumbaugh, 2006). El "quién" corresponden a los roles, el "qué" y el "cómo" corresponde a las actividades y artefactos, y el "cuándo" corresponde al flujo de trabajo. Para ello es necesario tener claro los siguientes elementos:

- Roles: Definen el comportamiento y las responsabilidades de cada individuo o de un grupo. Una persona puede desempeñar varios roles y un rol puede ser desempeñado por varias personas. Los roles definidos en RUP son: Analistas, desarrolladores, gestores, apoyo, especialista en pruebas y cualquier otro rol del cual se tuviera necesidad.

- Actividades: Es una unidad de trabajo que una persona que desempeña un rol puede realizar. Las actividades tienen objetivos concretos tales como: Planear una iteración, revisar el diseño, ejecutar pruebas de rendimiento, entre otras.

- Artefactos: También denominado producto, es un modelo de información que es producido o modificado durante el proceso de desarrollo de software. Los artefactos son los resultados tangibles del proyecto, las cosas que se van creando y usando hasta tener el producto software terminado. Algunos artefactos pueden ser: un modelo de casos de uso, el documento de la arquitectura, etc.

- Flujo de trabajo: Es la relación entre los roles y los artefactos o productos que producen resultados observable en el desarrollo del sistema software. Estos se dividen en flujos de trabajo de proceso y flujos de trabajo de soporte, los primeros reflejan actividades propias del modelo en cascada y contiene el modelado de negocios, requerimientos, análisis y diseño, implementación, pruebas y despliegue; y los segundos contienen la configuración y gestión de cambios, la gestión del proyecto y el entorno.

c) La perspectiva práctica describe seis buenas prácticas en Ingeniería de Software que son recomendables en el desarrollo de sistemas software, las cuales son: Desarrollo iterativo, gestión de requisitos, desarrollo basado en componentes, modelado visual UML, verificación continua de la calidad y control de cambios de software (Leterlier, s. f.) Estas prácticas se ejecutan durante todo el proyecto y de manera transversal a las perspectivas dinámica y estática.

El ciclo de vida del RUP

La metodología RUP se repite a lo largo de una serie de ciclos (figura 4) que constituyen la vida de un sistema desde su nacimiento hasta su muerte. Cada ciclo concluye con una versión del producto para los clientes (Traa, 2006),

En la figura 4 se puede apreciar que el ciclo de vida de RUP está comprendido por varios ciclos. Las versiones y ciclos le añaden funcionalidad al sistema hasta el punto donde ya termine su ciclo de vida con la muerte o cumplimiento total del objetivo para el cual fue diseñado el software. En la figura 5 se explican los elementos que conforman el proceso interno de un ciclo. Los cuales comprenden las fases y sus respectivas iteraciones, a su vez cada ciclo concluye con una versión del sistema software.

Figura 4: *Ciclo de vida de Rup Fuente: Adaptado de RUP (Booch, Rumbaugh y Jacobson, 2000)*

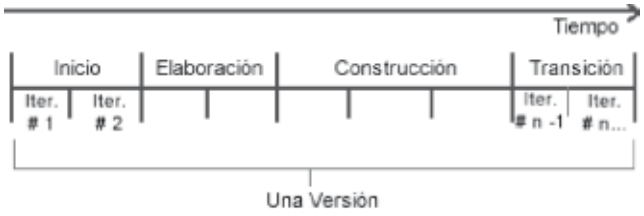

Figura 5: *Un ciclo Rup Fuente: Adaptado de RUP (Booch, Rumbaugh y Jacobson, 2000)*

En síntesis, la metodología RUP es un proceso de desarrollo de software que trabaja de la mano con el UML y es una de las metodologías estándar más usadas para el análisis, desarrollo y documentación de sistemas orientados a objetos, además de su gran respaldo por parte de IBM.

Por sus características se implementa con mayor frecuencia en proyecto de gran complejidad y magnitud, que dispongan de un equipo de trabajo con experiencia en desarrollo de proyectos, así como con un alto conocimiento en la metodología, sin dejar de lado su aplicabilidad en proyectos de corto tiempo y poca complejidad, pues la metodología tiene la capacidad de poder adaptarse a los diferentes tipos de proyecto software.

III. MICROSOFT SOLUTIONS FRAMEWORK (MSF)

MSF es una guía de desarrollo de software flexible que permite aplicar de manera individual e independiente cada unos de sus componentes, es escalable pues está diseñada para poder expandirse según la magnitud del proyecto. La metodología MSF está basada en un conjunto de principios, modelos, disciplinas, conceptos, directrices y practicas aprobadas por Microsoft, que asegura resultados con menor riesgo y de mayor calidad, centrándose en el proceso y las personas (Gattaca S. A.)

Microsoft Solutions Framework se introdujo por primera vez en 1994 como un conjunto de las mejores prácticas en los desarrollo de Software de Microsoft y Microsoft Consulting Service. Esta metodología ha estado evolucionando y mejorando con la experiencia de grupos de trabajo reales los cuales contribuyeron a perfeccionar este Framework (Wilmot *et al*., 2004). De igual manera, MSF también retoma algunas de las características propias de metodologías tradicionales.

Características de MSF

Éste Framework está basado en los modelos espiral y cascada, lo cual indica que toma elementos de los métodos tradicionales que aún son referentes importantes para procesos de software. Es adaptable, flexible y escalable, e independiente de tecnologías, lo cual significa que no se cierra a un sólo modelo de programación sino más bien queda abierto según la naturaleza del proyecto. Usa como referente el DSL (Domain-Specific Language) para realizar el modelado, así como RUP se apoya en UML para hacer el modelado (Microsoft).

Componentes

MSF es un FrameWork que contiene tres componentes: Los principios fundamentales, los modelos y las disciplinas. Éstos componentes pueden ser utilizados individuamente o adoptados como un todo integrado según la naturaleza del proyecto (Microsoft, 2003), a continuación se explican los tres componentes:

Principios fundamentales

Los principios de MSF son 8 valores y normas que son comunes en todo el FrameWork, los cuales contribuyen a mejorar el trabajo en equipo y a centrarse en mantener el objetivo del proyecto siempre en marcha (Abrahamsson, Salo, Ronkainen y Warsta, 2002), estos principios son:

· **Fomentar la comunicación abierta:** El mantener un buen flujo de información entre los integrantes del equipo, pueden reducir las posibilidades de malentendidos y contribuir a la solución colectiva de problemas.

· **Trabajar hacia una visión compartida:** El equipo de trabajo debe comprender y trabajar hacia las metas y objetivos del proyecto.

· **Empoderar a los miembros del equipo:** Los miembros del equipo SCRUM deben tener funciones claras, de tal manera que puedan desenvolverse como un grupo creativo, eficaz y capaz de cumplir sus propios objetivos, al igual que resolver sus dificultades.

· **Establecer la rendición de cuentas claras y la responsabilidad compartida:** Cada equipo tiene funciones claras y son responsables de una parte de la solución, así como del éxito global del proyecto, trabajando desde la individualidad en procura de objetivos colectivos.

· **Centrarse en ofrecer valor empresarial:** El equipo de trabajo debe procurar ofrecer soluciones reales que satisfagan necesidades y beneficien al comprador del producto, pues entregar un producto bien elaborado y que cumpla los requerimientos para el cual fue diseñado es el objetivo a alcanzar.

· **Mantenerse ágil, en espera de un cambio:** El cambio constante debe ser aceptado, es decir, el quipo

debe estar preparado para afrontar estas situaciones y ofrecer soluciones eficaces. MSF acepta la combinación de caos y orden lo cual deja entrever las altas posibilidades de que los proyectos software se salgan de control.

· **Invertir en la calidad:** El equipo debe mantener la mentalidad centrada en la calidad, se debe invertir en las personas, en los procesos y las herramientas, pues ésta inversión puede retribuir en resultados de mayor calidad.

· **Aprender de todas las experiencias:** Tener una mentalidad abierta para aprender del éxito o fracaso de proyectos propios y ajenos.

Modelos

Los modelos describen esquemas a seguir para la organización de los equipos y los procesos del proyecto (Microsoft, 2003), lo cual especifica un modelo para el equipo de trabajo y uno para los procesos:

a) Equipo de trabajo:

Este modelo se encarga de organizar las personas para que realicen el trabajo y se asegura que todas las metas del proyecto se cumplan. Define los principios, los roles y las actividades involucrando al equipo en todas las decisiones fundamentales que rodean el proyecto.

· Principios: Todos los miembros del equipo son compañeros por lo tanto no hay jefes, se deben tener las funciones y las responsabilidades claras. Procurar evitar los defectos en los procesos relacionados con el equipo y tener una mentalidad de calidad orientada a la satisfacción del cliente, así como tener voluntad y capacidad de aprender.

· Roles y actividades: Un rol define comportamientos y actividades de cada individuo o grupo, cada persona puede tomar más de un rol y múltiples personas pueden tomar un rol, en este modelo se describen seis roles con sus respectivas actividades, las cuales se relacionan en la figura 6.

Figura 6: *Modelo de equipo de trabajo. Fuente: Adaptado de Microsoft, 2003.*

b) Proceso:

Éste modelo se encarga de organizar los procesos necesarios para lograr llevar a término una solución. Esto se logra dividiendo las tareas del proyecto en cinco fases, las cuales proporcionan herramientas para mejorar el control sobre el proyecto, minimizar el riesgo y aumentar la calidad del producto. Al igual que el proceso RUP, MSF también tiene sus prácticas inherentes al desarrollo de software, tales como la especificación, el desarrollo, la validación y la evolución del software.

De lo anterior se desprenden unos principios relacionados con el proceso MSF, los cuales son: el manejo de versiones, la gestión del riesgo, dividir el proyecto en partes y realizar construcciones diarias.

Por otra parte, la metodología MSF establece 5 fases con sus respectivas tareas, las cuales se aprecian en la figura 7. En las características de MSF se mencionó que la metodología se podía aplicar de manera flexible, es decir, que los componentes no eran dependientes unas de otras. Es importante aclarar que ese grado de flexibilidad no aplica para éste modelo de proceso (Figura 7).

Figura 7: *Modelo de proceso. Fuente: Adaptado de Microsoft, 2003.*

La figura 7 muestra las fases y sus respectivos objetivos a cumplir. Los cuales se explican a continuación:

· **Fase visión:** Se debe tener el objetivo y limitaciones del proyecto, el análisis de los problemas de negocios, el ámbito de la aplicación, la evaluación del riesgo y panificación del producto.

· **Fase planificación:** Se debe tener la ingeniería de requerimientos, planificación y gestión de riesgos.

· **Fase desarrollo:** En esta fase se codifica y se realizan las respectivas pruebas, también se identifican y mitigan los riesgos existentes.

· **Fase estabilización:** Se realizan pruebas beta, se crea un plan de gestión de incidencias, se revisa la documentación final de la arquitectura y se elabora un plan de despliegue.

· **Fase implantación:** Se libera la solución software, se crea un registro de mejoras y sugerencias, se revisan las guías y manuales de usuario y se entrega el proyecto final.

Gestión de requerimientos

En todo proceso de desarrollo de software es fundamental el proceso de gestión de requerimientos. MSF incluye dentro del modelo de proceso un documento de visión, y documentos de especificación funcional. De igual forma se incluye también un acuerdo inicial sobre la funcionalidad con el cliente, se toman los resultados para verificar que se cumplen los requisitos y se hace seguimiento a cada elemento al final de la ejecución.

El ciclo de vida de MSF

El proceso del MSF se puede llevar a cabo de forma iterativa, tal como se aprecia en la figura 8, de tal forma que al liberar una solución, se puede iniciar nuevamente la metodología para darle más funcionalidad al producto (Llorens, 2005).

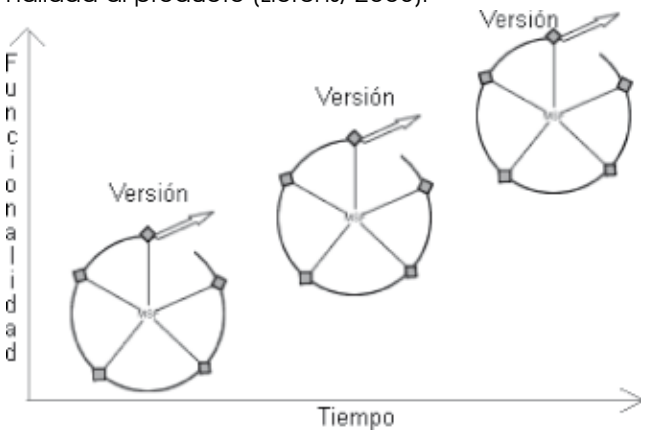

Figura 8: *Ciclo de vida MSF. Fuente: Adaptado de Microsoft, 2003.*

De forma similar al ciclo propuesto en RUP, la metodología MSF que se muestra en la figura 7, propone para su ciclo de vida, un desarrollo por fases, las cuales culminan con una versión del producto, estas versiones se pueden seguir trabajando de manera iterativa hasta conseguir el resultado esperado, y es en ese momento donde terminaría el ciclo de vida.

Disciplinas

MSF presenta un conjunto de métodos para la gestión del proyecto, la gestión del riesgo y la gestión de preparación para el cambio (Del Maschi *et al.*, 2008). Dentro de las cuales comprende:

· **Gestión de proyecto:** Tiene como objetivo permitir mayor escalabilidad en proyectos pequeños, grandes y complejos, basado en la planificación sobre las entregas cortas, la incorporación de nuevas características sucesivamente, e identificar cambios ajustando el cronograma.

· **Gestión del riesgo:** En la figura 9 se pueden observarlos pasos de la disciplina la cual se encarga de ayuda al equipo a tomar las decisiones correctas y controlar las emergencias que puedan presentarse, por medio de un entorno estructurado para la toma de decisiones y acciones, valorando los riesgos que puedan provocar.

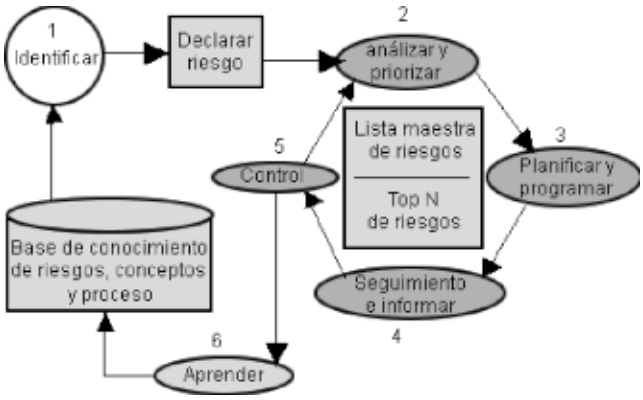

Figura 9. *Gestión del riesgo. Fuente: Adaptado de Microsoft, 2003.*

En sintonía con el resto de la metodología, en la figura 9 se evidencia la importancia de la comunicación con el equipo. El declarar el riesgo a tiempo permite analizar, buscar soluciones y aprender de los riesgos ya superados.

· **Gestión de cambios:** Esta disciplina tiene como objetivo lograr que el equipo sea proactivo en lugar de reactivo (Microsoft, 2003), teniendo en cuenta que los cambios deben considerase riesgos y por lo tanto se deben registrar y hacer evidentes. En la figura 10 se plantea las acciones y flujos a seguir para lograr anteceder a los riesgos.

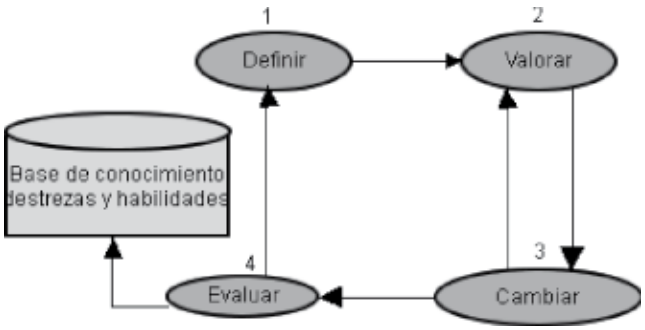

Figura 10: *Gestión de cambios. Fuente: Adaptado de Microsoft, 2003.*

En conclusión, la metodología propuesta por MSF tiene como propósito lograr entregas con un margen amplio de éxito, basados en la calidad del producto software, teniendo presente las necesidades del cliente y el principio de flexibilidad, así como el cumplimiento con los compromisos adquiridos, la gestión de los costos y la minimización de los riesgos inherentes en todo proyecto de desarrollo de software.

IV. PROGRAMACIÓN EXTREMA

La programación extrema o Extreme Programming, es una disciplina de desarrollo de software basada en los métodos ágiles, que evidencia principios tales como el desarrollo incremental, la participación activa del cliente, el interés en las personas y no en los procesos como elemento principal, y aceptar el cambio y la simplicidad (Beck *et al*., 2001). El trabajo fundamental se publicó por Kent Beck en 1999, y tomó el nombre de Programación Extrema por las prácticas reconocidas en el desarrollo de software y por la participación del cliente en niveles extremos (Wells, 2009). Éste método, al igual que RUP y MSF, también tiene principios los cuales son buenas prácticas a tener presente en el desarrollo del software.

Los principios XP comprenden diez buenas prácticas que involucran al equipo de trabajo, los procesos y el cliente; los cuales son:

· **Planificación incremental:** Se toman los requerimientos en Historias de Usuario, las cuales son negociadas progresivamente con el cliente.

· **Entregas pequeñas:** Se desarrolla primero la más mínima parte útil que le proporcione funcionalidad al sistema, y poco a poco se efectúan incrementos que añaden funcionalidad a la primera entrega, cada ciclo termina con una entrega del sistema, en la figura 11, se muestra como es el proceso de entrega en XP (Programación Extrema, s. d.)

Figura 11: *El ciclo de entrega de XP. Fuente: Adaptado de XP.*

Al igual que en el ciclo de vida de RUP y MSF, en XP el ciclo de vida termina cuando ya no hay más ciclos de entrega y el sistema ha cumplido con el objetivo para el cual fue diseñado, de no ser así, se deberá continuar con el ciclo especificado en la figura 11 hasta que la funcionalidad del sistema sea la deseada.

· **Diseño sencillo:** Solo se efectúa el diseño necesario para cumplir con los requerimientos actuales, es decir, no se abordan requerimientos futuros.

· **Desarrollo previamente aprobado:** Una de las características relevantes y propias de XP es que primero se escriben las pruebas y luego se da la codificación, esto con la finalidad de asegurar la satisfacción del requerimiento.

· **Limpieza del código o refactorización:** Consiste en simplificar y optimizar el programa sin perder funcionalidad, es decir, alterar su estructura interna sin afectar su comportamiento externo (Abrahamsson, Salo, Ronkainen y Warsta, 2002).

· **Programación en parejas:** Es otra de las características de ésta metodología, que propone que los desarrolladores trabajen en parejas en una terminal, verificando cada uno el trabajo del otro y ayudándose para buscar las mejores soluciones. Se entiende que de esta forma el trabajo será más eficiente y de mayor calidad.

· **Propiedad colectiva:** El conocimiento y la información deben ser de todos, por lo tanto no se desarrollan islas de conocimiento, todos los programadores poseen todo el código y cualquiera puede sugerir y realizar mejoras.

· **Integración continua:** Al terminar una tarea, ésta se integra al sistema entero y se realizan pruebas de unidad a todo el sistema, ésta práctica permite que la aplicación sea más funcional en cada iteración y garantiza su funcionamiento con los demás módulos del sistema.

· **Ritmo sostenible:** No es aceptable trabajar durante grandes cantidades de horas ya que se considera que puede reducir la calidad del código y la productividad del equipo a mediano plazo, se sugieren 40 horas semanales.

· **Cliente presente:** Se debe tener un representante (Cliente o usuario final) tiempo completo, ya que en XP éste hace parte del equipo de desarrollo y es responsable de formular los requerimientos para el desarrollo del sistema.

Valores en XP

En todo desarrollo de un proyecto de software, los cambios serán algo inevitables, los requerimientos

cambiarán, las reglas del negocio, el equipo de trabajo y la tecnología, entre otros elementos involucrados en el proyecto. Por esta razón XP propone valores, que permitirán afrontar y sortear de una manera más efectiva los cambios en el proyecto (Wellington, 2005) los cuales se enfocan al equipo de trabajo de la siguiente manera:

· **Comunicación:** Aunque hay circunstancias que conducen a la ruptura de la comunicación, se debe procurar por comunicar cualquier cambio con el resto del equipo ya sean desarrolladores, cliente o jefe.

· **Sencillez:** Iniciar desde lo parte más simple que pueda darle funcionalidad al sistema, es decir abordar el problema con el mayor nivel de granularidad.

· **Retroalimentación:** La mejor manera de conocer el estado actual del sistema es haciéndole pruebas funcionales al software, esto proporcionará información real y confiable sobre el grado de fiabilidad del sistema.

· **Valentía:** El equipo de trabajo debe estar presto para asumir retos, ser valiente ante los problemas y afrontarlos, no tapar los errores, ya que tarde o temprano saldrán a flote y todo el sistema colapsará no se puede avanzar sobre los errores. Se recomienda tomar acciones correctivas a tiempo a fin de lograr el objetivo del proyecto.

Objetivos de XP

Aunque las metodologías RUP y MSF no lo muestran de una manera explícita, también para ellas la satisfacción del cliente y el trabajo en equipo es un objetivo. La metodología XP tiene dos objetivosprimordiales para el correcto desarrollo del proyecto (Jeffries, s. f.):

· **La satisfacción de cliente:** Entendida como dar al cliente lo que necesita y cuando lo necesita, respondiendo rápidamente a las necesidades de este. Uno de los factores importantes en todo proyecto de software es que el sistema software logre el objetivo para el cual fue diseñado y que el equipo de trabajo logre el objetivo para el cual fue contratado, de ahí que el incumplimiento de esto termine con un producto incompleto y un cliente insatisfecho.

· **Potenciar al máximo el trabajo en grupo:** Todos están involucrados y comprometidos con el desarrollo del software, tanto los jefes como los desarrolladores y los clientes, no hay agentes individuales o aislados al proyecto.

El proceso de XP

El proceso de XP al igual que RUP y MSF se presenta en fases, en XP se ejecuta en cuatro fases teniendo presente los principios y valores antes mencionados, los cuales son un eje fundamental para el correcto desarrollo de cada fase durante el ciclo. En la figura 12 se puede apreciar la relación de principios, valores y fases:

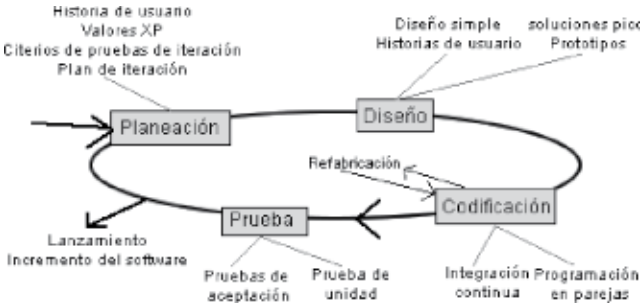

Figura 12: *El proceso X. Fuente: Adaptado de Pressman y Murrieta, 2006*

Al igual que en las metodologías antes mencionadas, en el proceso XP (figura 12), se observan una serie de fases que al ser concluidas dan origen a una versión del producto software, y cada versión es un ciclo, el cual hace parte del ciclo de vida del software. Al no tener más ciclos a ejecutar se entiende que los sistemas han cumplido con su objetivo, en caso contrario se deben seguir desarrollando ciclos para agregar la funcionalidad deseada. Cada fase del ciclo comprende lo siguiente:

· **Fase de planeación:** Ésta fase inicia con las historias de usuario que describen las características y funcionalidades del software. El cliente asigna un valor o prioridad a la historia, los desarrolladores evalúan cada historia y le asignan un costo el cual se mide en semanas de desarrollo.

· **Fase de diseño:** El proceso de diseño debe procurar diseños simples y sencillos para facilitar el desarrollo. Se recomienda elaborar un glosario de términos y la correcta especificación de métodos y clases para facilitar posteriores modificaciones, ampliaciones o reutilización del código. Anteriormente este proceso se apoyaba en el uso de tarjetas CRC (Colaborador-Responsabilidad-Clase) la cual identifica las clases orientadas a objetos que son relevantes para el incremento del software.

· **Fase de codificación:** En ésta fase los desarrolladores deben diseñar las pruebas de unidad que ejercitarán cada historia de usuario. Después de tener las pruebas, los desarrolladores trabajarán en parejas para concentrarse en lo que debe implementarse para pasar la prueba de unidad.

· **Fase de pruebas:** Las pruebas de unidad deben implementarse con un marco de trabajo que permita automatizarlas, con la finalidad de realizar pruebas de

integración y validación diarias, esto proporcionará al equipo un indicador del progreso y revelarán a tiempo si existe alguna falla en el sistema. Las pruebas (Sommerville, 2005) tienen las siguientes características:

- Desarrollo previamente aprobado: Significa que primero se escriben las pruebas y luego el código. Las pruebas deben simular el envío de la entrada ha probar y deben verificar que el resultado cumpla con las especificaciones de salida.

- Desarrollo de pruebas incremental: Los requerimientos del usuario se expresan como historias, el equipo de desarrollo evalúa cada historia y la divide en tareas. Cada una representa una característica distinta del sistema y se pueden diseñar pruebas de unidad para esa tarea.

- Participación del usuario en el desarrollo de las pruebas: El usuario ayuda a desarrollar las pruebas de aceptación, las cuales son pruebas que se implementan con los datos reales del cliente para verificar el cumplimiento real de sus necesidades.

- Uso de bancos de pruebas automatizados: Se debe usar un sistema que envíe a ejecución las pruebas automatizadas y de esta forma probar constantemente el sistema software.

Artefactos

En todo proceso de desarrollo de software se generan modelos de información. En XP se generan varios artefactos como las tarjetas de historias de usuario (Story Card), las tarjetas de tareas para la descarga de documentos, el código, las pruebas unitarias y de integración y las pruebas de aceptación. Los artefactos son importantes para conocer cuál fue el proceso de desarrollo del software y lograr entender cómo está construido el sistema, así como la ruta a seguir para agregar funcionalidad al sistema.

Roles

Los miembros de un equipo trabajan mejor cuando hay roles establecidos, cada rol tiene consigo responsabilidades que tienen como finalidad cumplir con los objetivos del proyecto. Algunos proyecto necesitan de múltiples roles como tésteres o probadores, ingenieros de calidad, analista de requerimientos, administrador del proyecto, administrador del producto, profesionales de marketing. El número de roles varía de acuerdo con el proyecto. A continuación se explican algunos de los más relevantes:

· **Programador:** Es el corazón de XP, el programador con base en su experiencia puede tomar decisiones que afecten el desarrollo del proyecto. Su tarea es lograr que la computadora comprenda y haga todo según los requerimientos del usuario, el programador debe conocer cómo hacer el programa y trabajar de la mano con el cliente.

· **Clientes:** El cliente dirige y conoce las metas a alcanzar en el proyecto. Debe conocer qué debe hacer el programa, para que de ésta forma guíe y trabaje de la mano con los programadores, por lo tanto debe aprender a escribir las historias de usuario. El cliente y los desarrolladores tienen gran responsabilidad en el proyecto.

· **Tester (probadores):** Su responsabilidad es correr las pruebas funcionales con regularidad y dar a conocer los resultados de esta, así como elaborar las pruebas junto con el cliente.

· **Tracker (responsable del seguimiento):** Debe conocer el alcance funcional del equipo, controla los tiempos de desarrollo, controlar los hitos y entregas, puede tomar decisiones estratégicas para el equipo y debe asegurar el alcance y despliegue de la aplicación.

En síntesis, actualmente XP es una de las metodologías de mayor aceptación en la industria del software, su enfoque basado en los métodos ágiles, su énfasis en la gestión del recurso humano el cuál es uno de los puntos más críticos en todo proyecto, y sus principios de previsibilidad y adaptabilidad hacen de esta metodología una buena opción a seguir.

V. SCRUM

SCRUM es un marco de trabajo basado en los métodos ágiles, que tiene como objetivo el control continuo sobre el estado actual del software, en el cual el cliente establece las prioridades y el equipo SCRUM se auto-organiza para determinar la mejor forma de entregar resultados (Abrahamsson, Salo, Ronkainen y Warsta, 2002).

SCRUM fue desarrollado en 1986 por Hirotaka Takeuchi e Ikujiro Nonaka quienes describieron una nueva aproximación metodológica que incrementa la rapidez y la flexibilidad en el desarrollo de nuevos productos comerciales. El enfoque de ésta metodología es como en el rugby, "donde el proceso es similar a un equipo entero que actúa como un sólo hombre para intentar llegar al otro lado del campo, pasando el balón de uno a otro". Ésta metodología se inició en el campo de las industrias automovilísticas y de tecnología, pero a principios de los años 1990 Ken Schwaber la llevó a la práctica en su compañía Advanced Development Methods (Mountain Goat Software, s.f.) al

igual que XP, en SCRUM se hace bastante énfasis en la gestión del recurso humano, esto se puede apreciar mejor en las características del método SCRUM, que se explican a continuación.

Características

SCRUM da prioridad a los individuos y las interacciones sobre los procesos y las tareas, lo cual significa que gran parte del éxito del proyecto radica en la forma cómo el equipo se organice para trabajar. Se debe tener una cohesión fuerte de equipo ya que el triunfo de un hito no es de un sólo miembro sino de todo el equipo SCRUM, todos se colaboran entre sí, y empujan a los integrantes que no están a la par con el equipo (Beck, K. *et al.*, 2001)

El enfoque SCRUM propone el software funcional sobre la excesiva documentación, a diferencia de RUP el cual es estricto en documentación. Se presenta al cliente las soluciones operables y no solo reportes de progresos, de ésta forma el cliente puede decidir avanzar o parar, en otros enfoques solo se ven resultados al final.

De igual forma, SCRUM promueve la colaboración con el cliente en lugar de rígida negociación de contratos. Por lo cual, es importante tener capacidad de respuesta para los cambios en lugar de seguir estrictamente una planificación, partiendo del principio que el proyecto software es cambiante. El propósito es que el cliente vaya observando los resultados, pueda decidir cambios en la marcha o incluso darle un giro completo al proyecto.

Valores

Al igual que en las tres metodologías abordadas anteriormente, SCRUM promueve valores (Sutherland y Schwaber, 2007) que ayudan a clarificar los procedimientos de la metodología y contribuye a garantizar el cumplimiento y la evolución de SCRUM; los cuales son:

· **Empoderamiento y compromiso de las personas:** Se procura delegar y atribuir responsabilidades con la finalidad que el equipo se pueda auto-organizar y tomar decisiones sobre el desarrollo del proyecto. Un miembro del equipo no puede tomar decisiones acertadas si no está involucrado en el proceso de desarrollo del software.

· **Foco en desarrollar lo comprometido:** Los miembros del equipo de trabajo deben centrarse en desarrollar lo pactado con el cliente y lo comprometido con el resto del equipo.

· **Transparencia y visibilidad del proyecto:** Se debe mantener informado al equipo, procurar evidenciar cualquier anomalía y proceder con transparencia, pues cualquier falla o error que no se socialice puede afectar el resto del proceso. También se recomienda hacer visible los avances durante el desarrollo del proyecto

· **Respeto entre las personas:** Los miembros del equipo, al igual que en un equipo deportivo deben confiar entre ellos y respetar sus conocimientos y capacidades, pues las cualidades de cada uno son las fortalezas de todo el equipo.

· **Coraje y responsabilidad:** Se debe tener responsabilidad y auto-disciplina (no disciplina impuesta), cada miembro del equipo debe estar presto a sortear dificultades y responder positivamente a los cambios que se puedan generar.

Roles

En todo proceso de desarrollo de software deben existir roles, los cuales definen comportamientos y actividades importantes para el proyecto. SCRUM divide su equipo de trabajo (Rising y Janoff, 2000) en cinco grupos de personas:

· **Propietario del producto:** Es la persona que determina las prioridades del proyecto, debe conocer muy bien y saber que se quiere del producto, para de esta forma guiar al equipo SCRUM hacia la consecución de los objetivos.

· **SCRUM Manager:** Es el encargado de gestionar y facilitar la ejecución del producto, debe asegurar el seguimiento de la metodología y el cumplimiento de las metas trazadas, así como de atender y solucionar los asuntos externos al proyecto.

· **Equipo SCRUM:** Es el corazón de la metodología pues ellos construyen el producto, está conformado por los desarrolladores.

· **Interesados:** También llamados StakeHolders son los que observan y asesoran el proceso, también pueden ser agentes externos interesados en financiar o promover el proyecto.

· **Usuarios:** Quizá uno de los menos tenidos en cuenta pero finalmente son ellos los que realizaran las pruebas lógicas de la aplicación y verificar si se cumplen sus expectativas. Los clientes pueden aportar ideas o necesidades no consideradas por el equipo SCRUM.

Artefactos

Al igual que en RUP, MSF y XP, los artefactos son los diferentes modelos de información (Mountain Goat Software, s.f.) generados durante el proceso de desa-

rrollo del software, SCRUM produce los siguientes tres artefactos:

· **Pila del producto:** Es el corazón de SCRUM, es la relación de requisitos del producto, en la cual no es necesario excesivo detalle pero sí deben estar priorizados. Ésta lista o pila del producto está en constante evolución y abierta a todos los roles, pero es el propietario del producto el responsable y quien decide sobre esta.

· **Pila del SPRINT:** Son los requisitos comprometidos por el equipo para el Sprint, se construyen con el nivel de detalle suficiente para lograr su ejecución por el equipo de trabajo.

· **Incremento:** Es una parte del producto desarrollado en un Sprint, y que es factible de ser usado, contiene las pruebas, una codificación limpia y documentada.

Reuniones

Es uno de los elementos fundamentales de la metodología SCRUM (Rising, y Janoff, 2000) y se realizan periódicamente. A diferencia de las metodologías expuestas anteriormente en este artículo, SCRUM define cómo deben ser las reuniones del equipo de trabajo y los resultados que ésta debe generar. A continuación se explican cada una de ellas:

· **Planificación del SPRINT:** Es una jornada de trabajo muy importante ya que su mala planificación puede arruinar todo el Sprint. En ésta reunión el propietario del producto explica las prioridades y dudas del equipo, estos estiman el esfuerzo de los requisitos prioritarios incluyendo una lista de miembros y nivel de dedicación, y a partir de ésta se elabora la pila de Sprint. El SCRUM Manager define en una frase el objetivo del Sprint.

· **Reunión diaria:** Comprende una reunión de mínimo 15 minutos y máximo 30 minutos de duración, en el mismo lugar de reunión y a la misma hora. La reunión está dirigida por el SCRUM Manager y sólo puede intervenir el Equipo SCRUM. Éste hace las siguientes preguntas a cada miembro del equipo:

¿Qué hiciste ayer?
¿Cuál es el trabajo para hoy?
¿Qué necesitas?

Una vez conocida la situación actual del equipo SCRUM se actualiza la pila del Sprint y el SCRUM Manager debe tomar decisiones de inmediato, también tiene la responsabilidad de señalar los obstáculos que deben ser resueltos externamente para no alargar más el tiempo de la reunión.

· **Revisión del SPRINT:** Es una reunión informativa, aproximadamente de 4 horas, en la que el moderador es el SCRUM Manager. En ésta reunión se hace la presentación del incremento, el planteamiento de sugerencias y anuncio del próximo Sprint.

· **Retrospectiva del SPRINT:** Después de cada Sprint, se reúnen los miembros del equipo (Aproximadamente 4 horas) y expresan sus opiniones del Sprint recién superado, con la finalidad de mejorar los procesos. Es básicamente una reunión de evaluación y mejoramiento.

El proceso SCRUM

Debido a que la metodología SCRUM es más enfocada a la organización del equipo de trabajo, así como también lo es en gran parte XP, en SCRUM a diferencia de XP que también está basado en los métodos ágiles, se divide el proyecto en periodos de 4 semanas aproximadamente, cada periodo se denomina Sprint y cada equipo SCRUM recibe una lista de pedidos a ejecutar en un sprint determinado. En la figura 13 se ve cómo los valores, artefactos y reuniones se conjugan en el proceso de desarrollo SCRUM:

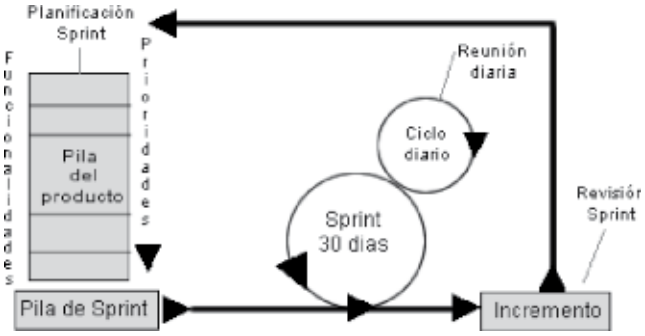

Figura 13: El proceso SCRUM. Fuente: Adaptado de The SCRUM Papers (Sutherland y Schwaber, 2007).

El proceso SCRUM (Figura 13) se compone de 5 fases las cuales contienen las actividades a desarrollar durante un periodo, comprendidas de la siguiente manera:

· **Revisión de planes de Release:** Corresponde (figura 13) a la "planificación del Sprint". Ésta fase se ejecuta una vez establecida la pila de producto y es llevada a cabo por el equipo a fin de evaluar las diferentes factibilidades de los requerimientos y estimaciones, basándose en la funcionalidad y las prioridades de la pila de producto.

· **Distribución, revisión y ajustes de estándares de producto:** Corresponde en la figura 13 a la "Pila de Sprint". En ésta fase los desarrolladores realizan los ajustes de los estándares y requerimientos mínimos, dejando todo listo para comenzar con la fase de Sprint.

· **Sprint:** Ésta Fase de aproximadamente 30 días es donde se efectúa el desarrollo del software y se llevan a cabo las reuniones, consta de las siguientes subfases: elaborar, integrar, revisar y ajustar. Estas subfases no son estrictas, pero claramente obedecen a prácticas ya mencionadas en las metodologías RUP, MSF y XP.

· **Revisión del Sprint:** Corresponde (figura 13) al "incremento". En ésta fase se revisa el Sprint y si es necesario se añaden nuevos ítems a la pila de producto. Éste proceso se repite hasta que el producto esté listo para la fase de cierre.

Cierre: En ésta fase se da lugar a la depuración y correcciones de errores (debugging), éste procedimiento se repite hasta alcanzar la calidad en el producto. Posterior a las correcciones y pruebas se realiza el Marketing y promoción del producto y al terminar ésta fase el proyecto queda cerrado.

En el ciclo de vida SCRUM cada periodo de aproximadamente 4 semanas daría como resultado una versión del producto. Al entregar esa versión, el equipo inicia de nuevo la planificación del próximo sprint e inicia de nuevo con el proceso SCRUM (Figura 13). El ciclo de vida SCRUM termina cuando el producto software haya cumplido el objetivo para el cual fue diseñado.

En conclusión, la metodología SCRUM, ofrece herramientas que permiten gestionar el equipo de trabajo hasta el punto de proponer tiempos para el proceso de desarrollo de software y para las reuniones del equipo, con la finalidad de asegurar el cumplimento de los objetivos del proyecto. SCRUM no define tácitamente las temas de bajo nivel en un proceso de desarrollo de software, tales como las relacionadas con el código que sí lo hace XP, las técnicas de modelamiento que sí lo hace RUP, y las tecnologías entre otras, lo cual deja entrever que más que una metodología sería una disciplina de trabajo para proyectos software.

VI. Conclusiones

· RUP es una metodología que usa algunas de las mejores prácticas en desarrollo de software, se adapta perfectamente a proyectos de gran escala y complejidad, así como de grandes equipos de trabajo, también cuenta con un gran nivel de aceptación entre desarrolladores.

· Por su parte MSF proporciona herramientas para llevar a cabo el éxito del proyecto, esto en cuanto a personas y procesos. Sus principios, modelos, disciplinas, conceptos y prácticas contribuyen a prevenir las causas de fracaso en el desarrollo de proyectos de software.

· Los métodos tradicionales como RUP y MSF entre otros, son bastante sistemáticos en su proceso, los cual implica altos niveles de dedicación en la planificación y documentación para posteriormente lograr el desarrollo deseado, aún así, éstos métodos han ido evolucionando a versiones de la metodología enfocada a procesos agiles, tales como RUP Agil y MSF ágil, ya que el mercado y la industria de desarrollo procura obtener resultados a poco tiempo y dispuestos al cambio constante.

· Todos los métodos tienen sus limitaciones, así como las metodologías ágiles son las más adecuadas para proyectos pequeños y medianos, no son las más adecuadas para sistemas de gran escala que requieran de interacciones complejas con otros sistemas, esto debido a que estos sistemas requieren de un nivel de precisión bastante alto, aunque no todos los métodos ágiles se basan en el desarrollo y entrega incremental, si comparten los principios del manifiesto ágil para el desarrollo de software.

· No sería conveniente implementar una metodología ágil para el desarrollo de un sistema crítico en el cual es necesario el análisis detallado de todos los requerimientos para comprender su complejidad e implicaciones, esto debido a la complejidad y la extrema precisión que pueda tener la captura de requerimientos, en los cuáles las metodologías XP y SCRUM ofrecen demasiada flexibilidad.

· Dado que los métodos ágiles hacen más explícita la importancia en el manejo del equipo y personas, se pueden pensar cómo un complemento para las metodologías que están más inclinadas a los procesos y la documentación, tales como RUP y MSF.

· Con base en la revisión de literatura y los análisis realizados en este artículo, se plantea la siguiente matriz (Tabla 1) que compara características de las cuatro metodologías, en la cual se observan las fortalezas y debilidades de cada una.

Característica	RUP	MSF	XP	SCRUM
Heredan modelos	X	X	-	-
Independiente de tecnologías	-	X	-	X
Documentación estricta	X	X	-	-
Estrictamente sistemático	X	-	X	-

Característica	RUP	MSF	XP	SCRUM
Más enfocado en los procesos	X	X	-	-
Más enfocado en las personas	-	-	X	X
Resultados rápidos	-	-	X	X
Cliente activo	-	-	X	X
Manejo del tiempo	X	X	X	X
Refactorización del código	-	-	X	-
Iterativo	X	X	X	X
Respuesta a los cambios	-	-	X	X

Tabla 1: *Matriz comparativa de las cuatro metodologías. Fuente: El autor.*

Las diferencias entre enfoques (Tabla 1) obedece a que éstas metodologías pueden ser implementadas en diferentes contextos, con diferencias en los requerimientos, en los niveles de riesgo que pueda tener cada proyecto, en los tipos de clientes y en los niveles de calidad entre otros muchos aspectos, lo cual hace que cada enfoque metodológico sea viable para los negocios con características similares o para un determinado contexto de aplicación.

VII. Referencias

[1] Abrahamsson, P.; Salo, O.; Ronkainen, J. & Warsta, J. (2002), Agile software development methods: *Review and analysis*, Espoo 2002, VTT Publications 478, Oulu.

[2] Amo, F.; Martínez, L, & Segovia, F, (2005). *Introducción a la ingeniería del software: Modelos de desarrollo de programas*, 1ª Edición. Madrid (España), Delta Publicaciones. pp 335-349.

[3] Beck, K. et al. (2001), *Manifesto for Agile Software Development*, disponible en: http://agilemanifesto.org/, recuperado: 18 de Febrero de 2011.

[4] Booch, G.; Rumbaugh, J. & Jacobson, I. (2000), *El proceso unificado de desarrollo de software*, Pearson Educación, Madrid.

[5] Booch, G.; Jacobson, I. & Rumbaugh, J.(2006), *El lenguaje Unificado de Modelado 2.0*. 2ª Edición. Addison Wesley Iberoamericana, Madrid.

[6] Del Maschi, V. et al. (2007), Practical Experience in Customization of a Software Development Process for Small Companies Based on RUP Processes and MSF, en *Portland International Center for Management of Engineering and Technology*, pp. 2440–2457.

[7] Gattaca S. A., "Presentación Metodología MSF", s. d.,disponible en: http://www.e-gattaca.com/eContent/home2.asp, recuperado:10 de Febrero de 2011.

[8] International Business Machines (s.f.), *Rational Unified Process*. IBM [citado 01 Febrero 2011] Disponible en ftp://public.dhe.ibm.com/software/rational/web/datasheets/RUP_DS.pdf

[9] Jeffries, R. (s. f.), *Extreme Programming: An agile software Development Resource*, disponible en: http://xprogramming.com/index.php , recuperado: 20 de Febrero de 2011.

[10] Leterlier, P. (s. f.), Introducción a RUP, Departamento de Sistemas informáticos y Computación (DSIC), Universidad Politécnica de Valencia (UPV), disponible en: https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20a%20RUP.doc, recuperado: 01 de Febrero de 2011.

[11] Llorens, J. (2005), *Gerencia de proyectos de tecnología de información*,Caracas, Los libros de EL NACIONAL.

[12] Microsoft, "Microsoft Solutions Framework (MSF): Disciplinas y buenas prácticas para el desarrollo e implantación de proyectos", literatura gris, Microsoft, s.d., disponible en: http://www.microsoft.com/colombia/portafolio/msf.htm, recuperado: 09 de Febrero de 2011.

[13] Microsoft (2003), "MSF Team Model Overview", white Paper, Microsoft, disponible en: http://technet.microsoft.com/es-es/library/cc784945%28WS.10%29.aspx, recuperado: 11 de Febrero de 2011.

[14] Microsoft (2003), "Microsoft Solutions Framework", White Paper, Microsoft, disponible en: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=A71AC896-1D28-45A4-880C-8B0CC8265C63, recuperado: 09 de Febrero de 2011.

[15] Mountain Goat Software (s.f.), I*ntroduction to SCRUM* - An Agile Process, disponible en: http://www.mountaingoatsoftware.com/topics/SCRUM, recuperado:18 de Febrero de 2011.

[16] Oktaba, H. & Piattini, M. (comps.), *Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies*, Hershey, ed. IG Global, pp. 71–93.

[17] Pressman, R. & Murrieta, J. (2006), *Ingeniería del software un enfoque práctico*. 6ª Edición. McGraw-Hill, pp. 67-73.

[18] Programación Extrema (s. d.), disponible en: http://www.programacionextrema.org/, recuperado: 19 de Febrero 2011.

[19] Rational Software Corporation (1998), R*ational Unified Process Best Practices for Software Development Teams*,disponible en: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf, citado 03 Febrero 2011]

[20] Rising, L, & Janoff, N. (2000),The SCRUM software development process for Small Teams, en *SOFTWARE, IEEE*, Vol. 17, No. 4, *July/August* 2000, pp. 26–32.

[21] Silva, M.; Barrera, A.; Arroyave, j. & Pineda, J. (2007), Un método para la trazabilidad de requisitos en el Proceso Unificado de Desarrollo, en *Revista EIA*, número 8, pp. 69–82.
[22] Sommerville, I. (2005), *Ingeniería de Software*, 7ª Edición, Madrid, Pearson Educación S. A. pp 76-78.
[23] Sutherland, J. & Schwaber, K. (2007), The Scrum Papers: Nuts, Bolts and Origins of an Agile Process, Boston, Scrum Inc.
[24] Traa, Johan W. A. (2006), *Rational unified process vs. Microsoft solutions framework: a comparative study*, Rotterdan, Erasmus University Rotterdam, The Netherlands.
[25] Wellington, C (2005), Managing a Project Course Using Extreme Programming, en *Frontiers in Education, 2005, FIE '05. Proceedings 35th Annual Conference*.
[26] Wells, D. (2009), *Extreme Programming: A gentle introduction*, disponible en: http://www.extremeprogramming.org/, recuperado:18 de Febrero 2011.

Oiver Adrés Pérez R. *Licenciado en Informática Educativa, Especialista en Docencia para la Educación Superior, Estudiante del Magister en Ingeniería de Sistemas y Computación, Profesor Investigador de la Facultad de Ingeniería de Unicatólica. oiverpr@gmail.com*