Universidad de las Ciencias Informáticas
Facultad 1

Estudio y propuesta de metodologías de desarrollo para los proyectos de la Facultad 1 según su alcance.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: **Roberto Arjona Miranda**

Tutor: **Ing. Dasiel Otero Dartayet**

Ciudad de La Habana, 4 de julio de 2008

"Año 50 de la Revolución"

AGRADECIMIENTOS

A mi tutor por el enorme esfuerzo brindado.

A mi familia por confiar en mí.

A Damarys Cano y su familia.

A los líderes de proyecto de la Facultad 1.

A todas aquellas personas que de una forma u otra ayudaron al desarrollo de esta investigación.

DEDICATORIA

A mis padres y mi hermana.

A mi novia Damarys y su familia.

A mi príncipe inglés Darrencito.

A mis hermanos del metal y de lucha.

A todos mis amigos.

Al estimado lector…

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____ _____

Roberto Arjona Miranda Ing. Dasiel Otero Dartayet

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título: Estudio y propuesta de metodologías de desarrollo para los proyectos de la Facultad 1 según su alcance.
Autor: Roberto Arjona Miranda.

El tutor del presente Trabajo de Diploma considera que durante su ejecución el estudiante mostró las cualidades que a continuación se detallan.

Por todo lo anteriormente expresado considero que el estudiante está apto para ejercer como Ingeniero en Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de ___.puntos.

____ días del mes de _____ del año 2007.

_____

Ing. Dasiel Otero Dartayet

RESUMEN

La Universidad de las Ciencias Informáticas está inmersa en una actividad productiva constante. Los estudiantes y profesores se vinculan a proyectos informáticos los cuales utilizan metodologías para el desarrollo de software, las cuales guían el proceso productivo. En el presente trabajo se realiza un estudio de metodologías de desarrollo RUP, XP y MSF de las cuales se hace una comparación de acuerdo a sus características para luego emitir una posible propuesta a utilizar en los proyectos de la Facultad 1 dependiendo del alcance de los mismos. Se establecerá un método para clasificar el tipo de proyecto de acuerdo a la complejidad de los mismos, para ello se entrevistó a líderes de proyecto en busca de información para luego definir una escala de requisitos funcionales de los proyectos de la Facultad 1. El mismo está compuesto por tres clasificaciones para el tipo de proyecto, los cuales son: pequeño, mediano y grande. Esta escala es una aproximación que se hace luego de la recogida de información de requisitos funcionales de los proyectos de la Facultad 1.

Palabras claves:

Proceso, metodología de desarrollo, metodología ágil.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La Universidad de las Ciencias Informáticas (UCI), primera universidad de la batalla de ideas, tiene la misión de graduar ingenieros capaces de aportar nuevas ideas, consagrados con la Revolución y dispuestos a cumplir misiones tanto dentro como fuera del país.

Actualmente en la Universidad existen disímiles proyectos dedicados a la producción de nuevos productos informáticos, pero producir un software tiene sus riesgos, si no se tiene en cuenta una metodología para su desarrollo, el proyecto se puede volver incontrolable alargando cada vez más su fecha de entrega, dejando al cliente insatisfecho al igual que a sus desarrolladores. En ocasiones para diseñar un software sólo se toman los pedidos del cliente, se hace un diseño rígido, sin metodologías de por medio y se comienza a desarrollar, sin tener en cuenta que cuando llegue la etapa de prueba el cliente solicita cambios.

Realizar un cambio en la etapa final de un software que se ha ido desarrollando con un diseño rígido puede ser imposible, habría que cambiar muchas cosas que ya estaban hechas, provocando incomodidad en los desarrolladores que no podrán entregar en tiempo el producto, incluso los clientes se sentirán molestos si no se cumplen sus pedidos. La mayoría de las veces los usuarios finales del producto se dan cuenta que no mencionaron algunas cosas, que necesitan algunas facilidades que no están implementadas en la etapa final del software, por eso es importante tener un conjunto de procedimientos, técnicas, herramientas y un soporte documental para su producción a esto se le llama metodología de desarrollo de software.

El análisis de los aspectos anteriormente mencionados conlleva a la siguiente **situación problémica**: En la Facultad 1 la mayoría de los proyectos se han desarrollado aplicando la metodología RUP sin tener en cuenta el alcance de los mismos. El desarrollo de un software guiado por una metodología de desarrollo depende de su dimensión para no poner en riesgo su plazo de entrega, costo y la optimización del uso de los recursos tecnológicos. Una metodología complicada y extensa como lo es el RUP no es recomendable aplicarla en

proyectos pequeños porque no cuenta con la cantidad de recursos humanos y tecnológicos para su aplicación.

Partiendo de la situación problémica se ha determinado como **problema científico:**

¿Qué metodología de desarrollo de software usar en los proyectos productivos de la Facultad 1 dado el alcance de los mismos?

El **objeto de estudio** es: **"**las metodologías de desarrollo" y el **campo de acción:** "los proyectos de la Facultad 1".

Visto lo anterior se llega a la siguiente **hipótesis**:

Si se estudia con profundidad el uso de las diferentes metodologías de desarrollo existentes, entonces se podrá establecer una documentación que sirva de guía para aplicar la más óptima para cada proyecto en de la Facultad 1, optimizando su costo, tiempo de entrega y uso de los recursos tecnológicos, de acuerdo al alcance que tenga el mismo.

El **objetivo general** es: lograr una documentación que permita determinar la metodología de desarrollo más óptima que se va a aplicar en los proyectos de la Facultad 1 teniendo en cuenta el tamaño de los mismos.

De lo anterior se derivan los siguientes **objetivos específicos:**

Estudiar diferentes metodologías de desarrollo de software.

Determinar una métrica que permita determinar a un líder de proyecto la escala de un proyecto.

Para lograr los objetivos planteados se cumplirán las siguientes **tareas:**

- Realizar un estudio preliminar acerca de las diferentes metodologías como RUP, XP y MSF así como el estado del arte de las metodologías de desarrollo.
- Establecer un método que permita a un líder de proyecto determinar la escala de un proyecto en la Facultad 1.

- Comparar detalladamente cada aspecto, ventaja y/o desventaja que conlleva la aplicación de una metodología con relación a las demás en dependencia con la escala del proyecto.
- Proponer el uso de la metodología más óptima para cada tipo de proyecto en la Facultad 1.

Del análisis de la hipótesis presentada se deriva “el estudio profundo de las metodologías existentes” como **variable independiente** y “una documentación bien definida acerca del uso de las metodologías de desarrollo” como **variable dependiente**.

El presente trabajo está compuesto por tres capítulos; los cuales servirán de guía en la comprensión del proceso llevado a cabo para dar cumplimiento a los objetivos trazados:

En el capítulo 1 se realizará un estudio del estado del arte de las metodologías RUP, XP y MSF. Se profundiza en temas como sus principios, sus tendencias actuales, los aspectos a seguir para elegir una metodología, se estudiará que es una metodología ágil y como se comporta.

En el capítulo 2 se analizará con más profundidad estas tres metodologías para ello se estudiará como funcionan, que características predominan en una y en otras, sus ventajas y desventajas, cómo está enfocada y pensada cada metodología para luego en el capítulo siguiente emitir una solución propuesta atendiendo a este estudio realizado.

Finalmente en el capítulo 3 se abordará sobre el estudio realizado anteriormente en el capítulo 2 para proponer el uso de metodologías para los diferentes tipos de proyecto de la Facultad 1 y se expondrá un método para clasificarlos.

CAPITULO 1.FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se procederá al estudio de las metodologías de desarrollo RUP, XP y MSF para ello se brindará un análisis de estas metodologías y que sucede en el mundo actual con ellas. Se presentará qué es una metodología, qué significa una metodología ágil, sus características principales, sus principios, cómo surgieron, su estado del arte; permitiendo brindar un conocimiento acerca de las mismas.

1.1 Definición de metodología de desarrollo

Se deberá conocer antes de estudiar las metodologías el concepto del mismo y no es más que un conjunto de procedimientos, técnicas, herramientas y un soporte documental que brinda a los desarrolladores una guía para realizar un nuevo software.

1.2 Estado del Arte

Surgimiento de las metodologías de desarrollo de software:

A medida de que las aplicaciones informáticas fueron tomando auge y ya se empezaba a considerarla como un proceso de ingeniería, disímiles metodologías de desarrollo fueron surgiendo para dar soporte al ciclo de desarrollo del proyecto. Entre las cuales, se pueden destacar algunas como MERISE elaborada en 1978 (Metodología de Análisis y Diseño de Sistemas de Información, define un conjunto de etapas: estudio preliminar, estudio detallado, implementación, realización y puesta en marcha), otras como SSADM creada en 1981.

Todos estas metodologías estaban orientadas desde sus comienzos al desarrollo de sistemas para gestionar una información que se encuentra almacenada en una o varias bases de datos, distribuidas o no. Entre los aspectos que se tratan como más críticos en estas metodologías,

los más importantes son el almacenamiento y la recuperación adecuada de la información, así como que las posibilidades funcionales que ofrezcan sean las necesarias.

En el mundo del desarrollo del software siempre se habla de buscar un modo de trabajar eficientemente, buscar la manera de evitar catástrofes, de que el proyecto no termine sin éxito. Elaborar un software es una tarea riesgosa que necesita de un proceso de desarrollo que contribuya a mejorar su calidad durante todas las fases por las que pasa, manteniendo un control y una transparencia en el proceso, es necesario terminar en el tiempo esperado y con el coste predefinido.

Implantar una metodología no es una labor de inmediato resultados, cuesta tiempo para el equipo de trabajo adaptarse a ellas, esto hace necesario adaptarlas a las necesidades de cada proyecto y de su equipo de realización. Actualmente entre las metodologías de desarrollo más utilizadas está el Proceso Unificado de Rational (RUP por sus siglas en inglés), el Microsoft Solution Framework (MSF) y la metodología XP. Las metodologías pueden seguir uno o varios ciclos de vida que indican lo que hay que obtener a lo largo del desarrollo del proyecto, lo que hay que obtener en los productos parciales y finales, pero no la forma en que se hace.

Las metodologías de desarrollo van indicando paso a paso las actividades que se van a realizar, que personas deben participar en el desenvolvimiento de las mismas y que papel deben desempeñar con el objetivo de lograr el producto final y además brindan la información necesaria para el comienzo de las mismas.

Para que una metodología sea factible debe tener:

Reglas predefinidas.

Cobertura total del ciclo de desarrollo.

Verificaciones intermedias.

Planificación y control.

Comunicación efectiva.

Fácil formación.

Actividades que mejoren el proceso de desarrollo.
Utilización de un abanico amplio del proyecto.
Soporte al mantenimiento.
Soporte a la reutilización del software.
Herramientas Case.

1.3 Principios de las metodologías de desarrollo

Sobre la metodología y su utilización se han establecido algunos principios extraídos de la práctica y la experiencia que son comunes en el desarrollo de productos informáticos en general.

- La metodología es un medio para desarrollar productos informáticos.
- La metodología abarca todo el proceso de desarrollo de los productos informáticos.
- La metodología establece las etapas, procedimientos y documentos que se han de realizar para desarrollar los productos informáticos.
- La metodología proporciona una norma para definir claramente y sin ambigüedad todos los elementos (conceptuales, estructurales y formales) que conforman un producto informático.
- La metodología se aplica flexiblemente pero sin excepción al desarrollo de todos los productos informáticos.
- La metodología se perfecciona con la práctica.

1.3.1 Aspectos a seguir para construir o elegir una metodología

En el momento de adoptar un estándar o construir una metodología, se han de considerar unos requisitos deseables, por lo que seguidamente se proponen una serie de criterios de evaluación de dichos requisitos:

- La metodología debe ajustarse a los objetivos.
- Cada aproximación al desarrollo de software está basada en unos objetivos. Por ello la metodología que se elija debe recoger el aspecto filosófico de la aproximación deseada, es decir que los objetivos generales del desarrollo deben estar

implementados en la metodología de desarrollo.

La metodología debe cubrir el ciclo entero de desarrollo de software.
Para ello la metodología ha de realizar unas etapas:

- Investigación
- Análisis de requisitos
- Diseño

La metodología debe integrar las distintas fases del ciclo de desarrollo:

• **Rastreabilidad.** Es importante poder referirse a otras fases de un proyecto y fusionarlo con las fases previas. Es importante poder moverse no sólo hacia adelante en el ciclo de vida, sino hacia atrás de forma que se pueda comprobar el trabajo realizado y se puedan efectuar correcciones.

• **Fácil interacción entre etapas del ciclo de desarrollo.** Es necesaria una validación formal de cada fase antes de pasar a la siguiente. La información que se pierde en una fase determinada queda perdida para siempre, con un impacto en el sistema resultante.

La metodología debe incluir la realización de validaciones:

La metodología debe detectar y corregir los errores cuanto antes. Uno de los problemas más frecuentes y costosos es el aplazamiento de la detección y corrección de problemas en las etapas finales del proyecto. Cuanto más tarde sea detectado el error más caro será corregirlo. Por lo tanto cada fase del proceso de desarrollo de software deberá incluir una actividad de validación explícita.

La metodología debe soportar la determinación de la exactitud del sistema a través del ciclo de desarrollo:

La exactitud del sistema implica muchos asuntos, incluyendo la correspondencia entre el sistema y sus especificaciones, así como que el sistema cumple con las necesidades del usuario. Por ejemplo, los métodos usados para análisis y especificación del sistema deberían colaborar a terminar con el problema del entendimiento entre los informáticos, los usuarios, y

otras partes implicadas.

La metodología debe ser la base de una comunicación efectiva.

Debe ser posible gestionar a los informáticos, y éstos deben ser capaces de trabajar conjuntamente. Ha de haber una comunicación efectiva entre analistas, programadores, usuarios y gestores, con pasos bien definidos para realizar progresos visibles durante la actividad del desarrollo.

La metodología debe funcionar en un entorno dinámico orientado al usuario.

A lo largo de todo el ciclo de vida del desarrollo se debe producir una transferencia de conocimientos hacia el usuario. La clave del éxito es que todas las partes implicadas han de intercambiar información libremente. La participación del usuario es de importancia vital debido a que sus necesidades evolucionan constantemente. Por otra parte la adquisición de conocimientos del usuario le permitirá la toma de decisiones correctas.

Para involucrar al usuario en el análisis, diseño y administración de datos, es aconsejable el empleo de técnicas estructuradas lo más sencillas posible. Para esto, es esencial contar con una buena técnica de diagramación.

La metodología debe especificar claramente los responsables de resultados:

Debe especificar claramente quienes son los participantes de cada tarea a desarrollar, debe detallar de una manera clara los resultados de los que serán responsables.

La metodología debe poder emplearse en un entorno amplio de proyectos software.

- ***Variedad***. Una empresa deberá adoptar una metodología que sea útil para un gran número de sistemas que vaya a construir. Por esta razón no es práctico adoptar varias metodologías en una misma empresa.
- ***Tamaño, vida***. Las metodologías deberán ser capaces de abordar sistemas de distintos tamaños y rangos de vida.
- ***Complejidad***. La metodología debe servir para sistemas de distinta complejidad, es decir puede abarcar un departamento, varios departamentos o varias empresas.

- ***Entorno.*** La metodología debe servir con independencia de la tecnología disponible en la empresa.

La metodología se debe poder enseñar:

Incluso en una organización sencilla, serán muchas las personas que la van a utilizar, incluyendo los que se incorporen posteriormente a la empresa. Cada persona debe entender las técnicas específicas de la metodología, los procedimientos organizativos y de gestión que la hacen efectiva, las herramientas automatizadas que soportan la metodología y las motivaciones que subyacen en ella.

La metodología debe estar soportada por herramientas CASE.

La metodología debe estar soportada por herramientas automatizadas que mejoren la productividad, tanto del ingeniero de software en particular, como la del desarrollo en general. El uso de estas herramientas reduce el número de personas requeridas y la sobrecarga de comunicación, además de ayudar a producir especificaciones y diseños con menos errores, más fáciles de probar, modificar y usar.

La metodología debe soportar la eventual evolución del sistema.

Normalmente durante su tiempo de vida los sistemas tienen muchas versiones, pudiendo durar incluso más de 10 años. Existen herramientas CASE para la gestión de la configuración y otras denominadas "Ingeniería inversa" para ayudar en el mantenimiento de los sistemas no estructurados, permitiendo estructurar los componentes de éstos facilitando así su mantenimiento.

La metodología debe contener actividades conducentes a mejorar el proceso de desarrollo de software.

Para mejorar el proceso es básico disponer de datos numéricos que evidencian la efectividad de la aplicación del proceso con respecto a cualquier producto software resultante del proceso. Para disponer de estos datos, la metodología debe contener un conjunto de mediciones de proceso para identificar la calidad y coste asociado a cada etapa del proceso.

Sería ideal el uso de herramientas CASE.

1.4 Tendencias Actuales

Muchas veces no se tiene en cuenta una metodología apropiada para el desarrollo de un software, cuando el proyecto es pequeño se separan de antemano los procesos en funciones y se determina el tiempo de desarrollo estimado de cada uno de ellos. En ocasiones sólo se tiene en cuenta los requerimientos de cliente. El desarrollo de un software que no lleva de por medio una metodología hará que los resultados finales sean impredecibles, no hay forma de llevar el control de los que sucede en el proyecto, los cambios que se realicen siempre van a afectar el proceso de desarrollo.

Elegir una metodología no es una cuestión simple y otro factor que puede tener cierto peso, pero que no tiene por que ser determinante del todo, es que esa metodología este soportada por determinadas herramientas aunque no es un factor determinante, contar con una herramienta adecuada es algo que puede hacer mucho más llevadera la implantación de una metodología

No va a existir un estimado de tiempo y costo preciso, por lo que el proyecto tiende a extenderse a medida que el cliente solicite cambios, que por lo general los solicita en la etapa final de desarrollo. Lo que ocurre es que con frecuencia cuando el proyecto llega a mano de los usuarios finales, estos casi siempre recuerdan que han olvidado detalles o se dan cuenta que no les gusta la forma en que se maneja el producto.

Se va a producir un estado de incomodidad en los desarrolladores que no van a poder entregar sus tareas en tiempo y los clientes quedarán insatisfechos sino se les tiene en consideración sus exigencias.

1.5 Principales Objetivos de las metodologías

Las metodologías persiguen tres necesidades principales:

- Busca robustecer aplicaciones, tendientes a una mejor calidad, aunque en ocasiones no es suficiente.
- Un proceso de desarrollo controlado, que asegure el uso de los recursos apropiados y un costo adecuado así como otros parámetros.
- Un proceso estándar en la organización, que no sienta los cambios del personal.

1.6 ¿Una metodología superior a otra?

En el transcurso del tiempo van apareciendo una amplia gama de metodologías prometiendo solucionar la crisis del software. ¿Qué se puede esperar de todos estos métodos? ¿Cumplen lo prometido? ¿Todas funcionan?

Es cierto que si se conversa con los participantes de un proyecto todos parecen encantados con sus respectivas metodologías. Programadores, jefes de proyecto, usuarios, clientes, todos parecen convencidos de las bondades de la metodología, de lo positivo que resultó para su proyecto, de que disminuyó el tiempo de desarrollo y aumentó la satisfacción general y la facilidad de uso.

¿Cómo puede ser esto así? En realidad no se descarta el hecho de que cierta metodología no haya proporcionado realmente tales beneficios. Pero ¿todas las metodologías? Unos sistemas abogan por aumentar la documentación, otros dicen que hay que aligerar los papeles del proyecto. Unas metodologías hablan de oficinas privadas y silencio para los programadores para aumentar su productividad, otras dicen que todo el trabajo hay que hacerlo en parejas y que todos los programadores deben estar juntos en una misma habitación para aumentar la agilidad y creatividad de los mismos.

En fin, cada metodología es completamente distinta. Y sin embargo todas ellas plantean un aumento de la productividad. ¿Cómo es esto posible?

1.6.1 El Efecto Hawthorne

Entre 1927 y 1932 se realizó un estudio en la Hawthorne Plant de la Western Electric Company, en Illinois. En este estudio se investigó el efecto que producían en la productividad los cambios ambientales introducidos por los investigadores. Los investigadores comprobaban que aumentando el nivel de intensidad luminosa en la planta, se aumentaba la productividad. La sorpresa fue cuando se disminuyó la intensidad luminosa y se comprobó que la productividad también aumentaba. Prácticamente se podía cambiar cualquier aspecto de la metodología de trabajo de la planta y la productividad aumentaba.

La mayor conclusión de dicho estudio es que a los trabajadores les agradaba la atención recibida durante el estudio y se esforzaban por rendir más. Otra conclusión equivalente es que la novedad introducida por el nuevo sistema saca a los trabajadores de su letargo metodológico, que a la gente le gusta la novedad y le aburre hacer las cosas siempre igual. A esto se le ha venido a denominar el Efecto Hawthorne.

¿No será este efecto lo que proporciona mejoras en todas y cada una de las nuevas metodologías? Esto parecería consistente con el resultado a largo plazo de estos sistemas, que son rápidamente olvidados y no parecen obtener tan buenos resultados cuando se convierten en la norma de trabajo.

El aburrimiento en el trabajo del programador es lo que realmente disminuye la productividad. Algunas pequeñas novedades introducidas en cada proyecto pueden hacer más por este que cualquier metodología existente.

1.7 Metodologías ágiles

En los últimos siete años ha surgido un nuevo grupo de metodologías entre las que se encuentra la llamada: Metodología ágil la cual surge para optimizar tiempo y evitar de la demora de aquellas metodologías que como RUP necesitan mucha documentación.

El término “ágil” surge en febrero de 2001 en una reunión celebrada en Utah-EEUU cuyo objetivo era hacer un esbozo de los valores y principios que permitieran a los equipos desarrollar productos que fueran capaces de responder con mayor rapidez a los cambios que pudieran surgir con la creación del proyecto y así poder desarrollar un software en la menor cantidad de tiempo posible.

Estas metodologías no necesitan tanta documentación para el desarrollo del proyecto, sino que exigen una cantidad más pequeña para cada tarea y no son tan “rígidas” como las demás metodologías.

Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

Las filosofías que rigen estas metodologías están concebidas en un manifiesto basado en valores y principios.

En el mismo se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. Las personas son el principal factor de éxito de un proyecto de 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.
- Desarrollar software que funciona más que conseguir una buena documentación. La regla a seguir es “no producir documentos a menos que sean necesarios de forma

inmediata para tomar un decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental.

- La colaboración con el cliente más que la negociación de un contrato. 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.
- Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

1.7.1 Principios del Manifiesto ágil

Plantea agil-spain los principios:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que agreguen valor al proceso.
II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.
III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre las entregas.
IV. Trabajar juntos desarrolladores y personal del negocio implicado a lo largo del proyecto.
V. Construir el proyecto entorno a individuos motivados. Proporcionarles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.
VI. El diálogo cara a cara es el método más eficiente y efectivo para transmitir información dentro de un equipo de desarrollo.
VII. El software que funciona es la medida principal de progreso.
VIII. Los procesos ágiles promueven un desarrollo sostenible. Los organizadores, desarrolladores y usuarios deberán ser capaces de mantener una paz constante.
IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares de tiempo, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y basado en esto ajusta su comportamiento.

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

Tabla 1.1 Comparaciones entre metodologías Tradicionales y ágiles.

1.8 Metodologías más usadas

1.8.1 Proceso Unificado de Rational (RUP)

(RUP, por sus siglas en inglés: Rational Unified Process). Junto con el lenguaje de modelado UML, constituye la metodología estándar más utilizada para el diseño, implementación y documentación de los sistemas orientados a objetos. Más allá de establecer un conjunto de pasos a seguir RUP es un conjunto de metodologías adaptables a las necesidades de cada organización. Los orígenes de RUP se remontan al modelo espiral original de Barry Boehm. Ken Hartman, uno de los contribuidores claves de RUP colaboró con Boehm en la investigación.

En 1995 Rational Software compró una compañía sueca llamada Objectory AB, fundada por Ivar Jacobson, famoso por haber incorporado los casos de uso a los métodos de desarrollo orientados a objetos. El Rational Unified Process fue el resultado de una convergencia de Rational Approach y Objectory (el proceso de la empresa Objectory AB). El primer resultado de esta fusión fue el Rational Objectory Process, la primera versión de RUP, fue puesta en el mercado en 1998.

Se caracteriza por:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
- Pretende implementar las mejores prácticas en Ingeniería de Software.
- Desarrollo iterativo.
- Administración de requisitos.
- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.
- Verificación de la calidad del software.

RUP es un producto de Rational (IBM). Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. El RUP está basado en 5 principios clave que son:

Adaptar el proceso: El proceso deberá adaptarse a las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear Prioridades: Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

Elevar el nivel de abstracción: Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes o esquemas (frameworks) por nombrar algunos. Esto previene a los ingenieros de software ir directamente de los requisitos a la codificación de software a la medida del cliente. Un nivel alto de abstracción también permite discusiones sobre diversos niveles arquitectónicos. Éstos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

Enfocarse en la calidad: El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

RUP es un producto de Rational (IBM), es más apropiada para proyectos grandes, dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. En proyectos pequeños, es posible que no sea posible cubrir los costos de dedicación del equipo de profesionales necesarios.

1.8.2 Microsoft Solution Framework (MSF)

Es un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, MSF es una serie de modelos que puede adaptarse a cualquier proyecto de tecnología de información. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

El Microsoft Solution Framework es adaptable, parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar. Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a 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.

- Modelo de roles

El modelo de equipos de MSF (MSF team model) fue desarrollado para compensar algunas de las desventajas impuestas por las estructuras jerárquicas de los equipos en los proyectos tradicionales. Los equipos organizados bajo este modelo son pequeños y multidisciplinarios, en los cuales los miembros comparten responsabilidades y balancean las destrezas del equipo para mantenerse enfocados en el proyecto que están desarrollando.

Comparten una visión común del proyecto y se enfocan en implementar la solución, con altos estándares de calidad y deseos de aprender.

. Sigue los siguientes principios:

- Promover comunicaciones abiertas.
- Trabajar para una visión compartida.
- Fortalecer los miembros del equipo.
- Establecer responsabilidades claras y compartidas.

- Centralizarse en agregar valor al negocio.
- Permanecer ágil y esperar cambios.
- Invertir en calidad.
- Aprender de todas las experiencias

1.8.3 Extreme Programming (XP)

Una de las metodologías de desarrollo mas usadas mundialmente es el Extreme Programming (XP) creado por Kent Beck el cual dice en una de sus frases que “Todo en el software cambia. Los requisitos cambian, el diseño, el negocio, la tecnología cambia, el equipo cambia. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar”.

O sea que esta metodología se ajusta a estos problemas. Como casi toda metodología ágil de desarrollo surgió como respuesta y posible solución a los problemas derivados del cambio en los requerimientos, XP se plantea como una metodología a emplear en proyectos de riesgo y traer consigo XP aumenta la productividad. Esta vinculada a 4 frases:

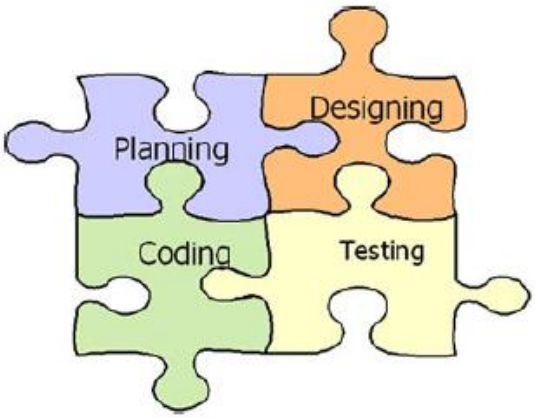


Figura 1. Funcionamiento de Extreme Programming.

Como se comentó anteriormente los distintos test se deben subir al repositorio de código acompañados del código que verifican. Ningún código puede ser publicado en el repositorio

sin que haya pasado su test de funcionamiento, de esta forma, aseguramos el uso colectivo del código.

XP es una novedosa metodología usada por el mundo entero incluso comparado con el RUP algunos optan por XP puesto que el primero utiliza muchos recursos y artefactos que al final reportan demora , pérdidas monetarias y entorpecen el entendimiento de los que allí laboran. Actualmente se efectuaron votaciones de varios desarrolladores de todo el mundo y los resultados obtenidos fueron los siguientes.

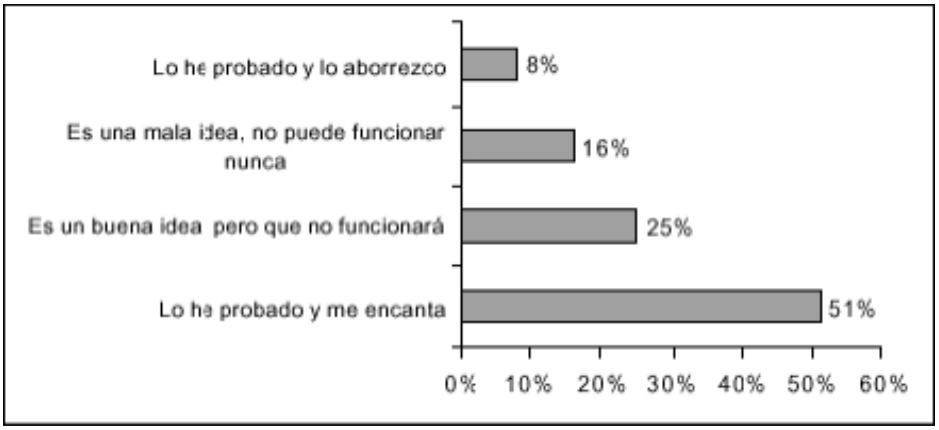


Figura 2 Votaciones sobre el uso de XP

Conclusiones

Se abordaron los principios, características, necesidades y objetivos de las metodologías, así como los principales aspectos a seguir para en un futuro utilizar la más adecuada. Se ha abarcado en este primer capítulo conceptos importantes para entender mejor el trabajo que se presenta.

En el mundo existen disímiles documentaciones y bibliografías acerca de las metodologías de desarrollo existentes. No obstante existe la tendencia a utilizar la más famosa para todos los proyectos productivos sin tener en cuenta que no siempre es la más adecuada, la más adaptable o la más óptima y obviando la existencia de las otras.

CAPÍTULO 2. ANÁLISIS DE LAS METODOLOGÍAS DE DESARROLLO RUP, XP Y MSF

Introducción

En el presente capítulo se detallarán las características principales de cada metodología de desarrollo de software mencionadas en el capítulo anterior. Se estudiará cómo trabajan, se analizará cuál es más ventajosa para un proyecto puesto que se estudiará la forma en que funcionan, ventajas y desventajas de las mismas así como sus objetivos. De este estudio se obtendrán las posibles propuestas de cuál metodología usar en un proyecto determinado.

2.1 Microsoft Solution Framework (MSF)

Más que una metodología, Microsoft Solution Framework (MSF) es una serie de modelos flexibles interrelacionados que guían a una organización sobre como ensamblar los recursos, el personal y las técnicas necesaria para asegurar que su infraestructura tecnológica y sus soluciones cumplan los objetivos de negocio.

MSF mantiene una relación clara entre los objetivos de negocio y las implementaciones tecnológicas. Plantea (Spaces 2005) que Microsoft Solution Framework (MSF) no es como tal una metodología, sino prácticas que, ajustándose al contexto de proyecto (tamaño del equipo, frecuencia de entregas) serán más o menos recomendables de aplicar. Para el proyecto se podrán seleccionar aquellas prácticas que realmente agreguen valor al proceso.

Cada práctica se compone de una secuencia de actividades y las mismas se describen en **ETVX**, un modelo de documentación de procesos introducido en los '80 que sirve para representar criterio de entrada/salida, tareas verificaciones y validaciones.

Figura 3. Actividades de MSF

Las prácticas generan resultados tangibles en forma de productos de trabajo. Estos resultados aunque análogos a los artefactos de RUP, no son necesariamente los mismos. Todo proyecto se separa en cinco fases principales como queda representado en la siguiente figura:

Figura 4. Fases de MSF análogas a RUP

2.1.1 Fases de la Metodología MSF

Visión y Alcance. Trata de unir el proyecto/producto a las experiencias de otros trabajos, el equipo debe saber lo que el cliente desea y cómo lo desea, en esta fase se definen los líderes del proyecto, se hace la primera evaluación de riesgo del proyecto, se plantean cuáles son los objetivos que se persiguen con el trabajo y hasta dónde se quiere llegar con el proyecto, además se realiza la evaluación inicial de riesgos del proyecto.

Planificación. Aquí se termina la planificación del proyecto, se realiza el proceso de diseño de la solución, se crea el plan de trabajo y se estima el costo. En fin, se planifica todo con respecto al proyecto.

Desarrollo. Se empieza a construir el proyecto tanto en la documentación como en el código y también se desarrolla la infraestructura del proyecto.

Estabilización. En esta fase se empiezan hacer las primeras pruebas, que enfatizan el uso y manipulación bajo condiciones reales; se empiezan a detectar los errores y a darle solución a los mismos.

Implantación. Se decide la tecnología base y sus componentes, por último se obtiene la aprobación del cliente.

En la siguiente tabla se muestra en lo que principalmente se centra MSF en cada una de sus fases.

Fases	Enfoque de la fase de MSF
Visión y Alcance	Se reúne un equipo del proyecto. Define la visión y el ámbito de una solución que cumplirá los objetivos del cliente. El equipo organiza entonces el proyecto y proporciona un documento de visión/ámbito aprobado. Los clústeres de funciones de administración de productos y administración de programas toman el mando en esta fase.
Planificación	Se desarrollan los procesos de diseño conceptual, lógico y físico, así como la especificación funcional. El clúster de funciones de administración de programas crea planes de proyecto que tratan el desarrollo, la comunicación y otras tareas; y cada función proporciona los datos para crear la programación del proyecto. El clúster de funciones de administración de programas toma el mando durante esta fase.
Desarrollo	El equipo crea y prueba la solución. El clúster de funciones de desarrollo toma el mando durante esta fase.

Estabilización	El equipo crea la solución piloto en preparación para el lanzamiento de producción. El clúster de funciones de prueba toma el mando durante esta fase.
Implantación	El equipo implementa la solución en todos sitios y comprueba que es estable y utilizable. La responsabilidad pasa entonces a los equipos de operaciones de TI y soporte. El clúster de funciones de administración de lanzamientos toma el mando durante esta fase.

Tabla 2.2 Cómo trabaja MSF en cada una de sus fases.

Para compensar algunas de las desventajas de las estructuras jerárquicas impuestas por los equipos de proyecto se desarrolla un modelo de roles, que no es más que la organización de los equipos de proyecto en modelos más pequeños y multidisciplinarios, para que así sus miembros puedan conformar una visión común del proyecto y guiar su enfoque de trabajo hacia la implementación de soluciones, ya que tienen que compartir responsabilidades con los demás miembros del equipo y así lograr un alto estándar de calidad y un mayor deseo de aprender en cuanto al tema en que se desenvuelve el proyecto.

Estos modelos de MSF tienen roles que corresponden a las metas de un proyecto y estos son responsables de que las mismas sean cumplidas. Todos los roles de un proyecto tienen igual importancia en su aporte al mismo, aunque estos pueden tener diferentes niveles de actividades durante las etapas de desarrollo y ninguno puede ser omitido.

MSF cuenta además con modelos que se encargan 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 de Riesgos, Modelo de Diseño de Soluciones, Modelo de Infraestructura, Modelo de Costo Total de Propiedad y finalmente el Modelo de Aplicación.

El éxito de aplicar estos modelos MSF se basa en la existencia de algunos factores como los que muestran en la siguiente figura.

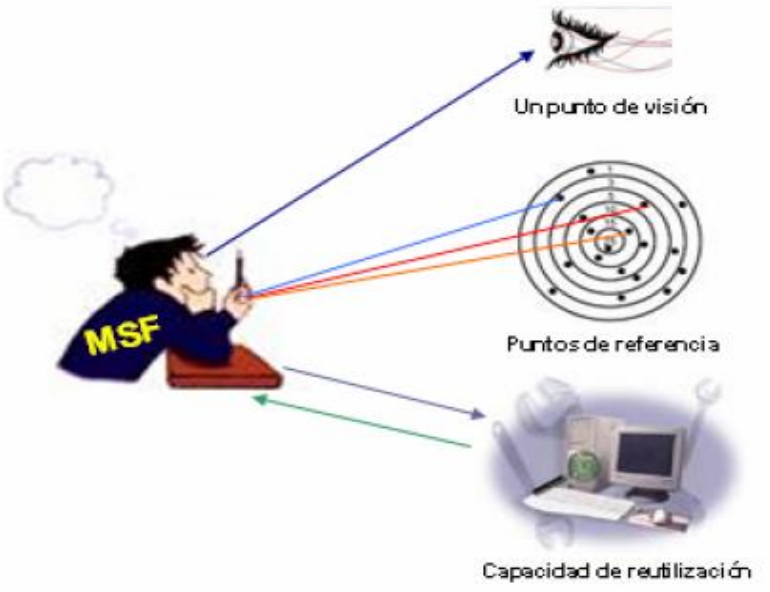


Figura 5. Factores para el éxito de los modelos de MSF

El punto de visión se necesita para proveer la guía requerida para tomar decisiones técnicas. Luego con un conjunto de puntos de referencia se puede realizar un seguimiento efectivo de la marcha de los procesos o proyectos, con énfasis en el manejo de los riesgos durante todo el ciclo de vida. La capacidad de reutilización es importante para tomar ventaja del conocimiento previo en forma estructurada y consistente en un ambiente tecnológico flexible.

El triángulo de equilibrios que se muestra en la figura 6 muestra un componente importante de MSF y ayuda a establecer el ámbito del proyecto. El triángulo de equilibrios indica que los recursos, la programación y las características son tres elementos relacionados entre sí de cualquier proyecto y que si restringe o se amplía uno o más de estos elementos, será necesario equilibrarlos con el resto. Por ejemplo, un equipo de proyecto con recursos disponibles finitos (por ejemplo, personal, presupuesto e instalaciones) que se compromete a cumplir una fecha de terminación temprana, probablemente conllevará equilibrios con

respecto al conjunto de características para cumplir la fecha, y quizás retrasará la implementación en algunos sitios remotos o en algunos sistemas.

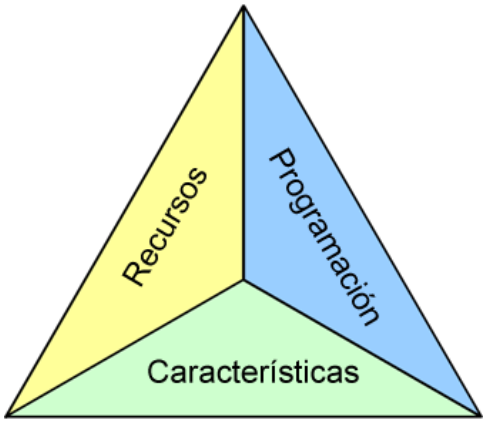


Figura 6. Triángulo de equilibrios de MSF

2.2 MSF para el desarrollo del software

Algo que ocurre frecuentemente por la necesidad de poner en orden las versiones del producto que se distribuyen a los usuarios, es que las empresas opten en algún momento por imponer a sus desarrolladores la aplicación de una metodología rigurosa para dar frente a los cambios bruscos y frecuentes en el producto, donde es casi imposible determinar la ocurrencia de estos cambios dado los vaivenes del negocio; pero por todos estos factores influyentes en la liberación de un producto final, es que las empresas se ven obligadas a permitir en algún momento que sus desarrolladores dejen de aplicar rigurosamente la metodología, dada la necesidad de no entorpecer la productividad de las versiones.

El primer problema con el proceso de desarrollo de aplicaciones es lograr que sea comprendido por el equipo de trabajo. Luego viene el problema siguiente: conseguir que sea aceptado.

Pueden surgir interrogantes como las plateadas por (Spaces 2005): ¿Por qué ese “malestar” con las metodologías? ¿Cuáles son las razones por las cuales las necesidades de cierta

rigidez en los procesos terminan endureciendo la productividad? ¿Acaso existe un punto de equilibrio entre formalismo y agilidad?

Para dar respuesta a las preguntas anteriores, (Spaces 2005) dice que a principios de los '90 *Microsoft* comenzó a recopilar a nivel interno las mejores prácticas en términos de procesos de desarrollo de software, no con la intención de establecer una metodología, sino con la idea de tener una colección de prácticas individuales de “lo que funciona”, aplicables dentro de determinados contextos.

Microsoft Solution Framework (MSF) es un conjunto de procesos, principios y probadas prácticas de desarrollo de software, altamente personalizables, escalables, totalmente integrados y pensados para brindar el tipo de guía que el usuario desea a la hora de satisfacer sus necesidades.

MSF no es más que la unión de las mejores prácticas para la administración de proyectos. Esta no es una metodología “rígida” que sirve para administrar proyectos, sino una serie de modelos que son adaptables a cualquier tipo de proyecto de tecnología de información.

Como concepto suele parecer interesante lo planteado sobre MSF en los dos párrafos anteriores, pero si encima se aplica con las herramientas adecuadas, integradas a los mismos mecanismos por los cuales se generan los productos de trabajo, los resultados son impresionantes. Sin embargo, la aplicación de una metodología obviando el uso de herramientas apropiadas e integradas, abre las puertas a la posibilidad de comenzar a incumplir con las formalidades del proceso de desarrollo.

2.3 MSF ágil para la Gestión de Proyectos Informáticos

Tanto o más importante que la tecnología que se usa en un proyecto informático es el proceso o método que se usa para controlar el proyecto. Sin duda, es una receta para el desastre no usar ninguna metodología pero, por otro lado, los años han demostrado que usar una metodología muy "protocolaria" (en el sentido de que exige una serie de pasos y

documentos largos y detallados) puede resultar improductivo, causar retrasos y desgastes. Paradójicamente, una metodología de proyectos detallista y demandante puede ser tan peligroso como no usar ninguna metodología.

MSF ágil se coloca justamente en el centro de estos dos extremos y acata los diferentes aspectos que se deben manipular en un proyecto informático, que son:

- La constitución y responsabilidades del equipo
- Las fases y entregables del proceso de desarrollo
- La vigilancia y mitigación continua de los riesgos
- La administración distribuida de responsabilidades y la escalabilidad del modelo.

Obviamente, MSF ágil establece la necesidad de realizar reuniones y crear documentos y otros entregables, pero tiene exigencias razonables y livianas, lo que aumenta las posibilidades de que el equipo informático promedio no deserte de uso de la metodología. La utilización de la metodología de gestión de proyectos MSF ágil permite introducir los conceptos necesarios para mejorar el resultado y garantizar el éxito del proyecto.

MSF ágil ayuda a implantar soluciones de tecnología según (Colombia 2000):

- Manteniendo siempre el enfoque al usuario. Es decir, ayudándolo a asegurar que la solución implantada realmente es lo que el usuario necesita, que mejorará el desempeño del usuario, y no una que va a ser desechada y olvidada al momento de su liberación.

- Proporcionando elementos valiosos a las inevitables preguntas: ¿Cuáles elementos van en el cliente?, ¿Cuáles en el servidor?

- Permitiendo desarrollar siguiendo una filosofía de reutilización de múltiples componentes, lo cual reduce el tiempo necesario para desarrollar nuevas aplicaciones, garantiza la uniformidad e interoperabilidad entre las mismas, y las hace mucho más flexibles para incorporar los cambios que sean necesarios en el futuro.

- Estableciendo un equipo de trabajo balanceado, con tareas y objetivos claramente definidos, que permitan no solo desarrollar buenos sistemas, sino también saber en todo momento cuál es el grado real de avance del proyecto, y cuáles son los riesgos que se corren si se decide introducir modificaciones al mismo una vez que el desarrollo se ha iniciado.

2.4 Conceptos Fundamentales

Se comentarán algunos conceptos importantes de MSF para comprender mejor como trabaja y como esta estructurado esta metodología.

2.4.1 Roles

Se denomina rol, según (Microsoft 2007), al papel que ejerce un actor en una actividad o proyecto. La (GUIDANCE 2005) plantea que en MSF ágil, los roles se agrupan por “equipo de pares” para representar los componentes del sistema implicados con la producción, el uso y el mantenimiento del producto.

Cada rol es responsable de representar las necesidades específicas de su trabajo y ninguno es más importante que otro, lo que explica el uso del término “pares”, pudiendo nombrarse también “equipo de iguales” para una mejor comprensión. Estos criterios proporcionan el equilibrio necesario para asegurar que el equipo produce la solución correcta.

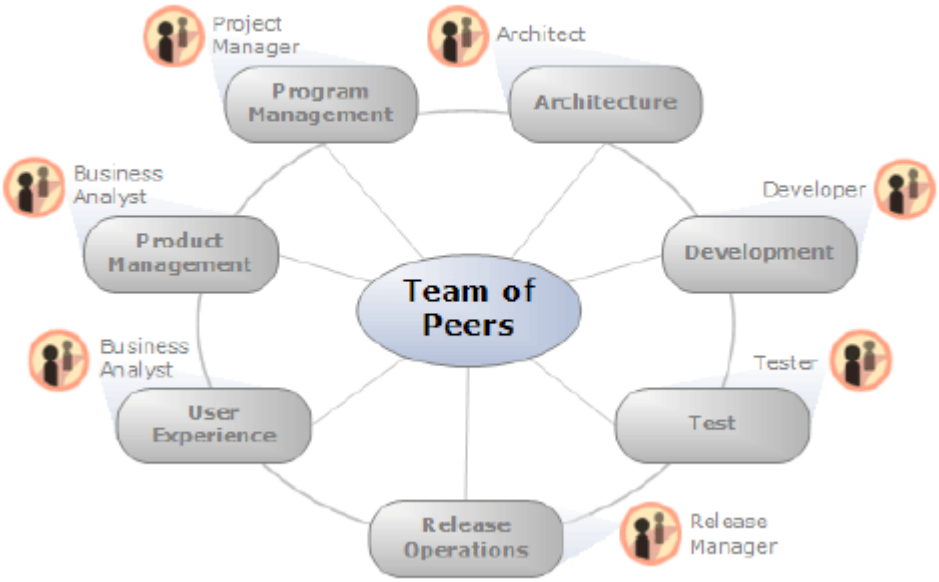


Figura 7. Roles de MSF (GUIDANCE 2005)

Los roles que intervienen dentro del proceso MSF ágil son seis fundamentalmente: analista de negocio, jefe de proyecto, arquitecto, desarrollador, tester y administrador de releases.

No necesariamente esos roles deben ser asignados a personas diferentes. Si bien es cierto que determinados roles pueden ser desempeñados por una misma persona, hay otras combinaciones en las que eso no es lo deseable.

2.4.2 Unidad de Trabajo o Work ítems

Las actividades atómicas en MFS ágil se asignan mediante el concepto de work item, que se puede describir como un registro de la base de datos que Visual Studio Team Foundation Server utiliza para rastrear trabajos asignados y el estado del mismo. El proceso MSF para el desarrollo ágil del software define work items para asignar y seguir el trabajo. La base de datos común del work item y el almacén de métrica hacen posible dar respuesta a preguntas sobre la “salud” del proyecto en tiempo real.

También según (GUIDANCE 2005), en el proceso MSF ágil los work items que por lo general se encuentran son los que siguen:

- Escenario

Tipo de work item, que va registrando una sola trayectoria de la interacción del usuario con el sistema. Por ejemplo, mientras el usuario procura alcanzar una meta, el escenario registra las medidas específicas que se tomarán para alcanzar dicha meta. Algunos escenarios registrarán una trayectoria acertada y otros no. Al escribir escenarios en un proyecto determinado, se debe ser muy específico, ya que existen muchas trayectorias posibles.

- Requerimiento de Calidad de Servicio

Este work item documenta las características del sistema tales como: funcionamiento, carga, disponibilidad, tensión, accesibilidad, utilidad y capacidad de mantenimiento. Estos requisitos toman generalmente la forma de alertas en cuanto a cómo el sistema debe funcionar.

- Riesgo

Un aspecto esencial de la administración de proyecto es identificar y manejar los riesgos inherentes al mismo. Un riesgo es cualquier acontecimiento o condición probable que pueda tener un resultado potencialmente negativo en el futuro del proyecto. El work item de tipo riesgo, documenta y sigue las debilidades técnicas o de organización de un proyecto. Cuando se requiere una acción concreta, estos riesgos pueden traducirse a tareas para atenuar el riesgo.

El ambiente debe ser tal que los individuos que identifican riesgos puedan hacerlo sin “miedo” a la consecuencia por dejar expuestas visiones tentativas o polémicas. Los equipos que crean un ambiente positivo en la administración de riesgos aciertan más en identificar los problemas que puedan afectar el desarrollo futuro del proyecto que aquellos que no lo instauran.

- Tarea

Un work item de tipo tarea comunica la necesidad de hacer un trabajo determinado. Cada rol tiene sus propios requisitos para una tarea. Por ejemplo, un tester utiliza tareas de los casos corrientes de prueba. Una tarea se puede utilizar también para señalar regresiones o para sugerir que la prueba de exploración sea realizada. También se puede usar genéricamente una tarea para asignar el trabajo por separado dentro del proyecto.

- Bug

Un bug es un work item que comunica un problema potencial existente o que ha existido en el sistema. La meta de abrir un bug es divulgar exactamente los bugs de una manera que permita que el lector entienda el impacto completo que puede acarrear el problema. La descripción en el informe de un bug debe hacer fácil la tarea cuando éste es encontrado, permitiendo así que sea reproducido fácilmente. Los resultados de la prueba que se realice deben demostrar claramente el problema. La poca claridad y la legibilidad de esta descripción afecta a menudo la probabilidad de que el bug sea fijo.

2.5 Principios de MSF ágil

Los principios para MSF ágil que define (GUIDANCE 2005) son:

- ***Aprender de todas las experiencias*** **-** La validación del cliente es a menudo la diferencia entre el valor de negocio verdadero y ficticio. Entender el asunto del valor y comunicarlo con eficacia es un factor dominante del éxito.
    - ***Alentar comunicaciones abiertas*** **-** para maximizar la eficacia individual de los miembros y optimizar la misma en el trabajo, la información tiene que estar fácilmente disponible y compartida.
- ***Trabajar hacia una visión compartida*** **-** La visión compartida asegura que todos los miembros del equipo negocien mientras se avanza en la construcción del producto. Se mejora cuando las decisiones no son arbitrarias.
    - ***Invertir en calidad*** **-** La calidad requiere la prevención del error y la verificación de la solución. Se utilizan prácticas tales como análisis del código y revisiones por pares para prevenir errores así como maximizar pruebas para encontrarlos. Todos los roles son responsables de la prevención y verificación de los errores.
- ***Permanecer ágil, esperar el cambio*** **-** Mientras más veces una organización intente maximizar el impacto del negocio de una inversión tecnológica, más se aventura en nuevos ámbitos.

- **Otorgar poder a los miembros del equipo** - Se debe confiar en el equipo al crear un producto de alta calidad mientras el mismo evoluciona. Cada cambio se debe hacer en el contexto de la creencia que el producto se encuentre listo en cualquier momento. Los liberadores deben poner a funcionar constantemente el producto y la compañía debe ensayar nuevos releases del mismo.
- **Concentrarse en agregar valor al negocio** - Planificar, ejecutar y medir el progreso y la velocidad basándose en el aumento de la entrega de valor al cliente. Se deben reducir al mínimo esas actividades que no agreguen valor al cliente porque son inútiles. Se utilizan las iteraciones para mantener la cadencia de los productos del trabajo que su cliente puede evaluar. Se debe delegar trabajo partiendo de un miembro del equipo a otro, tan eficiente como esto sea posible.

2.6 Ciclos e Iteraciones

Los ciclos describen la frecuencia con que se realizan las actividades o trabajan los productos producidos y puestos al día. Los ciclos están sobre la ejecución del proyecto y de sus tareas.
La iteración de proyecto es una jerarquía de proceso de los eventos del ciclo de vida del proyecto. Cuando se crea por primera vez el proyecto de equipo o *Team Project*, su ciclo de vida se expresa en términos de iteraciones. MSF ágil define las iteraciones como un período fijo de tiempo en el cual programar tareas. Las iteraciones generalmente son numeradas consecutivamente y siguen una a otra de manera continua. El desarrollo iterativo se presentó como antídoto al desarrollo secuencial en "cascada".

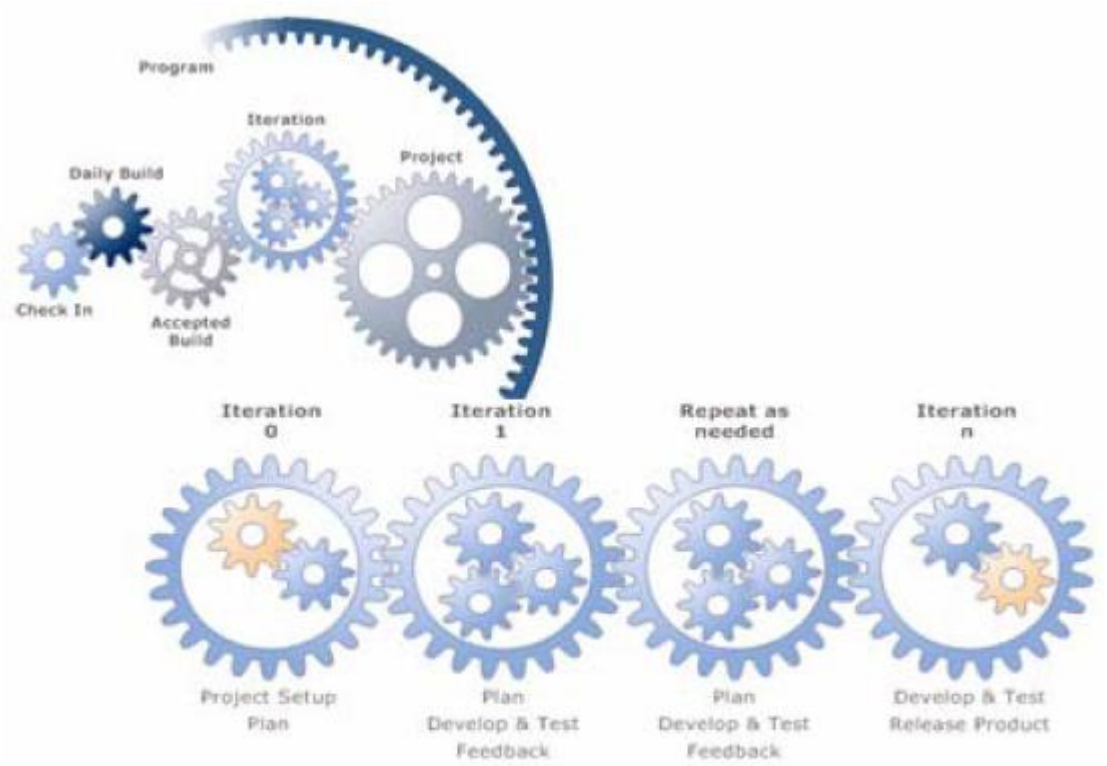

Program
Iteration
Project
Daily Build
Check In
Accepted Build
Iteration 0
Iteration 1
Repeat as needed
Iteration n
Project Setup
Plan
Plan
Develop & Test
Feedback
Plan
Develop & Test
Feedback
Develop & Test
Release Product

Figura 8. Ciclos e iteraciones en MSF.

Con independencia de la metodología utilizada, lo que todas las metodologías populares tienen en común, es que al principio de una iteración se planifica el trabajo que se va a abordar a corto plazo y al final de una iteración se sacan conclusiones de la iteración que recién se terminó. Este proceso de “mirar atrás” se debe hacer siempre con el afán de que sirva para avanzar positivamente. El final de una iteración siempre coincide con el comienzo de otra, de manera que en ese punto lo que se ve son los resultados de una iteración y estos resultados son los que deben guiar a la hora de planificar la siguiente iteración.

Evidentemente, el proceso descrito, se puede aplicar de manera anidada. Una iteración puede contener a otras. Esto es así porque al principio de un proyecto es habitual no tener un alto nivel de detalle sobre cuáles serán las iteraciones, y por lo tanto se pueden definir 3 ó 4 iteraciones que luego se irán refinando en sub-iteraciones. Otro motivo para anidar iteraciones es la posibilidad de tener, en desarrollos grandes, diferentes equipos en diferentes iteraciones, pero esto no es lo usual.

MSF es un proceso iterativo e incremental. El desarrollo iterativo se ha definido generalmente como "la técnica en la cual la definición, el diseño, la puesta en práctica y la prueba de los requisitos ocurren solapadamente dando por resultado la terminación incremental del producto de software total."

2.6.1 ¿Por qué se itera?

Hay muchas discusiones, según (Sam Guckenheimer 2006) que obligan a poner en marcha el desarrollo iterativo, por ejemplo:

- **Administración de riesgos:** El resultado deseado es incognoscible por adelantado. Para manejar riesgos, se deben probar o refutar los requisitos y diseñar incrementalmente poniendo elementos del sistema en ejecución, comenzando con los elementos de alto riesgo.

- **Economía:** En un clima de negocio incierto, es importante repasar prioridades con frecuencia y tratar las inversiones como si fueran opciones financieras. Mientras más flexibilidad se gane a través de la rentabilidad temprana y de puntos de comprobación frecuentes, más valiosas llegan a ser las opciones.
    - **Foco:** Tratando el trabajo por lotes en iteraciones pequeñas, los miembros del equipo centralizan su labor, y así los analistas pueden hacer un mejor trabajo con los requisitos, los arquitectos con el diseño y los liberadores con el código por solos citar algunos ejemplos.
    - **Motivación:** La ocurrencia de la motivación en un equipo de software propicia que los lanzamientos tempranos del producto (demos) se realicen de manera efectiva. No hay una cifra numérica de revisión de especificaciones que pueda sustituir el valor del trabajo iterado.
    - **Teoría de control:** Las iteraciones pequeñas permiten reducir el margen de error en las estimaciones y proporcionar la regeneración rápida sobre la exactitud de los planes del proyecto.
    - **Implicación del cliente:** Los stakeholders ven resultados rápidos y se enganchan más al proyecto, ofreciendo más de su tiempo, compenetración y financiamiento.
- **El aprendizaje constante:** El equipo completo aprende de cada iteración, mejorando la exactitud, la calidad y la eficacia del producto.

No obstante, el desarrollo iterativo debe ser adoptado por las organizaciones ya que en la práctica, requiere que el equipo y los managers del proyecto tengan una vista total del trabajo que harán, de la capacidad de supervisar y de ceder prioridad con frecuencia en los límites cortos de la iteración. Ésta puesta al día frecuente requiere una reserva fácilmente visible, de preferencia con la colección de datos automatizada, tal como la base de datos de work items que **VSTS** proporciona.

Según plantea el paradigma value-up del desarrollo iterativo, hay muchos ciclos en los cuales las actividades se solapan en paralelo. Una iteración es el número fijo de semanas, a veces

llamado "caja del tiempo", utilizado para programar un sistema de tareas. Usando iteraciones, el intervalo en el cual se pensaron escenarios mide el flujo del valor, determinan el proceso real, examinan embotellamientos y hacen ajustes.

2.7 Modelos de MSF ágil

Se enunciarán los diferentes modelos de MSF (que pueden ser usados individualmente o combinados). Estos modelos son adoptados por la metodología ágil y es por ello que son aplicables a MSF ágil.

2.7.1 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.

2.7.2 Modelo de equipos

El modelo de equipos de trabajo de MSF ágil facilita la estructuración de equipos para construir o implantar soluciones eficientes y de manera oportuna proyectadas hacia el mejoramiento continuo.

Este modelo se define como un equipo no jerarquizado cuyos integrantes trabajan en los seis roles interdependientes y cooperativos abordados al inicio del capítulo. Por ejemplo, los líderes de cada equipo son responsables de la administración.

La meta principal del equipo de trabajo es entregar un sistema o solución de calidad, o sea, realizar un software y que el cliente quede satisfecho con este, entregarlo en tiempo, gastando así la menor cantidad de recursos y coste, preparar al usuario para que sepa cómo usar el software, lo que implica que para que se cumplan todos estos objetivos el equipo de trabajo tiene que lograr un mejor entendimiento y trabajar en una misma línea al tiempo que se

responsabilizan con la(s) tarea(s) asignada(s). Por todas estas ventajas es que se aplica este nuevo modelo de equipos de MSF ágil.

En el modelo de equipos, las características de los miembros del equipo, ya sean la visión compartida, la comunicación, la delegación, la responsabilidad o el entendimiento, permiten la obtención de un producto con calidad en conjunto con un cliente satisfecho con el trabajo realizado. Las metas de calidad sobre las cuales se concentran los esfuerzos del equipo de desarrollo son:

- Cumplir con las expectativas del usuario.
- Entregar el sistema o solución dentro de las restricciones del proyecto (tiempo, recursos, costos).
- Identificar todos los problemas o riesgos de importancia para el usuario y manejarlos de forma oportuna.
- Asegurar que el usuario final sepa cómo usar el sistema.
- Asegurar una implantación/replicación del sistema sin contratiempos.

Este modelo de MSF ágil puede escalarse de dos maneras principales, plantea (Microsoft 2002):

1. Abstrayendo las funciones del equipo como un grupo de responsabilidades funcionales, en lugar de hacerlo como descripciones de trabajos específicos. De esta manera, las responsabilidades de cada función no están sujetas a los límites de una sola persona. Una función puede ampliarse a grupos de funciones y cada uno de los miembros se especializa en un grupo más específico de responsabilidades. Una o más personas pueden llevar a cabo estas funciones más especializadas.
2. Usando equipos de producto y calidad y equipos funcionales en diversas combinaciones para crear cualquier número de posibles estructuras de grandes equipos.

Este modelo además, propicia la agilidad para hacer frente a nuevos cambios involucrando a todo el equipo en las decisiones fundamentales, asegurándose así que se exploran y revisan los elementos de juicio desde todas las perspectivas críticas.

El modelo de equipos se basa en la premisa de que cada función presenta una perspectiva única en el proyecto y que ninguna persona por sí misma puede representar de una manera satisfactoria todos los objetivos de calidad de todas las funciones. Sin embargo, el cliente necesita una sola fuente de información que cuente con autoridad sobre el estado del proyecto, las acciones y los problemas actuales. Para resolver este dilema, el equipo de trabajo debe combinar una línea clara de responsabilidades hacia el cliente con la responsabilidad compartida con el propósito de lograr el éxito general en el proceso de desarrollo.

En resumen, las funciones del equipo de MSF ágil comparten responsabilidades en muchos aspectos de la administración del proyecto como por ejemplo, la administración del riesgo, del tiempo, de la calidad, del planeamiento, de la programación, de la selección de los miembros del equipo y de los recursos humanos.

2.7.3 Modelo de procesos

MSF ágil también tiene su modelo de procesos, que se puede utilizar en la realización del proyecto y que hace posible avizorar el cambio y controlarlo, por ende, un modelo de procesos dirige el orden de las actividades del proyecto y representa el ciclo de vida de dicho proyecto.

Algunos modelos de procesos cuyo surgimiento fue anterior a este que propone MFS ágil eran estáticos, y otros no permitían puntos de comprobación. El modelo de “cascada” y el de “espiral” son ejemplos de esos arcaicos modelos.

Como modelo de procesos superior, el de MSF ágil a través de su estrategia iterativa en la construcción de productos del proyecto, suministra una imagen más clara del estado de dichos productos en cada etapa sucesiva. El equipo puede identificar con mayor facilidad el

impacto de cualquier cambio y lidiar con él de manera efectiva, minimizando los efectos colateralmente negativos y optimizando los beneficios.

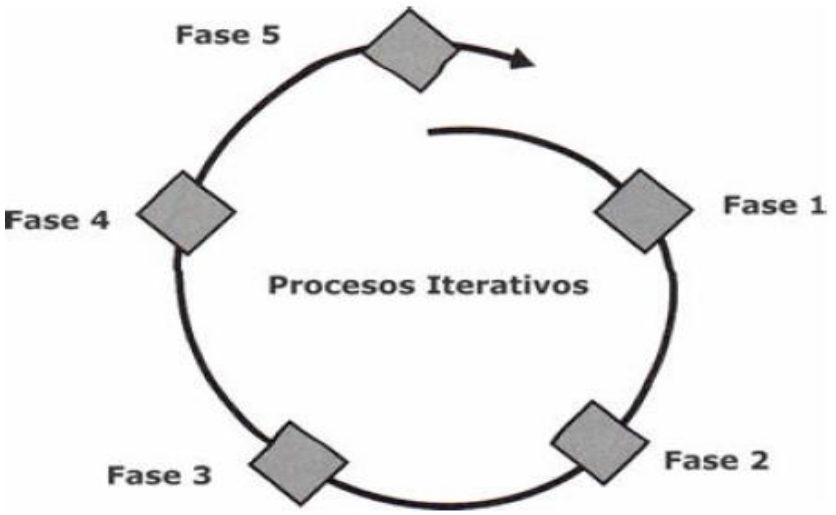


Figura 9. Procesos de MSF ágil por fases

Este modelo de MSF ágil se encarga de planificar y controlar el proyecto dirigiendo sus objetivos a resultados específicos, es iterativo y se basa en puntos de revisión.
Para profundizar más en este modelo se debe conocer que:

- Consta de cinco fases distintas
- Se basa en obtener una visión de los próximos proyectos o sistemas, prioriza el análisis de riesgos e implementa incrementalmente con puntos de revisiones frecuentes que relaciona el trabajo del equipo con las expectativas de los usuarios a lo largo de todo el proyecto. (Ivar Jacobson 1999)
- Los cuatro puntos de revisión principales están precedidos por una fase principal dentro de la cual ocurren puntos de revisión internos y se pueden generar productos entregables.(Ivar Jacobson 1999)

También fue planteado por (Ivar Jacobson 1999) que los puntos de revisión no necesariamente son puntos de congelación. En cada uno de ellos los entregables pueden

colocarse bajo un proceso de control de cambios. Cada punto de revisión establece un punto de partida que permite en forma confiable que el equipo planee y continúe hacia el siguiente punto de revisión.

En resumen, el modelo de procesos de MSF ágil se aplica para encontrar un balance en cuanto a alcance, tiempo y recursos de un proyecto y a su vez puede ser aplicado a una amplia variedad de proyectos sin pérdida del flujo del proceso.

2.7.4 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.

2.7.5 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.

2.7.6 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.

2.8 Ventajas de MSF

- La ventaja principal es que al ser un modelo desarrollado por Microsoft se puede tener mayor soporte y mantenimiento, por parte de esta empresa.
- Brinda una vinculación estrecha con los clientes y que todos los productos sean entregables.
- Esta constituida por dos modelos y tres disciplinas Modelo de Equipo y de Proceso. Disciplina de Administración de Proyecto, Administración de Riesgos y Administración de la Preparación.
- Uno de los aspectos más interesantes es el hecho que MSF viene integrado a la plataforma de desarrollo **Visual Studio 2005 Team System (VSTS)** el cual es una herramienta muy poderosa ya que VSTS centra su aplicación en proyectos de software complejos construidos por un equipo grande y tiene todos los ingredientes necesarios para un equipo distribuido de arquitectos, desarrolladores y testers.
- Metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.
- Es común ver en proyectos que no se tienen roles definidos, ver personas haciendo de todo, y haciendo tareas que se supone que no se deberían de mezclar, esta metodología cuenta con roles que a su vez tienen actividades y responsabilidades definidas para cada uno. En resumen MSF cuenta con una cómoda y eficiente planificación del equipo de desarrollo.

2.9 Desventajas de MSF

- La principal desventaja es que se torna un trabajo bastante largo, ya que para cada fase se debe documentar profundamente todo lo que se haga.
- Es un proceso muy largo pero a la vez muy solvente.
- No utiliza lenguaje modelado pues para empezar un proyecto determinado se empieza con la visión para establecer la comunicación y evaluar las limitaciones. Dicha visión se

realiza entre cliente/usuario y los analistas.

- La utilización de la metodología MSF implica la utilización de herramientas de la Microsoft, aunque tiene mayor soporte constituye una gran desventaja e importante para la investigación pues va en contra de las reglas y políticas de la Universidad ya que se debe emigrar hacia software libre.

2.10 Extreme Programming (XP) metodología ágil para desarrollo de software

Otro ejemplo de metodología ágil es la metodología *Extreme Programming* (XP), que ha sido creada para eliminar las actividades improductivas y a su vez reducir costos y frustraciones. El éxito de la misma consiste en enfatizar la satisfacción del cliente y promover el trabajo en equipo para solucionar el eterno problema del desarrollo de software por encargo y entregar a tiempo el resultado que el cliente necesita. También presenta una relación con el cliente muy diferente a la que presentan las metodologías tradicionales que se basan en la fase de captura de requisitos que se realiza antes de comenzar a realizar el software y una fase de validación que se hace posterior al desarrollo del mismo.

La programación extrema se basa en una serie de reglas y principios que se han ido gestando a lo largo de toda la historia de la ingeniería del software. Usadas conjuntamente proporcionan una nueva metodología de desarrollo software que se puede englobar dentro de las metodologías ligeras, que son aquéllas en la que se da prioridad a las tareas que dan resultados directos y que reducen la burocracia que hay alrededor tanto como sea posible. El resultado ha sido una metodología única y compacta. Por eso, aunque se pueda alegar que la programación extrema no se base en principios nada nuevos, se ha de aclarar que, en conjunto, es una nueva forma de ver el desarrollo de software.

XP se basa en unos principios y prácticas los cuales no se han hecho a priori o porque sí, sino que tienen un porqué a partir de una forma global de desarrollar software que, al menos en teoría, es más eficiente.

La metodología XP (Plantea Sánchez 2005) tiene como principio para su fundamentación acortar el ciclo de desarrollo de software y la participación del cliente desde el primer momento en que comienza el proyecto y hasta el final del mismo, esto permite que en el momento que surja el problema se le pueda dar solución de manera inmediata y así no tener que arrastrarlo hasta el final del proyecto. También responde de manera rápida a las necesidades del cliente y los cambios que en el transcurso del proyecto éstos puedan hacer a su pedido.

La metodología XP es útil para fortalecer la unidad del equipo de trabajo y lograr una mejor comunicación, de manera que se puedan explotar las capacidades de cada uno de los integrantes.

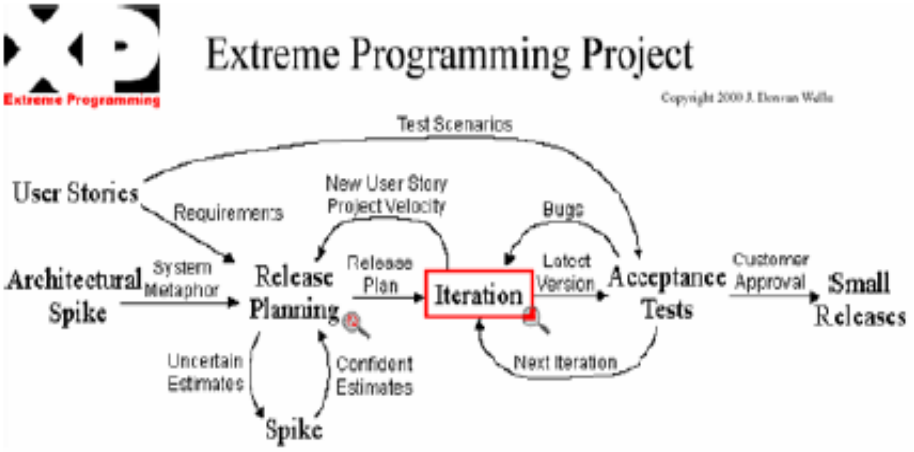


Figura 10. Proceso de desarrollo con la metodología XP

Esta metodología tiene las características siguientes:

- Existe una estrecha comunicación entre los diseñadores y programadores, los integrantes del equipo y el cliente, y entre los mismos integrantes del equipo.
- El diseño del software es sencillo.
- El grupo de prueba permite obtener una mejor comunicación entre los usuarios y el cliente y así tomar sus experiencias.

- A medida que se proponen los cambios en el producto se van realizando lo más pronto posible y se reajustan el tiempo y costo en función de la evolución real del proyecto.

Estas características demuestran que la metodología XP es la más adecuada para un proyecto que se encuentre en constante cambio y evolución a causa de los avances en la tecnología y en los negocios. Los principales valores que posee la metodología XP es la comunicación entre sus integrantes, la simplicidad, el valor y respeto tanto entre sus miembros como entre los jefes del proyecto y el cliente.

La unión y correcta aplicación de estos valores conlleva al éxito del proyecto y al de la propia metodología. Por lo general todas las demás metodologías tienen por norma que el coste del cambio en el desarrollo de un proyecto va aumentando a medida que va pasando el tiempo, sin embargo XP es innovadora en este sentido.

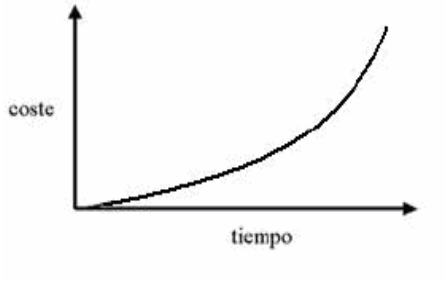


Figura 11. Curva de los métodos tradicionales

La metodología XP plantea que la curva anteriormente mostrada ya ha perdido validez y que al combinar buenas prácticas de programación y tecnología es posible lograr que la curva sea la contraria, como la figura que sigue, y esto es lo que se pretende conseguir con esta tecnología.

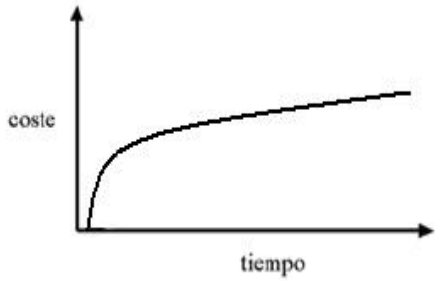


Figura 12 Curva planteada por la metodología XP

Lo planteado por XP en esta gráfica se lograría si en vez de diseñar para el cambio, se diseñara lo más sencillo posible, haciendo en cada momento solo lo que es imprescindible hacer, pues la propia simplicidad del código, y sobre todo las pruebas y la integración continua, hacen posible que los cambios puedan ser llevados a cabo tan a menudo como sea necesario.

2.11 Valores de la Programación Extrema

El proceso de desarrollo descrito en la sección anterior está fundamentado en una serie de valores y principios que lo guían. Los valores representan aquellos aspectos que los autores de XP han considerado como fundamentales para garantizar el éxito de un proyecto de desarrollo de software. Los cuatro valores de XP son:

- Comunicación.
- Simplicidad.
- Coraje.
- Realimentación.

Los partidarios de la programación extrema dicen que son los necesarios para conseguir diseños y códigos simples, métodos eficientes de desarrollo software y clientes contentos. Los valores deben ser intrínsecos al equipo de desarrollo. De los cuatro valores, quizás el que llame más la atención es el de coraje. Detrás de este valor se encierra el lema "si funciona,

mejóralo", que choca con la práctica habitual de no tocar algo que funciona, por si acaso. Aunque también es cierto que se tienen las pruebas unitarias, de modo que no se pide a los desarrolladores una heroicidad, sino sólo coraje.

2.12 Principios de la Programación Extrema

Los principios fundamentales se apoyan en los valores y también son cuatro. Se busca:

- Realimentación veloz
- Modificaciones incrementales
- Trabajo de calidad
- Asunción de simplicidad.

2.13 Fases del la Metodología XP

- Planificación del proyecto.
- Diseño.
- Codificación o Desarrollo.
- Pruebas

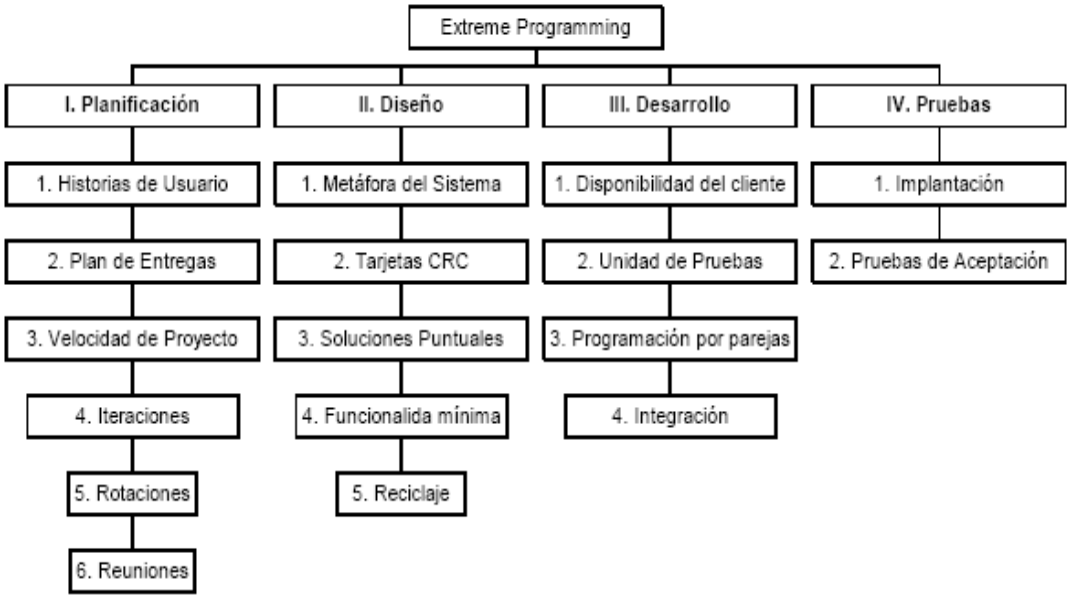


Figura 13. Fases de la Metodología XP

2.13.1 Fase 1-Planificación del proyecto

Extreme Programming hace un uso genial de la planificación de las actividades las cuales ofrecen numerosas ventajas aquí se introducen Las Historias de Usuario estas historias tienen las mismas funcionalidades que los casos de usos pero con algunas diferencias ya que el usuario escribe 3 o 4 líneas en lenguaje no técnico sin hacer hincapié en los detalles, también son usadas para estimar el tiempo de la aplicación que describen.

Crea un plan de publicaciones, en inglés "Release plan", donde se indiquen las historias de usuario que se crearán para cada versión del programa y las fechas en las que se publicarán estas versiones.

Trabaja también con un recurso muy importante denominadas iteraciones, al comienzo de cada iteración los clientes deben seleccionar las historias de usuario definidas que serán implementadas. También se seleccionan las historias de usuario que no pasaron el test de aceptación que se realizó al terminar la iteración anterior.

Velocidad del proyecto es una medida que representa la rapidez con la que se desarrolla el proyecto; estimarla es muy sencillo, basta con contar el número de historias de usuario que se pueden implementar en una iteración.

Extreme Programming: Usa la velocidad del proyecto permitiendo que todas las tareas se puedan desarrollar en el tiempo del que dispone la iteración.

Propone la Programación en pareja pues la metodología XP aconseja la programación en parejas pues incrementa la productividad y la calidad del software desarrollado.

Reuniones diarias. Es necesario que los desarrolladores se reúnan diariamente y expongan sus problemas, soluciones e ideas de forma conjunta. Las reuniones tienen que ser fluidas y todo el mundo tiene que tener voz y voto.

2.13.2 Fase 2-Diseño

La metodología XP es muy genial ya que enfoca sus esfuerzos para conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible y que sea posible implementarlo que a la larga costará menos tiempo y esfuerzo desarrollar.

Propone además un glosario de términos y una correcta especificación de los nombres de métodos y clases ayudará a comprender el diseño y facilitará sus posteriores ampliaciones y la reutilización del código.

XP recomienda para los casos de riesgos si surgen problemas potenciales durante el diseño, sugiere utilizar una pareja de desarrolladores para que investiguen y reduzcan al máximo el riesgo que supone ese problema.

Si se quiere añadir una funcionalidad extra XP recomienda por experiencia que nunca se debe añadir funcionalidad extra al programa aunque se piense que en un futuro será utilizada. Sólo el 10% de la misma es utilizada, lo que implica que el desarrollo de funcionalidad extra es un desperdicio de tiempo y recursos.

Además XP brinda la posibilidad de refactorizar la estructura y codificación de códigos ya creados sin alterar su funcionalidad Y optimizar su funcionamiento.

2.13.3 Fase 3-Codificación o Desarrollo

Crear test que prueben el funcionamiento de los distintos códigos implementados ayudará a desarrollar dicho código. Crear estos test antes ayuda a saber qué es exactamente lo que tiene que hacer el código a implementar y por tanto una vez implementado pasará dichos test sin problemas ya que dicho código ha sido diseñado para ese fin.

Se puede dividir la funcionalidad que debe cumplir una tarea a programar en pequeñas unidades, de esta forma se crearán primero los test para cada unidad y a continuación se desarrollará dicha unidad, así poco a poco desarrollará cumpliendo todos los requisitos especificados.

Como ya se comentó anteriormente, XP opta por la programación en pareja ya que permite un código más eficiente y con una gran calidad.

XP también propone un modelo de desarrollo colectivo en el que todos los programadores están implicados en todas las tareas; cualquiera puede modificar o ampliar una clase o método de otro programador si es necesario y subirla al repositorio de código. El permitir al resto de los programadores modificar códigos que no son suyos no supone ningún riesgo ya que para que un código pueda ser publicado en el repositorio tiene que pasar los test de funcionamiento definidos para el mismo.

2.13.4 Fase 4-Prueba

Uno de los pilares de la metodología XP es el uso de test para comprobar el funcionamiento de los códigos que se implementen. El uso de los test en XP es el siguiente:
Se deben crear las aplicaciones que realizarán las pruebas con un entorno de desarrollo específico para test. Hay que someter a pruebas las distintas clases del sistema omitiendo los métodos más triviales.

Se deben crear los test que pasarán los códigos antes de implementarlos; en el apartado anterior se explicó la importancia de crear antes los test que el código. Un punto importante es crear test que no tengan ninguna dependencia del código que en un futuro evaluará. Hay que crear los test abstrayéndose del futuro código, de esta forma se asegura la independencia del test respecto al código que evalúa.

2.14 Planificación del Proyecto

Para la planificación del proyecto una serie de reglas que hay que seguir para que las tres partes implicadas en este proceso (equipo de gestión, equipo de desarrollo y cliente) tengan voz y se sientan parte de la decisión tomada, que al fin y al cabo debe contentar a todos. La planificación debe de seguir premisas. La primordial es que las entregas se hagan cuanto antes y que con cada iteración el cliente reciba una nueva versión.

Cuanto más tiempo se tarde en introducir una parte esencial, menos tiempo habrá para trabajar en ella posteriormente. Se aconsejan muchas entregas y muy frecuentes. De esta forma, un error en una parte esencial del sistema se encontrará pronto y, por tanto, se podrá arreglar antes.

Sin embargo, los requisitos anteriores en cuanto a la planificación no deben suponer horas extra para el equipo de desarrollo. El argumento que se esboza es que lo que se trabaja de más un día, se deja de trabajar al siguiente. Diversas prácticas como las pruebas unitarias, la integración continua o el juego de la planificación permiten eliminar los principales motivos por los que suele ser necesario trabajar muchas horas extra.

Al planificar pueden introducirse errores, por lo cual la metodología ya tiene previsto mecanismos de revisión. Por tanto, es normal que cada 3 a 5 iteraciones se tengan que revisar las historias de los usuarios y renegociar nuevamente la planificación. A esto hay que añadir que en cada iteración también hay que realizar la planificación de la misma, lo que ha venido a llamarse planificación iterativa.

En la planificación iterativa se especifican las historias de los usuarios cuya implementación se considera primordial y se añaden aquéllas que no han pasado las pruebas de aceptación de anteriores iteraciones. La planificación de una iteración también hace uso de tarjetas en las que se escribirán tareas, que durarán entre uno y tres días (la duración la deben decidir los propios desarrolladores).

Es por eso, que el diseño seguido se puede calificar de continuo. Como se puede ver se añade agilidad al proceso de desarrollo y evita mirar demasiado adelante e implementar tareas que no estén programadas (algo que se ha venido a llamar programación just-in-time). También es cierto que no hay nada que pueda evitar que los retrasos se acumulen y como ya decía Brooks en esos casos añadir gente a un proyecto retrasado, sólo lo retrasa más.

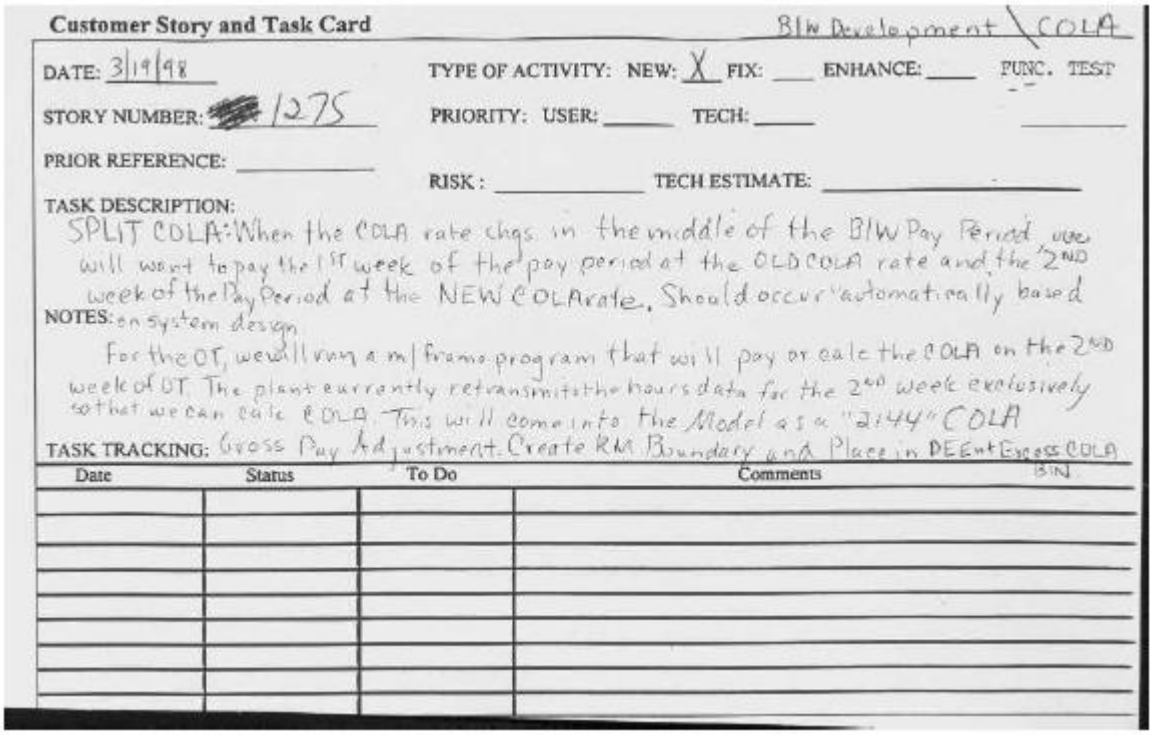
A raíz de lo anterior, se puede entender el siguiente consejo: optimizar al final. El eslogan subyacente es “haz que funcione, hazlo bien y entonces haz que sea rápido”. Y es que nunca se sabe a priori dónde puede estar el verdadero cuello de botella, así que lo mejor es no añadir funcionalidad demasiado temprano y concentrarse completamente en lo que es necesario hoy. Para la optimización siempre habrá tiempo después cuando sea prioritaria, si es que de verdad llega a serlo.

2.15 Las Historias de Usuario

Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Beck en su libro presenta un ejemplo de ficha (*customer story and task card*) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios. A efectos de planificación, las historias pueden ser de una a tres semanas de tiempo de programación (para no superar el tamaño de una iteración). Las historias de usuario son descompuestas en tareas de programación (task card) y asignadas a los programadores para ser implementadas durante una iteración.

- Las historias de usuario tienen el mismo propósito que los casos de uso.
- Las escriben los propios clientes, tal y como ven ellos las necesidades del sistema.
- Las historias de usuario son similares al empleo de escenarios, con la excepción de que no se limitan a la descripción de la interfaz de usuario.
- También conducirán el proceso de creación de los test de aceptación (empleados para verificar que las historias de usuario han sido implementadas correctamente).
- Existen diferencias entre estas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario.



Customer Story and Task Card — B1W Development \ COLA

DATE: 3/19/98 — TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

STORY NUMBER: [redacted] 1275 — PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___ — RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate chgs in the middle of the B1W Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES:
For the OT, we will run a mf frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2:44" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEntExpress COLA BIN

Date	Status	To Do	Comments

Figura 14. Tarjeta Historia del Usuario.

2.16 Roles XP

Los roles de acuerdo con la propuesta original de Beck son:

- 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.

2.17 El ciclo de vida ideal de XP

- Exploración
- Planificación de la Entrega (*Release*)
- Iteraciones
- Producción
- Mantenimiento
- Muerte del Proyecto.

2.17.1Fase I: 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.

2.17.2 Fase II: 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 la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la “velocidad” de desarrollo, establecida en 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.

2.17.3 Fase III: 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.

2.17.4 Fase IV: 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.

Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

2.17.5 Fase V: 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. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

2.17.6 Fase VI: 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.

2.18 Proceso XP

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

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 ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

El ciclo de vida ideal de XP consiste de seis fases [2]: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

2.19 Diseño, Desarrollo y Pruebas

El desarrollo es la pieza clave de todo el proceso de programación extrema. Todas las tareas tienen como objetivo que se desarrollo a la máxima velocidad, sin interrupciones y siempre en la dirección correcta.

También se otorga una gran importancia al diseño y establece que éste debe ser revisado y mejorado de forma continua según se van añadiendo funcionalidades al sistema. Esto se contrapone a la práctica conocida como "Gran diseño previo" habitual en otras metodologías. Los autores de XP opinan que este enfoque es incorrecto dado que a priori no se tiene toda la información suficiente para diseñar todo el sistema y se limita la posibilidad del cliente de cambiar de opinión respecto a las funcionalidades deseadas.

Como se verá a continuación a cambio se establecen los mecanismos para ir remodelando el diseño de forma flexible durante todo el desarrollo. La clave del proceso de desarrollo de XP es la comunicación. La gran mayoría de los problemas en los proyectos de desarrollo son provocados por falta de comunicación en el equipo, así que se pone un gran énfasis en facilitar que la información fluya lo más eficientemente posible.

Es en este punto donde entra uno de los términos estrella de la programación extrema: la metáfora. El principal objetivo de la metáfora es mejorar la comunicación entre los todos

integrantes del equipo al crear una visión global y común del sistema que se pretende desarrollar. La metáfora debe estar expresada en términos conocidos para los integrantes del grupo, por ejemplo comparando lo que se va a desarrollar con algo que se puede encontrar en la vida real.

Aunque en general el diseño es realizado por los propios desarrolladores en ocasiones se reúnen aquellos con más experiencia o incluso se involucra al cliente para diseñar las partes más complejas. En estas reuniones se emplean un tipo de tarjetas denominadas CRC (Class, Responsabilities and Collaboration - Clases, Responsabilidades y Colaboración) cuyo objetivo es facilitar la comunicación y documentar los resultados. Para cada clase identificada se rellenará una tarjeta de este tipo y se especificará su finalidad así como otras clases con las que interaccione. Las tarjetas CRC son una buena forma de cambiar de la programación estructurada a una filosofía orientada a objetos.

Aunque los grandes gurús de la programación extrema sostienen que bien hechas suelen hacer el diseño obvio, recomiendan hacer sesiones CRC en caso de que el sistema que se pretenda crear tenga un grado de complejidad grande. Este tipo de sesiones es una simulación, tarjetas CRC en mano, de las interacciones entre los diferentes objetos que puede realizar el equipo de desarrollo.

Luego de planificar y dividir las tareas, como se ha comentado en los párrafos anteriores. Lo lógico sería empezar ya a codificar. Pues no. Otro de los puntos claves de la programación extrema (y que sí es innovador en ella) consiste en que: las pruebas unitarias se implementan a la vez que el código de producción. De hecho cada vez que se va a implementar una pequeña parte se escribe una prueba sencilla y luego el código suficiente para que la pase.

Cuando la haya pasado se repite el proceso con la siguiente parte. Aunque intuitivamente esto parezca contraproducente, a la larga hará que la generación de código se acelere. Los creadores de la programación extrema argumentan que encontrar un error puede llegar a ser cien veces más caro que realizar las pruebas unitarias. La idea, en definitiva, se resumen en

la siguiente frase: “Todo código que pueda fallar debe tener una prueba". Además, hay que tener en cuenta que se hacen una vez y luego se pueden reutilizar multitud de veces, incluso por otros desarrolladores que desconocen los entresijos de esa parte o de todo el sistema, por lo que permiten compartir código (otra de las prácticas que permiten acelerar el desarrollo tal y como se verá más adelante).

Esta forma de usar las pruebas unitarias ayuda a priorizar y comprobar la evolución del desarrollo y que ofrecen realimentación inmediata. Ya no hay imprescindibles dos equipos diferenciados que desarrollan y prueban cada uno por su cuenta. Ahora el ciclo se basa en implementar una prueba unitaria, codificar la solución y pasar la prueba, con lo que se consigue un código simple y funcional de manera bastante rápida. Por eso es importante que las pruebas se pasen siempre al 100%.

Hay mucha literatura sobre las pruebas unitarias. La mayoría de los autores están de acuerdo en que cuanto más difícil sea implementar una prueba, más necesarias son. Algunos incluso dicen que entonces quizás sea porque lo que se intenta probar no es lo suficientemente sencillo y ha de rediseñarse. En cuanto a herramientas para realizar tests unitarios, existen varias para los diferentes lenguajes, lo que hace que su ejecución sea simple y, sobre todo, automáticas.

Las pruebas unitarias no se han de confundir con las pruebas de aceptación que han sido mencionadas con anterioridad. Éstas últimas son pruebas realizadas por el cliente o por el usuario final para constatar que el sistema hace realmente lo que él quiere. En caso de que existan fallos, debe especificar la prioridad en que deben ser solucionados los diferentes problemas encontrados. Este tipo de pruebas son pruebas de caja negra y se hacen contra las historias de los usuarios. Se suele tender a que sean parcialmente automáticos y que los resultados sean públicos.

Es hora entonces de ampliar el ciclo de creación de pruebas unitarias, codificación, paso de las pruebas y añadirle un paso más: la integración. La programación extrema viene a

perseguir lo que se ha venido a llamar integración continua. De esta forma, haciéndolo cada vez con pequeños fragmentos de código, se evita la gran integración final. Las ventajas de este enfoque es que permite la realización de pruebas completas y la pronta detección de problemas de incompatibilidad.

En todo desarrollo de programación extrema debería existir, por tanto, una versión siempre integrada (incluso se puede asegurar su existencia mediante cerrojos - locks). La sincronización por parte de los desarrolladores con el repositorio central debe darse como mínimo una vez al día, de manera que los cambios siempre se realicen sobre la última versión. De esta forma se puede asegurar que las modificaciones que se hacen no se estén haciendo sobre una versión obsoleta.

En cierto modo la afirmación anterior, es cierta en las metodologías tradicionales, pero en la programación extrema no: se llega a la importancia de refactorizar.

Refactorizar consiste básicamente en quitar redundancia, eliminar funcionalidad que no se usa o "rejuvenecer" diseños viejos. Tiene su justificación principal en que el código no sólo tiene porque funcionar, también debe ser simple. Esto hace que a la larga refactorizar ahorre mucho tiempo y suponga un incremento de calidad. Por cierto, tal es el énfasis que se pone en la refactorización que de la misma no se libran ni las pruebas unitarias.

Como uno de los objetivos de la programación extrema es que cualquier miembro del equipo de desarrollo puede mejorar cualquier parte del sistema, se busca que el código sea de todos. Cualquier desarrollador puede realizar cambios, corregir erratas o refactorizar en cualquier momento.

Un desarrollador que deje el proyecto (algo habitual, por otra parte no tiene por qué convertirse en un hecho catastrófico. El mejor método para conseguir que el código sea de todos es seguir unos estándares de codificación consistentes, de manera que la lectura (y refactorización) por parte del resto del equipo de desarrollo se facilite al máximo.

2.20 Prácticas de la Programación Extrema

No es fácil aplicar una nueva metodología en un equipo de desarrollo ya que obliga a aprender una nueva forma de trabajar. También obliga a abandonar cómo se hacían las cosas antes, que aunque no fuera la mejor forma posible ya se conocía. XP ha sido adoptado por un gran número de equipos en los últimos años y de sus experiencias se ha extraído una conclusión sencilla: es mejor empezar a hacer XP gradualmente.

El proceso que recomiendan los autores de XP es el siguiente: identificar el principal problema del proceso de desarrollo actual. Escoger la práctica que ayuda a resolver ese problema y aplicarla. Cuando ese haya dejado de ser un problema, escoger el siguiente. En realidad se recomienda que se apliquen las prácticas de dos en dos.

El objetivo es que las prácticas de XP se apoyan unas a otras y por tanto dos prácticas aportan más que la suma de ambas y por tanto es más fácil comprobar los resultados. El objetivo final debe ser aplicar todas las prácticas, ya que representan un conjunto completo, "si no se aplican todas se está haciendo Extreme Programming”.

Por su parte, las prácticas son las siguientes:
-El juego de la planificación (the planning game)
-Pequeñas entregas (small releases)
-Metáfora (metaphor)
-Diseño simple (simple design)
-Pruebas (testing)
-Refactorización (refactoring)
-Programación por parejas (pair programming)
-Propiedad colectiva (collective ownership)
-Integración continua (continous integration)
-40 horas semanales (40-hour week)
-Cliente en casa (on-site costumer)

-Estándares de codificación (coding standards).

El juego de la planificación. Hay una comunicación frecuente 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. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.

· **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. Éstas 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. Se mejora la estructura interna del código sin alterar su comportamiento externo.

· **Programación en parejas**. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Las principales ventajas de introducir este estilo de programación son: muchos errores son detectados conforme son introducidos en el código (inspecciones de código continuas), por consiguiente la tasa de errores del producto final es más baja, los diseños son mejores y el tamaño del código menor (continua discusión de ideas de los programadores), los problemas de programación se resuelven más rápido, se posibilita la transferencia de conocimientos de programación entre los miembros del equipo, varias

personas entienden las diferentes partes del sistema, los programadores conversan mejorando así el flujo de información y la dinámica del equipo, y finalmente, los programadores disfrutan más su trabajo. Dichos beneficios se consiguen después de varios meses de practicar la programación en parejas.

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

· **Integración 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.

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

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

· **Estándares de 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.

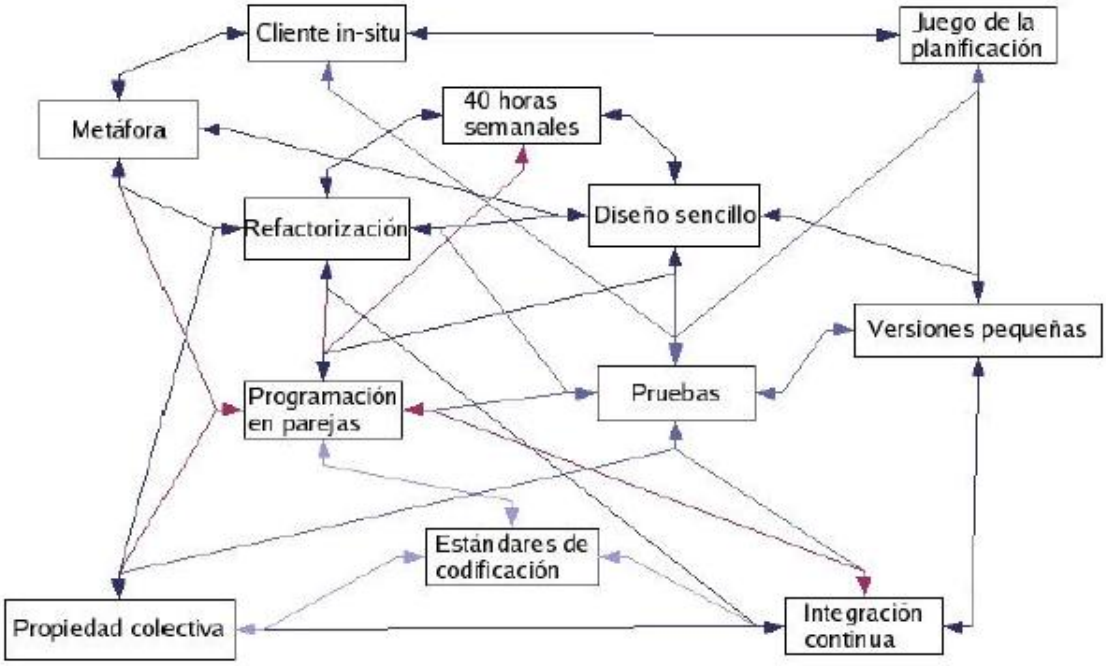


Figura 15. Las Prácticas se interrelacionan entre sí.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. Esto se ilustra en la Figura 15, donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí. La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica. El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

2.21 Ventajas del Extreme Programming

- Existe una estrecha vinculación con el cliente de tal forma que el mismo se llega a considerar un miembro más del equipo de proyecto.
- Es muy cómodo y fácil de aprender.
- Es una de las metodologías más veloz que se conoce con resultados satisfactorios mundialmente.

- A través de las historias de los usuarios los desarrolladores van directo al grano quieren funcionalidades y no complicaciones.
- Abogan por un diseño muy simple mientras mas simple mejor, mayor entendimiento del resto de desarrolladores y del propio cliente.
- Como la mayoría de las metodologías ágiles la **programación en parejas** es una ventajosa arma la cual permite una mayor velocidad de implementación y entendimiento pero también cuando se implementa muchos errores son introducidos y es posible corregirlos con esta modalidad además los programadores se sienten contentos.
- No tiene cuidado de no asignar demasiadas tareas organizativas sobre los desarrolladores (como modelado, generación de documentación, etc.), cuyo efecto se minimiza por medio de un representante del cliente.
- Se obtiene un producto en tiempo récord aunque puede estar defectuoso pero si el cliente aprueba su entrega y hace las funcionalidades que se acordó en los contratos. El cliente puede apoderarse del producto culminando así el proyecto velozmente.
- Hace uso de la **Refactorización** de la cual se habló anteriormente la cual permite reutilizar un código una y otra vez.
- Se realizan pruebas unitarias constantemente cada línea de código es testeada continuamente al terminar alguna implementación así evitará futuros errores de código.

2.22 Desventajas del Extreme Programming

- No se enfoca en la calidad del producto por lo puede salir una versión defectuosa y aunque algunos clientes queden satisfechos, algunos no se contentan y no aceptan el producto lo cual indica una pérdida de esfuerzos, costo y tiempo para el equipo de proyecto.
- Poca documentación, aunque la mayoría de los desarrolladores huyen de la abultada documentación e incluso algunos estén contentos de no llevar en XP la

burocracia que conllevan algunas metodologías como RUP, MSF esto constituye a la vez una desventaja crucial ya que la mayor parte del razonamiento y entendimiento está centrada en las mismas. Si alguna persona desea ver dicha documentación se puede ver limitado ya que XP es pobre en ese aspecto.

- Poca organización y debido a este aspecto XP se recomienda para proyectos a corto plazo ya que para un equipo grande XP no alcanza para organizar todo ese volumen de procesos y de personas.

2.23 Rational Unified Process (RUP) para el desarrollo de software

Hoy en día las empresas han desarrollado muchos sistemas de información y siguen creciendo con nuevos desarrollos y dando mantenimientos a los sistemas actuales. RUP constituye un marco de trabajo para el desarrollo de procesos que habilita el desarrollo de software de una manera organizada, con una asignación precisa de responsabilidades entre los miembros de un equipo, y con un enfoque rigurosamente centrado en las necesidades de los usuarios.

El mismo está basado y formado por componentes de software los cuales están interconectados por interfaces bien definidas. También posee una importantísima herramienta como es el Unified Modeling Languaje (UML) en otras palabras Lenguaje Unificado de Modelado la cual forma una parte esencial dentro de la construcción del software ya que de acuerdo como se levanten los requisitos con los casos de usos estos serán lo que guiará la arquitectura del futuro sistema y la arquitectura determinará la selección de los casos de usos.

RUP esta constituido por decirlo de alguna manera por tres piezas fundamentales y es que esta bien dirigido por los casos de usos, esta centrado por la arquitectura que se defina y es iterativo e incremental. Y aunque los casos de usos guían el proceso no se desarrolla aisladamente se desarrolla a la vez de la arquitectura del sistema.

2.24 Objetivos Principales

• Tener un control total en el Ciclo de Vida del Desarrollo del Software

• Tener la documentación de todos los Procesos de Negocios y Diagramas del Sistema

• Visualizar los beneficios que conlleva la implantación de RUP como el nuevo marco metodológico del Desarrollo de Sistemas.

• Conocer y aplicar las cuatro fases de la metodología Rational Unified Process - RUP para el desarrollo de un producto de software.

2.25 La vida de un software (RUP)

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema. Cada ciclo constituye una versión del producto para los clientes. Los mismos están constituidos por 4 fases inicio colaboración, construcción y transición. En cada una de sus fases se subdividen en iteraciones como muestra la figura 16.

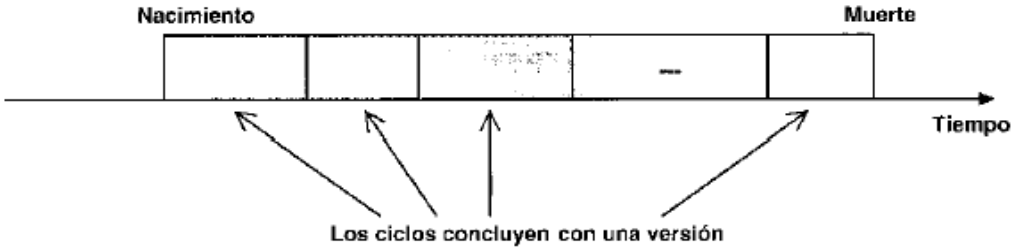


Figura.16 Vida de un software en RUP.

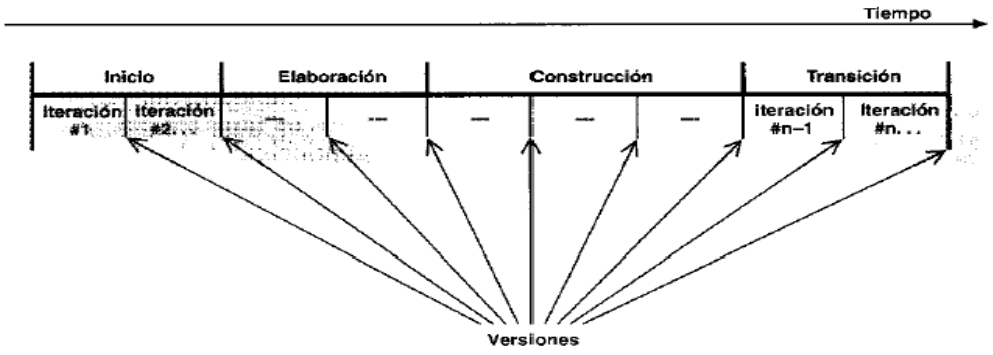


Figura.17 Un ciclo con sus fases e iteraciones.

Como se dijo anteriormente cada ciclo produce una nueva versión del sistema, y cada versión es un producto preparado para su entrega. Consta de un código fuente incluido en componentes que puede compilarse y ejecutarse, además de manuales y otros productos asociados. El producto debe adaptarse a las necesidades de los usuarios más que el código máquina que se ejecuta.

2.26 ¿Cómo funciona el Proceso Unificado (RUP)?

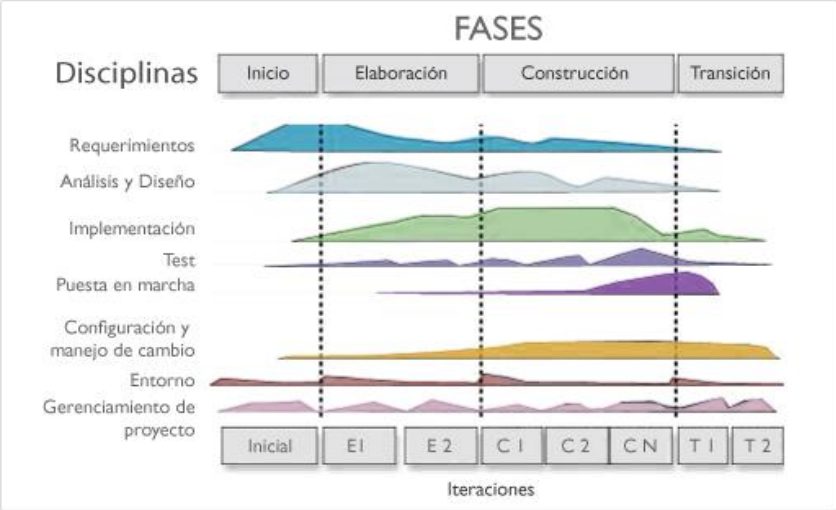


Figura 18. Fases y Flujos de trabajo de RUP.

2.27 Fases de RUP

El RUP consta de una serie de ciclos los cuales se le denomina la vida de un sistema de software cada ciclo consta de 4 fases:

- **Inicio**

Durante la fase de inicio se desarrollan ideas a partir de una buena descripción del producto que se llevará a cabo y se presenta el análisis del negocio para el producto. Esencialmente en esta fase se deberá dar soluciones a las funcionalidades del sistema para los usuarios

importantes por medio de el modelado de casos de usos del negocio que contenga los casos de usos más críticos, también saldrá la incipiente arquitectura que luego se fortalecerá a medida de que desarrolle el sistema.

- **Elaboración**

Se especifica en detalle la mayoría de los casos de usos del producto y luego se diseña la arquitectura del sistema. La misma será expresada en forma de vistas de todos los modelos del sistema. Existen numerosas vistas arquitectónicas como el modelo de casos de uso, modelo de análisis y diseño, modelo de despliegue, modelo de implementación (para probar si la arquitectura es ejecutable).El resultado de esta fase no es mas que una línea base de la arquitectura.

- **Construcción**

A lo largo de esta fase se construye el producto, las descripciones evolucionan hasta convertirse en un producto listo para entregar a los usuarios. La mayor parte de los recursos se gasta en esta fase. Ya en esta etapa la arquitectura es sólida y robusta aunque se puede pensar como hacerla estructurar mejor pues los arquitectos recibirán siempre propuestas de cambios de menor importancia, sin embargo al final de esta fase se el producto tendrá todos los casos de usos que la dirección acordó para esa versión. Eso no quita que el mismo no este libre de defectos todos o casi todos estos defectos se corregirán en la fase de transición.

- **Transición**

Ya una vez en esta fase el producto se convierte en una versión beta la cual será testeada por pequeño de usuarios con experiencia para describir algún fallo que este tenga. De existir algún problema le son informados a los desarrolladores los cuales procurarán una solución inmediata. Esta fase es fundamental ya que no solo contribuye a la fabricación del producto como tal y eliminar las deficiencias que este tenga sino de proporcionar al cliente una línea de ayuda así como soporte luego de la entrega del producto.

El primer término a analizar en la configuración serían los roles a desempeñar por los miembros del equipo de desarrollo y para introducir mejor el tema se brinda el siguiente concepto enunciado por el RUP:

“Un rol es una definición abstracta de un conjunto de responsabilidades, para realizar actividades y producir artefactos, los roles son realizados típicamente por un individuo, o un conjunto de individuos trabajando juntos como un equipo por lo que un miembro del equipo de proyecto puede cumplir diferentes roles.” (Rational Software Corporation, 2003)

Como segundo término se detallarán las actividades que estos roles realizan, las mismas no son mas que “un conjunto de operaciones o tareas propias de una persona o entidad” (WordReference.com, 2005), estas actividades están estrechamente relacionadas y funcionalmente acopladas y son mejor realizadas por un individuo.

2.28 Roles importantes que intervienen en RUP

El RUP clásico en su totalidad propone una serie de roles que abarcan todo el ciclo de vida del proyecto, los mismos los agrupa según su funcionalidad, a continuación se da un breve resumen de ellos:

• **Analistas**: agrupa un conjunto de roles principalmente envueltos en la investigación y obtención de requerimientos
• **Desarrollador:** Organiza el conjunto de roles principalmente implicados en el diseño y la implementación del software
• **Administradores**: Organiza el conjunto de roles principalmente implicados en la administración y configuración del proceso de ingeniería de software.
• **Producción y Soporte:** Son aquellos no directamente relacionados con la administración, desarrollo y prueba de software, pero son necesarios para soportar el proceso de desarrollo de software o para producir materiales adicionales requeridos por el producto final del sistema.
• **Probadores:** Organiza el conjunto de roles que tratan con las habilidades únicas especificas para prueba.
• **Roles adicionales:** Son aquellos que no se acomodan dentro de ningún conjunto de roles anteriores.

2.29 Algunos artefactos que genera RUP

Se analizarán los artefactos generados de acuerdo a los roles y sus actividades, los mismos son producto del trabajo final o intermedio y son producidos y usados durante el proyecto capturando diferentes informaciones.

El RUP propone un número grande de ellos, de los cuales para esta configuración se mantuvieron algunos y se agregaron otros como se explicó en el capitulo anterior, a continuación se detallarán los artefactos según el orden de las disciplinas del RUP:

- **Modelado del Negocio**

  - **Glosario del negocio:** Define los términos importantes usados en el modelado del negocio, una parte del proyecto.
  - **Reglas del negocio:** Las reglas de negocio describen políticas que deben cumplirse o condiciones que deben satisfacerse, por lo que regulan algún aspecto del negocio.
  - **Modelo de casos de uso del negocio:** Es un modelo que describe los procesos de un negocio (casos de uso del negocio) y su interacción con elementos externos (actores), donde las funciones que el negocio pretende realizar y su objetivo básico es describir cómo el negocio es utilizado por sus clientes y socios. El mismo esta conformado por:

    - Actor del negocio: es cualquier individuo, grupo, entidad, organización, máquina o sistema de información externos; con los que el negocio interactúa. Lo que se modela como actor es el rol que se juega cuando se interactúa con el negocio para beneficiarse de sus resultados.
    - Caso de uso del negocio: un proceso o caso de uso del negocio es un grupo de tareas relacionadas lógicamente que se llevan a cabo en una determinada secuencia y manera y que emplean los recursos de la organización para dar resultados en apoyo a sus objetivos.
  - **Modelo de análisis del negocio:** describe las realizaciones de casos de uso del negocio por la interacción entre los trabajadores y entidades del negocio. Este sirve como abstracción de cómo los trabajadores y entidades del negocio deben relacionarse

y cómo ellos colaboran para desarrollar los casos de uso del negocio. Este modelo esta compuesto por:

- **Realización de casos de uso del negocio:** Mientras un caso de uso del negocio describe que pasos deben realizarse para producir un resultado observable para ciertos actores del negocio, una realización de caso de uso describe cómo estos pasos se realizan dentro de la organización. Se describen los casos de uso del negocio desde una perspectiva externa, mientras que la realización de un caso de uso se describe desde una perspectiva interior. Las realizaciones pueden ser a través de:

  - ✓ Descripción textual: es un documento donde se hace la descripción del caso de uso en detalles.
  - ✓ Roles participantes: es una lista con nombres y breve descripción de los actores, trabajadores, entidades y eventos del negocio que aparecen como roles u objetos en los diagramas de actividad y clases.
  - ✓ Diagrama de actividad: describe un proceso que explora el ordenamiento de las tareas o actividades que cumplen los objetivos del negocio y que satisfacen el propósito entre los actores externos y los trabajadores internos del negocio.
  - ✓ Diagrama de clases: muestra las asociaciones, agregaciones y generalizaciones entre los trabajadores y entidades del negocio.

- **Trabajadores del negocio:** Es una abstracción de un humano o sistema del software que representa un papel dentro de las realizaciones de caso de uso. Un trabajador del negocio colabora con otros trabajadores, se notifica de eventos del negocio y manipula las entidades del negocio para realizar sus responsabilidades.
- **Entidades del negocio:** Son una abstracción de elementos persistente dentro del negocio, estas representan algo que los trabajadores toman, manipulan, modifican y utilizan.

- **Sistemas de negocio:** Encapsula un conjunto de roles y recursos que juntos cumplen cabalmente un propósito específico y define un conjunto de responsabilidades con las cuales estos propósitos pueden ser alcanzados.
- **Eventos del negocio:** Representa una ocurrencia significativa en las actividades del negocio que requieren acción inmediata.

- **Requerimientos**
  - **Glosario:** Define los términos importantes utilizados en el proyecto, es un refinamiento del glosario del negocio.
  - **Visión:** Define la perspectiva de los involucrados del producto a desarrollar, detallando en sus términos las necesidades y rasgos principales, además de ser un refinamiento de la visión del negocio.
  - **Requerimientos del software:** Condición o capacidad que el sistema debe cumplir.
  - **Atributos de los requerimientos:** Es un almacén de requerimientos de proyecto, atributos y dependencias para asistir el cambio dirigido desde una perspectiva de los requerimientos.
  - **Especificación de los requerimientos de software:** Captura los requerimientos de software para el sistema completo.
  - **Plan de administración de requerimientos:** Describe los artefactos de requerimientos, los tipos y sus respectivos atributos, detallando la información a recolectar y los mecanismos de control a ser usados para la medición y cambios controlados en los requerimientos del producto.
  - **Modelo de casos de uso:** Es un modelo de las funciones deseadas del sistema que contiene actores, paquetes, casos de uso y sus relaciones. Es usado para como una entrada esencial a actividades en diseño y prueba, esta conformado por:
    - Paquetes de casos de uso: es una colección de casos de uso, actores, relaciones, diagramas y otros paquetes, usados para estructurar el modelo de casos de uso y dividirlo en pequeñas partes.

- Casos de uso: se define como un conjunto de instancias de casos de uso donde cada instancia es una secuencia de acciones a ejecutar en un sistema que producen un resultado observable de importancia para un actor en particular.
- Actores: se define como un conjunto lógico de roles que los usuarios del sistema pueden jugar cuando interactúan con él. Una instancia de un actor puede ser jugada por un individuo cualquiera o un sistema externo.

Realización de casos de uso del sistema:

- ✓ Descripción textual: lleva el mismo contenido de la descripción textual del negocio lo que se le agrega al final el prototipo de la interfaz.
- ✓ Roles participantes: es una lista con nombres y breve descripción de los actores y entidades del sistema que aparecen como roles u objetos en los diagramas de actividad.
- ✓ Diagrama de actividad: describe un proceso que explora el ordenamiento de las tareas o actividades que cumplen los objetivos del sistema y que satisfacen el propósito entre los actores externos y el sistema.

- **Prototipo de interfaz de usuario (diseño):** Es un ejemplo de diseño de la interfaz de usuario.
- **Solicitudes de los Stakeholder:** Este artefacto contiene cualquier tipo de solicitudes de un Stakeholder que el sistema tiene que desarrollar.

• Diseño

- **Modelo del Diseño:** Es un modelo de objetos que describe la realización de los casos de uso y es usado como una abstracción del modelo de implementación y su código fuente y como una entrada esencial en actividades de implementación y prueba, además está conformado por:

- Realización de casos de uso del diseño: describe como un caso de uso articular es realizado dentro del modelo del diseño en términos de colaboración de objetos.

- ✓ Diagramas de interacción: son usados para mostrar como los objetos actúan para ejecutar el comportamiento de un caso de uso en particular. Existen dos formas para esto:
  - Diagrama de colaboración: destaca la organización de los objetos que participan en una interacción.
  - Diagrama de secuencia: destaca la ordenación temporal de los mensajes.
- ✓ Objetos participantes: breve descripción y clasificación de los objetos participantes.

- ➢ Clases del diseño: es la descripción de un conjunto de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos y semántica.
- ➢ Paquetes del diseño: es una colección de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes y es utilizado para estructurar el modelo del diseño en partes más pequeñas.
- ➢ Subsistemas del diseño: son una forma de organizar los artefactos del modelo de diseño en piezas más manejables. Puede constar de clases del diseño, realizaciones de casos de uso y otros subsistemas.

Modelo de Datos: Describe la representación lógica y física de los datos persistentes usados por la aplicación.

Documento de la arquitectura del software: Proporciona una descripción comprensiva de la arquitectura del sistema, usando un número de vistas arquitectónicas diferentes para representar los aspectos desiguales del sistema.

Modelo de Despliegue: muestra la configuración de los nodos procesadores y los dispositivos a poner en funcionamiento y los enlaces de comunicación entre ellos mediante los protocolos.

Prototipo de interfaz de usuario (funcional): es un ejemplo de interfaz de usuario construido para explorar y/o validar el diseño de la interfaz de usuario.

- **Implementación**

  - **Modelo de Implementación:** Representa la composición física de la implementación en términos de subsistemas de implementación y elementos de implementación (archivos de código fuente y ejecutables, datos, etc.)
    - Subsistemas de implementación: es un conjunto de elementos de implementación para estructurar el modelo de implementación en pequeñas partes que puedan ser separadamente integradas y probadas.
    - Elementos de implementación: son la parte física que construye una implementación incluyendo archivos, directorios, archivos de código de software (código binario o ejecutable), ficheros de datos, archivos de documentación, como archivos de ayuda en línea.
  - **Build:** es una versión operacional del sistema o parte de un sistema que demuestra un subconjunto de capacidades que serán proporcionadas en el producto final.
  - **Plan de Construcción e Integración:** proporciona un plan detallado para la integración dentro de una iteración.

- **Prueba**

  - **Plan de prueba:** Contiene la definición de metas y objetivos de prueba dentro del alcance del proyecto, los objetos a ser probados, la estrategia a ser tomada, los recursos requeridos y los entregables a producir, además de los responsables de las tareas previstas.
  - **Resultados de prueba:** Es una colección de información resumida determinada del análisis de una o más anotaciones de prueba o solicitudes de cambio, proporcionando una valoración relativamente detallada de la calidad de los objetos de prueba.

Datos de prueba: Son la definición de una colección de valores de entrada de prueba que son usados durante la ejecución de una prueba.

Casos de prueba: Es un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollado para cumplir un objetivo en particular ó una función esperada.

No conformidades: Documento donde se muestra el incumplimiento de los requerimientos.

- **Despliegue**

Producto: Es definido como una unidad de despliegue que debe ser empaquetada para la venta y distribución. El mismo esta conformado por:

- Lista de materiales: describen las partes constituyentes y los cambios hechos de una versión determinada de un producto y donde pueden ser encontradas las partes físicas.
- Artefactos de instalación: se refieren al software e instrucciones documentadas requeridas para instalar el producto.
- Unidad de despliegue: documentos y artefactos de instalación suficientemente completos para ser descargados, instalados y corridos en un nodo.

- **Material de soporte para usuarios finales:** Ayudan a los usuarios finales en conocimiento, uso, funcionamiento y manutención del producto.
  - Materiales de entrenamiento: hace referencia al material que es usado en capacitación o cursos para ayudar al usuario final con el uso del producto, funcionamiento y/o mantenimiento.
- **Plan del despliegue:** Describe el conjunto de tareas necesarias para instalar y probar el producto desarrollado tal que pueda ser transformado eficazmente a la comunidad del usuario.

• Administración del proyecto

Plan de desarrollo del software: Es un artefacto compuesto, comprensivo que recoge toda la información requerida para administrar el proyecto, describe el acercamiento al desarrollo

del software y es el plan de alto nivel generado y usado por los administradores para dirigir el esfuerzo de desarrollo. El incluye un número de artefactos como:

- Plan de administración de riesgos: detalla como administrar los riesgos asociados con un proyecto, las tareas de administración de riesgos que serán realizadas, las responsabilidades asignadas y cualquier recurso adicional requerido para esta actividad
- Plan de control de la calidad: es un artefacto que provee una vista clara de cómo el producto, artefacto y calidad del proceso deben ser asegurados.

- **Plan de iteraciones:** Es un conjunto de actividades y tareas en una secuencia de tiempo, con recursos asignados y comprendiendo dependencias de tareas para la iteración.
- **Lista de riesgos:** Es una lista clasificada de riesgos conocidos y abiertos para el proyecto, clasificada en orden de importancia decreciente y asociado con mitigaciones específicas o acciones de contingencia.
- **Registro de revisión:** Este es un documento creado para capturar los resultados de una actividad de revisión en la cual uno o más artefactos son revisados.

2.30 Ventajas del RUP

- RUP es una metodología muy integral que abarca una amplia gama de procesos, hoy en día es la más utilizada.
- Está muy enfocada en la calidad del software pues se le realizan a los productos una serie de pruebas entre las que se destacan pruebas de caja negra y blanca que están encaminada a la búsqueda de un software más preparado, más fuerte a prueba de fallos.
- El ser iterativo e incremental le permite al equipo de desarrollo en un tiempo estimado obtener una pequeña parte del producto de acuerdo a una fase que pase por todas las disciplinas, esto se conoce por iteración, permite además según cada iteración un crecimiento del producto, conocido como incremento.

- Muy organizativo ya que en cada fase se documenta todo RUP en ese sentido es muy cuidadoso ya que va siguiendo una serie de paso para que se entienda bien el sistema como tal, por medio de glosarios, modelos, diagramas, especificaciones y otros.
- Si bien RUP corresponde a una metodología de trabajo intensiva en recursos, su aproximación al problema no sólo garantiza que los proyectos abordados serán ejecutados íntegramente sino que además evita desviaciones importantes respecto de los plazos y también permite una definición acertada del sistema en un inicio para hacer innecesarias las reconstrucciones parciales posteriores.
- La metodología RUP es más apropiada para empresas grandes y proyectos del mismo tamaño ya que se tienen que manejar procesos complejos y un desarrollo iterativo, en proyectos pequeños es posible que no sea posible cubrir los costos de dedicación del equipo de profesionales, ya que tendrían que invertir una gran cantidad de horas y esfuerzo para hacer las cosas según lo que dicen los procesos.

2.31 Desventajas del RUP

- La principal desventaja que tiene RUP es que son una serie de procesos complicados y que todo desarrollador que no haya trabajado con RUP le va a ser difícil comprender.
- RUP es una metodología muy pesada en cuanto a equipo de proyectos puesto que se necesita una gran cantidad de personas capacitadas para desarrollar cada uno de los roles aunque algunos roles se pueden desempeñar por una misma persona implicaría invertir una gran cantidad de horas de esfuerzo.
- Como se dijo anteriormente RUP tiene mucha documentación, hasta lo más mínimo es documentado y esto para algunos proyectos (corto plazo) que requieren de rapidez de entrega constituye un freno al desarrollo del software por la burocracia del RUP. Además como se conoce que por cada fase que pasa RUP, se generan artefactos que es otra carga más para el desarrollador y algunos artefactos son requeridos por otras fases así que los mismos no se pueden obviar.

Conclusiones

En este capítulo se ha estudiado con detalle las diferentes metodologías anteriormente citadas analizando sus características, ventajas y desventajas, que elementos tienen unas que no tienen las otras así como para que equipo de trabajo está pensada cada una. Se ha llegado a una primera conclusión importante del trabajo y es desechar la opción de proponer la metodología ágil MSF pues es software propietario y la Universidad de las Ciencias Informáticas esta luchando para emigrar hacia software libre.

Por otra parte se ha visto que RUP es una metodología robusta, intensiva en recursos y en equipo de trabajo, muy organizativa y enfocada en la calidad a diferencia de XP.

Extreme Programming es casi todo lo contrario sólo que tiene el mismo propósito de RUP y es lograr conformar un software a partir de los requerimientos del cliente.

Luego de realizado un estudio de las metodologías MSF, RUP y XP entonces se podrá emitir una propuesta en el capítulo 3 y acorde a las características propias y necesidades de los proyectos de la Facultad 1.

CAPÍTULO 3. SOLUCIÓN PROPUESTA.

Introducción.

Después de haber estudiado las metodologías que se han visto en los capítulos anteriores se da paso a la solución propuesta del trabajo de diploma luego de crear un método que permita clasificar los proyectos de acuerdo a su tamaño. También se validará el trabajo por el método de expertos.

3.1 Validación del procedimiento. Método de Expertos

Para la validación y aceptación del la propuesta del proceso del trabajo se utilizó el Método de Expertos, consiste en la evaluación cuantitativa de criterios que permite determinar si la propuesta analizada es aceptada o no.

Este grupo de expertos se conformó con especialistas que poseen un vasto conocimiento de los temas relacionados con las metodologías de desarrollo como son líderes de algunos proyectos. La correcta elección de los expertos propicia obtener resultados con calidad y una opinión grupal con un alto grado de consenso.

En este capítulo se hará la descripción de los pasos utilizados para la selección de los Expertos y los resultados obtenidos de los mismos.

3.1.1 Proceso de selección de los expertos

Un experto debe ser una persona experimentada en el tema que se estudia la cual posee una gran experiencia o habilidad en la misma, capaz de ofrecer valoraciones conclusivas de un problema en cuestión y hacer recomendaciones al respecto.

Esta selección se realizó atendiendo a la posibilidades reales de participación de los candidatos, pues este grupo de expertos que tienen experiencia en el proceso productivo de la Facultad 1. Poseen además, amplios conocimientos en temas relacionados con el proceso a evaluar, estos son:

1. Proceso de Desarrollo de Software

2. Metodologías de desarrollo

3. Gestión de Proyectos

4. Planificación de proyectos

El conocimiento sobre estos temas ha permitido que las opiniones brindadas sean confiables y válidas para el objetivo propuesto

3.1.2 Cantidad de Expertos seleccionados

No existe una norma generalizada que determine un número óptimo respecto al número de expertos. Los investigadores de *Rand Corporation*, [Landeta, 1999] indican que es necesario como mínimo de siete expertos y un máximo de 30.

En este trabajo se decidió contar con un número de 7 expertos, teniendo en cuenta nivel de complejidad y profundidad del contenido.

3.1.3 Guía para la validación de la propuesta

Para llevar a cabo el desarrollo de la validación se efectuaron un conjunto de pasos, los cuales se detallan a continuación:

1) Elaboración de los criterios de evaluación que fueron utilizado en el desarrollo de la validación y se agruparon por categoría.

Grupo No 1: Criterios de mérito científico

1. Valor científico de la propuesta.

2. Calidad de la investigación.

3. Contribución científica de la propuesta.

4. Responsabilidad científica del estudiante.

Grupo No 2: Criterios de implantación

5. Necesidad de empleo de la propuesta.

6. Posibilidades de aplicación.

Grupo No 3: Criterios de flexibilidad

7. Adaptabilidad a proyectos productivos de la facultad 1.

8. Uso de los recursos necesarios para la aplicación de estas metodologías.

Grupo No 4. Criterios de impacto

9. Impacto en el área para la cual está destinada la propuesta.

10. Organización en el proceso de desarrollo.

2) *Determinar el peso relativo de cada grupo de criterios de acuerdo al por ciento que representa cada grupo del total y los intereses a evaluar.*

Grupo No.1…… 40

Grupo No.2…… 20

Grupo No.3…… 20

Grupo No.4…… 20

3) *Solicitud a los expertos seleccionados la evaluación de cada uno de los criterios en una escala del 1 al 10, teniendo en cuenta que la suma del valor dado por parte de los*

expertos a cada criterio de un grupo no exceda del peso relativo asignado a este. Para recoger la información anterior se definieron dos modelos. Ver anexos 1 y 2.

4) *Construcción de una tabla que guarda el resultado del trabajo de los expertos. Ver tabla 3.3.*

G	C/E	E1	E2	E3	E4	E5	E6	E7	Ep
40	**C1**	11	12	10	11	10	10	12	10,85714
	C2	10	9	10	10	11	11	10	10,14286
	C3	9	9	10	10	9	9	9	9,285714
	C4	10	10	10	9	10	10	9	9,714286
20	**C5**	10	11	9	10	10	9	10	9,857143
	C6	10	9	11	10	10	11	10	10,14286
20	**C7**	12	12	11	10	10	11	11	11
	C8	8	8	9	10	10	9	9	9
20	**C9**	12	13	12	10	13	14	10	12
	C10	8	7	8	10	7	6	10	8
T		100	100	100	100	100	100	100	100

Tabla 3.3 Resultado del trabajo de expertos.

5) *Verificación de la consistencia en el trabajo de los expertos, para lo que se utiliza el coeficiente de concordancia de Kendall y el estadígrafo Chi cuadrado (X2).*

Para esto se sigue el procedimiento siguiente:

- Sea C el número de criterios que van a evaluarse y (E) el número de expertos que realizan la evaluación.
- Para cada criterio se determina:
    - ∑E: Sumatoria del peso dado por cada experto.
    - Ep: Puntuación promedio del peso dado por cada experto.

- M∑E: media de los ∑E.
- ΔC: Diferencia entre ∑E y M∑E.

- Se determina la desviación de la media, que posteriormente se eleva al cuadrado para obtener la dispersión (S) por la expresión:

$$S = \sum (\sum E - \sum \sum E / C)^2$$

- Conociendo la dispersión se puede calcular el coeficiente de concordancia de Kendall (W):

$$W = S / E^2 (C^3 – C)/ 12$$

- El coeficiente de concordancia de Kendall permite calcular el Chi cuadrado real:

$$X^2 = E (C-1) W$$

- Los valores obtenidos se muestran en la siguiente tabla.

Expertos/Criterios	**E_1**	**E_2**	**E_3**	**E_4**	**E_5**	**E_6**	**E_7**	**∑E**	**Ep**	**ΔC**	**ΔC^2**
C_1	11	12	10	11	10	10	12	76	10,85714	6	36
C_2	10	9	10	10	11	11	10	71	10,14286	1	1
C_3	9	9	10	10	9	9	9	65	9,285714	-5	25
C_4	10	10	10	9	10	10	9	68	9,714286	-2	4
C_5	10	11	9	10	10	9	10	69	9,857143	-1	1
C_6	10	9	11	10	10	11	10	71	10,14286	1	1
C_7	12	12	11	10	10	11	11	77	11	7	49
C_8	8	8	9	10	10	9	9	63	9	-7	49
C_9	12	13	12	10	13	14	10	84	12	14	196
C_{10}	8	7	8	10	7	6	10	56	8	-14	196
DC	100	100	100	100	100	100	100	700	100	27	558
M ∑E	70										

S	398
W	0.098
X^2	6.20
$\mathbf{X^2}_{(\alpha, c-1)}$	21.6660

Tabla 3.4 Tabla para el cálculo de concordancia de Kendall

- El Chi cuadrado calculado se compara con el obtenido del las tablas estadísticas.
- Si se cumple:

$$X^2_{real} < X^2_{(\alpha, c-1)}$$

$$6.20 < 21.6660$$

Lo que demuestra que existe concordancia en el trabajo de expertos.

6) *Si no existe concordancia se hace necesario repetir el trabajo de expertos.*
7) *Después de comprobar la consistencia del trabajo de expertos se puede definir el peso relativo de cada criterio (P).*
8) *Conociendo el peso de cada criterio y la calificación dada por los evaluadores en una escala de 1-5 se puede construir la Tabla 5 calificación de cada criterio, para obtener el valor de de P × c., donde (c), es el criterio promedio concebido por los expertos.*

Criterios	**Calificación (c)**					**P**	**P × c**
	1	**2**	**3**	**4**	**5**		
$\mathbf{C_1}$					X	0.108	0.54
$\mathbf{C_2}$				X		0.101	0.404
$\mathbf{C_3}$				X		0.0928	0.3712
$\mathbf{C_4}$					X	0.0971	0.4855
$\mathbf{C_5}$				X		0.0985	0.394

C_6				X		0.101	0.404
C_7				X		0.11	0.44
C_8					X	0.09	0.45
C_9			X			0.12	0.36
C_{10}				X		0.08	0.32

Tabla 3.5 Tabla de calificación de cada criterio

9) Se calcula el Índice de aceptación del proyecto (IA).

$IA = \sum (P \times c) / 5$

$IA = 0.833$

10) Por último se determina la probabilidad de éxito de la propuesta

3.1.4 Rangos predefinidos de Índice de Aceptación

IA > 0,7 Existe alta probabilidad de éxito

0,7 > IA > 0,5 Existe probabilidad media de éxito

0,5 > IA > 0,3 Probabilidad de éxito baja

0,3 > IA Fracaso seguro

Por lo que se puede concluir que la propuesta tiene una alta probabilidad de éxito.

3.2 ¿Cómo determinar si un determinado proyecto es pequeño, mediano o grande en la Facultad 1?

Teniendo en cuenta, por lo anteriormente visto, que existen grandes posibilidades de éxito del empleo de la propuesta y ya una vez validado el trabajo por el método de expertos. Se pasa a

un punto muy importante del presente trabajo es el hecho de saber cuando se esta en presencia de un determinado proyecto (pequeño, mediano o grande) para luego saber entonces que metodología pueda ser más óptima a utilizar en el mismo. Surge la siguiente interrogante entonces, ¿de qué forma se puede determinar dicho tamaño?

Todo software esta limitado por sus requerimientos o requisitos ya que los mismos son los que definen el tamaño del proyecto. Por citar un ejemplo: si un cliente hace un contrato con la Facultad 1 y dicho pedido tiene una enorme cantidad de requisitos, requerirá entonces de un gran equipo de desarrollo convirtiéndolo así en un proyecto grande. El objetivo del presente epígrafe es darle respuesta a esto; para ello, se tratará de establecer un rango de requisitos que permita evaluar que tipo de proyecto es de acuerdo a ese rango. Para ello se ha realizado un estudio recorriendo la mayoría de los proyectos de la Facultad 1 en busca de esta valiosa información. Los resultados se enuncian a continuación.

Antes de comenzar a analizar los resultados es necesario aclarar este rango que se va a establecer constituye un cálculo aproximado el mismo va a dar una idea a desarrolladores y líderes de proyecto de cuan análogo es su proyecto a dicho rango, puede darse el caso que un proyecto este dentro de un rango y se aproxime mucho a otro entonces a consideración de su líder puede aproximarlo al rango superior o inferior, es decir por ejemplo si se esta en presencia de un proyecto pequeño según el rango de requisitos y quede lo suficientemente cerca (y de acuerdo con la opinión de su líder de proyecto) del rango de proyecto mediano se puede considerar por aproximación un proyecto mediano.

Para el siguiente estudio se entrevistaron a los líderes de proyecto de Calidad, Kainos, SICI, Identidad, Akademos, Seguridad y Protección, Servicios Comunitarios, Intranet de los cuales se obtuvo los siguientes resultados:

Proyectos	Tamaño del Proyecto	Requisitos Funcionales
Identidad	Grande	Más de 250
Kainos	Grande	220
Akademos	Grande	180
SICI	Mediano	130
Servicios Comunitarios	Mediano	125
Seguridad y Protección	Pequeño	50
Intranet	Pequeño	16

Tabla 3.6 de Requisitos Funcionales.

De acuerdo a los resultados obtenidos se ve claramente que existe un rango entre cada tipo de proyecto, los proyectos pequeños no pasan de 100 requisitos los medianos no llegan a 200 y los grandes sobrepasan los 200 requisitos funcionales, aunque Akademos tiene 180 requisitos está formado por 11 módulos y cuenta con un gran equipo de desarrollo se considera por aproximación que es un proyecto grande, el rango que se ha establecido se visualiza a continuación.

Figura 19. Rango de Aproximación de Requisitos Funcionales.

3.3 Propuesta de la metodología ágil XP para proyectos pequeños de la Facultad 1.

Luego que se sepa a que tipo de proyecto se tiene y después de haber estudiado y analizado las diferentes metodologías XP, RUP y MSF se propone a utilizar en proyectos a corto plazo la metodología ágil Extreme Programming (XP) para los proyectos de la Universidad de las Ciencias Informáticas.

3.3.1 ¿Por qué utilizar XP en proyectos pequeños en la Facultad 1?

Ninguna metodología es perfecta y se pretende decir cual se adapta mejor a cada caso, para proyectos pequeños, XP viene como anillo al dedo ya que siempre estos proyectos son una carrera contra reloj, que huyen de la extensa documentación y carga de trabajo pues XP se centra solo en la funcionalidad y no en la calidad, ni el diseño, ni otras desviaciones. Como problema fundamental en esta metodología se encuentra que está diseñada prácticamente para un equipo de trabajadores expertos, en la Facultad 1, la fuerza productiva está principalmente concentrada en estudiantes de distintos años y de profesores jóvenes, en la mayoría de los casos recién graduados, por lo que el término de expertos va a jugar un papel nulo prácticamente.

Como el cliente es parte del equipo de desarrollo no sería nada ventajoso para un polo productivo que fuera este el que estimara los costos del negocio a implementar como plantea el proceso de desarrollo de la metodología. En este caso sería de mayor seguridad un especialista de la Facultad 1 trabajando junto al cliente. A la hora de señalar los roles de los equipos no sólo en XP también en RUP y MSF se llega a otro problema y es que la Facultad 1 cuenta con estudiantes que aún se le imparten clases y en un determinado proyecto no pueda jugar el rol que se le asignen.

Esta fuerza inexperta se le pudiera dar una posible solución como año cursado luego un estudiante de tercer año pudiera jugar un rol de Bases de Datos que se cursa en segundo y a la vez un individuo que cursa segundo podrá ejercer ese rol también pero no podrá; por citar otro ejemplo, desempeñar el rol de un analista de sistema. Se podría citar como una de sus

principales ventajas la comunicación constante entre el cliente y los restantes miembros del proyecto, esto para la Facultad 1 mejoraría las relaciones interpersonales y consigo se lograría una mejor cohesión del grupo de trabajo en conjunto con los clientes.

Para cualquier grupo de desarrolladores es elemental la agilidad y el culminar rápidamente y de forma organizada un sistema; si se toma como referencia lo planteado por XP en cuanto a la planificación de iteraciones cortas e integraciones constantes del software podría ser adquirido como una ventaja práctica a aplicar en la Facultad 1. Esto traería consigo no solo una mejoría para el proyecto, influiría también en la satisfacción del cliente porque en cada release o entrega se obtendría una versión del sistema.

Esto trae consigo las implantaciones de pruebas para dar una idea de cuan adelantado esta el proyecto y demostraría la corrección del código. Uno de los principios de XP es la programación en pareja y si se aplicara a la facultad 1 se obtendría mejoras inimaginables pues la misma vincula a los dos programadores y estrecha su vinculación con lo que hacen, conversan, se ponen de acuerdo de lo que van hacer teniendo en cuenta que lo que se puede ocurrir a uno quizás al otro no le guste y se ocurra alguna otra vía de solución más óptima.

Siempre teniendo en cuenta que dos cabezas piensan mejor que una, se logrará una mejora a la hora de implementar y se agilizará más en la corrección de errores, además se aprovecharían mejor los recursos tecnológicos.

XP es una metodología muy fácil de aprender lo que posibilita que el estudiantado y trabajadores se le puedan dar cursos de Introducción a la metodología XP. Esta fue creada para eliminar las actividades improductivas además esta pensado para equipos pequeños pues el mismo define pocos roles. Su creador Kent Beck dejó citado que lo recomendable para usar XP sería en un equipo de 3 a 20 personas como máximo, otros opinan que hasta 10 cada equipo como máximo.

3.3.2 ¿Por qué no utilizar RUP y MSF en proyectos pequeños en la Facultad 1?

Teniendo en cuenta las comparaciones y el estudio de las metodologías en el segundo capítulo, se puede alegar que RUP representa un proceso muy abultado para desarrollarlos en proyectos a corto plazo debido a la burocracia o documentación que generan cada actividad de cada fase y esto representa una pérdida de tiempo para los desarrolladores. Además RUP carga con demasiada tareas a los mismos cosa que evidentemente no puede ocurrir, tiene definidos 31 roles y genera más de 100 artefactos distintos lo cual requiere de tiempo y coste.

MSF es otra metodología que puede ser utilizada para proyectos pequeños pues es flexible y escalable puede adaptarse para equipos de 3 a 4 personas hasta 50 o más quiere decir que de hecho MSF puede utilizarse para cualquier tamaño de proyecto pero cuando se compara con XP no es la más recomendada y mundialmente se esta utilizando XP para estos tipos de proyectos puesto que la mayoría de los clientes solo quieren el producto lo antes posible y que haga lo que esta establecido en el contrato o sea que esta encaminada a las funcionalidades. Además MSF utiliza herramientas de Microsoft por lo que va en contra de las reglas de la universidad ya que se pretende migrar hacia software libre.

3.4 Propuesta de la metodología RUP para proyectos medianos y grandes de la Facultad 1.

En primer lugar, el entorno en el cual se desarrolla el problema es la Universidad de las Ciencias Informáticas, en la cual en tercer año se imparte una de las asignaturas más importantes para un ingeniero informático y es “Ingeniería de Software”, la misma se encarga de enseñar a los estudiantes una metodología robusta y exigente como RUP. Por ese motivo resulta conveniente utilizarla en los proyectos de la UCI, ya que desde al menos segundo año el estudiante puede vincularse a un proyecto de producción.

Los proyectos medianos y grandes requieren ya de una carga de trabajo mayor para los equipos de proyectos y si no se tiene una metodología de plan como RUP, la cual es muy

organizativa y rigurosa pues asigna tareas y responsabilidades a cada rol que labora con el fin de obtener un software sólido y de calidad. Para cada equipo se le asignan metas a cumplir con fechas de entrega, cada actividad es documentada llevando el control del software y de sus artefactos.

RUP es aplicada en disímiles proyectos en la UCI dando resultados satisfactorios y eso es debido a que los procesos de RUP son flexibles a cualquier tipo de proyecto. Si se está en presencia de un proyecto grande o mediano; se hace necesario tener una buena arquitectura, para poder sostener ese software pues lo que no puede ocurrir es que a un proyecto determinado se le aplique una determinada metodología que al final fracase porque la metodología no tenía una arquitectura sólida ni la calidad requerida para llevar a cabo el proyecto.

RUP otorga al rol de arquitecto una vital importancia dentro un proyecto ya que la misma esta centrada en la arquitectura el arquitecto es uno de los principales trabajadores en la toma de decisiones también es el encargado de la integración y de la lógica de las principales funcionalidades del sistema y además esta enfocada a la calidad por medio de una serie de pruebas como las de caja blanca y caja negra. Brindándole así al software una gran robustez para al final cumplir con las expectativas del software.

3.4.1 ¿Por qué no utilizar XP y MSF en proyectos medianos y grandes?

El primer problema que salta a la vista es que son metodologías ágiles y que no se imparten en la Universidad, por lo que no se tienen conocimientos básicos de estas metodologías. Se tendría que pensar cómo preparar al personal por medio de cursos u otros métodos de aprendizajes para luego pensar en la utilización de dichas metodologías en un determinado proyecto.

XP es una metodología pensada para proyectos pequeños, su rapidez y agilidad de nada servirá frente a una situación más grande; los cuales son característicos de los proyectos no pequeños, su trabajo se verá limitado por la falta de organización del personal como se

explicaba en los demás capítulos, algo que tiene XP a su favor y se debería usar en las demás metodologías es su Programación en Parejas que siempre es muy provechoso. XP define pocos roles, los cuales no se podrá distribuir entre equipos mas grandes, es decir, quedarían roles por definir lo que costaría tiempo y retrasaría al proyecto.

Otro problema de XP es su diseño simple; en estos tipos de proyectos un diseño simple no cumpliría con las funcionalidades de un sistema mediano o grande, el hecho de entregar versiones a los clientes que pueden ser defectuosos constituye otro dilema para esta metodología. Un software de tales magnitudes no debe tener errores. Aún si el software es aprobado por el cliente quizás otro no este conforme con tales defectos y no sólo en la UCI, sino en el mundo entero, un software obtenido en tiempo récord por una metodología ágil, el cliente (que se recuerda que es parte del equipo de desarrollo) pensará que el equipo está malgastando el tiempo (pues una metodología ágil como XP optimiza el tiempo de entrega imprimiéndole agilidad al desarrollo del software) sin trabajar duro y sólo le han hecho un sistema barato lo cual no debe ocurrir en nuestra universidad .

Si el cliente necesita un software grande o mediano es porque no es un sistema sencillo y se debe hacer con calidad, a prueba de errores, a tiempo para la entrega que es muy importante dejándole ver al cliente que su equipo trabaja eficientemente durante la jornada laboral. No se esta diciendo que XP es una mala metodología pues XP es una de las metodologías más usadas en el mundo y es innovadora sino que no se ajusta a tales proyectos.

MSF no es recomendable usarla debido que aunque es una metodología ágil, está conformada por una serie de procesos y modelos muy cargados. Pues su principal desventaja es que se torna un trabajo bastante largo, ya que para cada fase se debe documentar profundamente todo lo que se haga, la misma, utiliza herramientas de Microsoft lo cual implicaría ir en contra de las políticas de la Universidad de las Ciencias Informáticas.

MSF es flexible para cualquier tipo de proyecto y para cualquier metodología y además es análoga a RUP en cuanto a la forma en que trabajan, es decir tienen fases aunque claro está,

son distintas, son iterativas e incremental, requieren de casos de usos , de levantamientos de requisitos funcionales y no funcionales , precisa de los requerimientos del sistema, genera artefactos casi idénticos a RUP como documento visión o sea la forma de trabajar es muy parecida y claro, está de más decir que es un proceso cargado de modelos , roles , disciplinas, planes entre otros.

Para la Universidad MSF jugará entonces un papel nulo pues es difícil de aprender al igual que RUP aunque este ya tiene mucha ventaja sobre MSF porque se imparte como asignatura y los estudiantes serán capaces al terminar tercer año de jugar de forma profesional un rol dentro de un proyecto productivo. Sin contar que ya se tiene conocimientos y experiencias de las herramientas que RUP utiliza como los lenguajes de modelados, bases de datos, servidores. El desarrollo de la metodología RUP ha traído para la UCI resultados más que satisfactorios.

3.5 ¿Qué condiciones debe tener un proyecto para usar XP en la Facultad 1?

- El cliente no sabe que es lo que quiere o bien se imagina mal el negocio que él mismo define.
- El cliente deberá estar a tiempo completo en la institución, no es recomendable que como otros proyectos de la facultad como SICI las entrevistas con el cliente son difíciles de concertar por el cúmulo de trabajo que existe en esa Dirección de la Universidad.
- Aunque el cliente forme parte del equipo de desarrollo no es recomendable un cliente que tome las decisiones más importantes, para ello se deberá utilizar una persona de experiencia para que lo asesore y lo aconseje, por lo que se debe tener personal calificado y con experiencia con gran conocimiento del negocio.
- Se necesitará programadores bien experimentados para la programación en pareja que propone XP.
- Necesidad de un equipo pequeño, mientras más pequeño mejor .Cómo máximo veinte personas según Kent Beck.

- El equipo deberá tener motivación de utilizar una metodología ágil, lo anteriormente planteado es muy importante, no importa si se tienen condiciones ideales para el desarrollo de XP si el equipo no se identifica con la metodología XP el trabajo no tendrá éxito.

3.6 ¿Qué condiciones debe tener un proyecto para usar RUP en la Facultad 1?

- No será necesario la presencia constante del cliente en la institución, el mismo deberá tener, conocer y definir bien el negocio durante el levantamiento de requisitos. Pues RUP es una metodología rígida, compuesta por una serie de fases y procesos complejos que un cambio en la etapa final significaría un retraso para el proyecto ya que RUP cuenta con iteraciones muy largas que conllevaría a volver a modificar todo el trabajo que se había hecho.
- Necesidad de profesionales con conocimiento de la metodología RUP. Este factor es importante debido que RUP es una metodología difícil de aprender a nivel mundial, sin embargo en la Universidad de las Ciencias Informáticas se imparte RUP como asignatura en tercer año: “Ingeniería de Software”. Posibilitando así la vinculación de los estudiantes en proyectos productivos las cuales utilizan y aplican esta metodología.
- Se requiere (Véase Capítulo 2 (epígrafe 2.5.5) de la guía completa del Proceso Unificado.) de herramientas de programación las cuales proporcionan una gama de herramientas como editores, compiladores, detectores de errores, depuradores y otros además para dar soporte al ciclo de vida completo de un software. En la Facultad 1 los proyectos son implementados en el lenguaje de programación java y otros proyectos utilizan PHP, lo mismo pasa con los gestores de bases de datos la mayoría de los proyectos utilizan PosgreSQL aunque otros utilizan MySQL.Las herramientas de modelado visual permiten la visualización y elaboración de sus diagramas, modelos y documentos como el Lenguaje Unificado de Modelado (UML) la cual permite ensamblar y visualizar una aplicación visualmente, aunque algunos proyectos de la Facultad 1 utilizan Rational la mayoría utiliza Visual Paradigm.
- Necesidad de un equipo de proyecto mediano o grande de más de treinta personas. Debido que es la cantidad ideal de miembros del equipo que propone RUP de acuerdo

a los roles que este define. No es recomendable asignarle más de un rol a una persona aunque esto debe formar parte de la decisión del líder de proyecto.

3.7 Aplicación de la metodología ágil XP en la Facultad 1

Desde que surgieron los proyectos productivos en la Facultad 1, no se había utilizado otro tipo de metodología que no fuera RUP. Hasta que un equipo de desarrollo debutó en un proyecto pequeño usando un híbrido que unía la metodología XP con SCRUM buscando las mejores opciones que brindan las mismas. SCRUM se utilizó fundamentalmente en la parte de planificación del proyecto para lograr una buena organización del equipo de desarrollo; debido a que XP es un tanto “liberal” para organizar el trabajo.

XP fue utilizado desde las historias de usuarios hasta las tempranas entregas al cliente obteniéndose resultados satisfactorios para la Facultad 1 y la UCI en general. Con un equipo de sólo 17 personas y el uso de la metodología ágil XP se han logrado resultados como los que se mencionan a continuación:

Se construyó el portal de la Facultad 1 nombrado “Intranet Facultad 1” el cual brinda informaciones referentes a noticias, avisos, fechas y cronogramas de pruebas de nivel y mundiales entre otros servicios.

Se obtuvo la comunidad de diseño web en la UCI llamado “Web 21” la misma está encaminada a mejorar el diseño y calidad de los sitios, es un sitio muy bueno para diseñadores y otros interesados en el tema.

También se construyó el sitio “Expediente Ambiente de Control Interno” destinado para gestionar información referente a una determinada entidad. El Control Interno, es el proceso integrado a las operaciones efectuadas por la dirección y el resto del personal de una entidad para proporcionar una seguridad razonable al logro de los aspectos siguientes:

Confiabilidad de la información.

Eficiencia y eficacia de las operaciones.

Cumplimiento de las leyes, reglamentos y políticas, establecidas.

Control de los recursos, de todo tipo, a disposición de la entidad.

3.8 Aplicación de la metodología RUP en la Facultad 1

La metodología RUP se aplica hoy día en la mayoría de los proyectos de la Facultad 1 los cuales han obtenido resultados satisfactorios, por solo citar algunos ejemplos se encuentran los proyectos Kainos que trabaja con un equipo de desarrollo grande. Además SICI otro de los proyectos de la facultad en estos momentos se encuentra terminando una primera versión del Sistema de Cooperación Internacional lo cual fue gracias entre otros factores al correcto uso de la metodología RUP.

Akademos, ha trabajado de forma iterativa e incremental dando cumplimiento a uno de las principales características del RUP, lo cual permite realizar una nueva versión del producto sin descuidar la que se encuentra en uso en estos momentos.

Conclusiones:

Se espera que se tenga en cuenta este estudio realizado para futuras mejoras en el proceso productivo de la Facultad 1 tomando en cuenta las aplicaciones que ya han surgido utilizando metodologías ágiles. De las opiniones recogidas durante la búsqueda de informaciones y los resultados obtenidos de la validación por métodos de expertos en los proyectos se demuestra que existe una tendencia a inclinarse por esta nueva metodología (XP) y que existen posibilidades de su aplicación en la Facultad 1.

Con este estudio realizado se podrán establecer métricas que permitan saber a qué tipo de proyecto se esta enfrentando y la metodología de desarrollo más óptima a utilizar en los proyectos productivos de la Facultad 1.

CONCLUSIONES

En el presente trabajo se ha realizado un estudio del estado del arte de las metodologías para el desarrollo de software en el mundo así como una análisis de las características principales de las mismas, modo en que estas operan, roles que definen, sus ventajas y desventajas con el fin de ver cuál metodología es más óptima para ser aplicada a los diferentes proyectos de la Facultad 1.

El uso de la propuesta ágil para proyectos pequeños aumentará sin dudas la agilidad y calidad permitiendo entregar los productos en fecha sin perder tiempo en usar metodologías que no estén acordes al tipo de proyecto que se desea implementar pues la relación costo-calidad-tiempo se verá favorecida si se usa una metodología correcta. Se ha establecido un método teniendo en cuenta las características propias del proceso productivo de la Facultad 1 que permitirá de acuerdo a la complejidad funcional de cada proyecto, determinar un rango o escala para clasificarlos en grandes medianos y pequeños. Por último para dar cumplimiento a los objetivos trazados se ha lanzado la propuesta de utilizar la metodología más óptima para cada tipo de proyecto como se dijo en el capítulo 3 las cuales son:

- Proponer la metodología ágil XP para proyectos pequeños de la Facultad 1.
- Mantener el uso de la metodología RUP en los proyectos medianos y grandes en la Facultad 1.

RECOMENDACIONES

Para aquellas personas interesadas sobre el tema de metodologías de desarrollo este trabajo brinda las características a seguir para llevar a cabo una metodología correcta, además cuenta con un estudio general sobre tres metodologías cotizadas mundialmente que servirían como soporte documental para el desarrollo de otros trabajos. Para las nuevas generaciones de desarrolladores ágiles que van surgiendo no solo en la Facultad 1 este trabajo puede brindarle una gran ayuda, así como también los líderes de proyecto que no tienen definido aún la metodología que va a usar puede apoyarse en el mismo .

Se recomienda que se impartan cursos optativos de la metodología XP para el estudiantado y trabajadores de la UCI con el fin de prepararlos hacia una metodología innovadora y veloz para proyectos pequeños.

Se incita a estudiar con profundidad las nuevas metodologías de desarrollo que vayan surgiendo así como también la publicación de manuales de estas metodologías para su posible estudio ya que no se cuenta con tales documentaciones.

Se propone a la Facultad 1 y a la Universidad en general que se tome en consideración la variante de impartir más de una metodología en la asignatura “Ingeniería de Software”, se recomienda que se estudie también la metodología ágil XP y otras metodologías novedosas.

REFERENCIAS BIBLIOGRÁFICAS

Ágil –Spain "Principios del Manifiesto Ágil " from http://www.agile-spain.com/agilev2/book/print/357

Colombia, M. c. (2000). "Microsoft Solutions Framework (MSF): Disciplinas y buenas prácticas para el desarrollo e implantación de proyectos.

GUIDANCE, P. (2005). "MSF for agile software development ", 2007, from http://www.microsoft.com/downloads/details.aspx?familyid=9F3EA426-C2B2-4264-BA0F-35A021D85234&displaylang=en.

Ivan Jacobson, Grady booch,James Rumbaugh El Proceso Unificado de Desarrollo de software guia completa de hecha por sus creadores http://bibliodoc.uci.cu/pdf/reg00060.pdf

“Métodologías Ágiles” en el Desarrollo de Software por José H. Canós, Patricio Letelier y Mª Carmen Penadés hace referencia autor Kent Beck en cuanto a tamaño de equipo en las conclusiones disponible en http://www.willydev.net/descargas/prev/TodoAgil.Pdf

“Método Delphi” Rand Corporation, [Landeta, 1999] http://www.codesyntax.com/prospectiva/Metodo_delphi.pdf

Microsoft. (2007). "Metodologías Ágiles en el Desarrollo de Software y la propuesta de Microsoft -Ser o no ser ágil, he ahí el dilema." 2007, from http://www.microsoft.com/colombia/empresas/clientepreferencial/actualizacion-gerencial/febrero/innovaciones2.mspx. Miguélez, M. M. (1999). "Criterios para la Superación”

Molpeceres, A. (2003). "Procesos de desarrollo:RUP, XP y FDD." Retrieved 11/11, 2006, from http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF.

Sanchez, M. A. M. (2004). "Metodologías De Desarrollo De Software." Retrieved 12, 2006, de http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf

Sam Guckenheimer, J. J. P. (2006). Software Engineering with Microsoft Visual Studio Team System, Addison Wesley Professional.

Spaces, W. L. (2005). "Qué hay para decir de MSF (Microsoft Solution Framework)." Retrieved 11, 2006, de http://diegumzone.spaces.live.com/Blog/cns!1pHxrrKG6RzuZjEIZgyJyg0A!119.entry.

Rational Software Corporation. 2003. Rational Unified Process. [Online] Rational Software Corporation, 2003. [Cited: Marzo 1, 2007.] \Rational\RationalUnifiedProcess\index.htm.

The Mythical Man-Month Por Frederick P. Brooks se hace referencia a este libro disponible en http://64.233.169.104/search?q=cache:QZaUioFzfJQJ:www.educagratis.cl/moodle/mod/resource/view.php%3Finpopup%3Dtrue%26id%3D299+fred+brooks+1975+en+esos+casos+a%C3%B1adir+gente+a+un+proyecto+retrasado,+s%C3%B3lo+lo+retrasa+m%C3%A1s&hl=es&ct=clnk&cd=2&gl=cu

WordReference.com. 2005**.** Diccionario de la lengua española. Madrid : Espasa-Calpe S.A., 2005.

BIBLIOGRAFÍA.

[Beck 2000] Kent Beck. Extreme Programming Explained: Embrace Change. Addison Wesley Longman, 2000. http://www.extremeprogramming.org/

Extreme Programming: A gentle introduction, What is Extreme Programming?, 17 de febrero de 2006. Disponible en: http://www.extremeprogramming.org/what.html

Extreme Programming for Web Projects by Doug Wallace

Fases de la Programación Extrema from http://programacionextrema.tripod.com/fases.htm

Ivar Jacobson, G. B., James Rumbaugh (1999). Proceso Unificado de Desarrollo de Software.

Molpeceres, A. (2003). "Procesos de desarrollo:RUP, XP y FDD." Retrieved 11/11, 2006, from http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF.

GLOSARIO DE TÉRMINOS

Casos de Uso: Es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema.

Ciclo de vida de un proyecto: Para obtención de un producto todos los proyectos llevan a cabo un proceso o servicio para lo cual se generan actividades. Las actividades se agrupan en fases debido a que generalmente están ligadas a la obtención de un producto intermedio que es necesario para facilitar la gestión del proyecto y continuar el paso a la obtención del producto final. A este conjunto de fases se les denomina ciclos. El ciclo de vida de un proyecto esta determinado por el conjunto de fases empleada en la obtención del producto final.

Herramientas CASE: CASE (Computer Aided Software Engineering). Es un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información

Ingeniería de Software: Según la definición del IEEE, citada por Lewis en 1994 "Software es la suma total de los programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo". Según el mismo autor, "un producto de software es un producto diseñado para un usuario". En este contexto, la Ingeniería de Software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software".

Software: El software es una producción inmaterial del cerebro humano. Básicamente es un plan de funcionamiento para un tipo especial de máquina virtual o abstracta. Luego de escrito mediante un lenguaje de programación, el software se hace funcionar en ordenadores, que temporalmente se convierten en esa máquina para la que el programa sirva de plan. El software permite poner en relación al ser humano y a la máquina y también a las máquinas entre sí. Sin ese conjunto de instrucciones programadas, los ordenadores serían objetos inertes, como cajas de zapatos, sin capacidad siquiera para mostrar algo en la pantalla.

Metodologías de Desarrollo: Son un conjunto de técnicas, procedimientos, herramientas y un soporte documental que utilizan los desarrolladores para la producción de un software nuevo.

Modelos de Proceso: Es un diseño efectivo, desarrollo y funcionamiento de soluciones empleado para maximizar el éxito de los proyectos de una empresa. Este conocimiento se deriva de la experiencia ganada dentro de Microsoft con sus clientes y vendedores en proyectos de grande desarrollo de software y de prestación de servicios, la experiencia de los grandes consultores de Microsoft y el mejor conocimiento de la industria mundial de Tecnologías de Información.

Proceso de ingeniería de software: Se define como "un conjunto de etapas parcialmente ordenadas con la intención de logra un objetivo, en este caso, la obtención de un producto de software de calidad" El proceso de desarrollo de software "es aquel en que las necesidades del usuario son traducidas en requerimientos de software.

Producto: La producción de un software se realiza por etapas, el resultado final de cada etapa es a lo que se le denomina producto.

Stakeholders: Se define como aquellas “partes interesadas” en el proyecto o personas que de alguna manera afectan y tienen relación con el proyecto; pueden ser patrocinadores, proveedores, la comunidad vecina, etc. Es importante su identificación, la gestión de la comunicación con éstos y visualizarlos como clientes externos o internos.

Visual Studio 2005 Team System (VSTS).

Como definición podría decirse que es el conjunto de herramientas con las cuales se puede optimizar el proceso de desarrollo de una solución, sacando ventaja de su integración a el ambiente de desarrollo Visual Studio 2005, dando así un modelo diferente para cubrir las necesidades de cada Rol en particular.

ANEXO

Plantilla para la recogida de Información.

Modelo para la recogida de información referente al peso de los criterios.

Guía para informar el peso de los criterios.
Nombre y Apellidos del evaluador…….

Le otorgará un peso a cada criterio de acuerdo a su opinión y el peso total de cada grupo debe sumar:

- Grupo No.1……. 40
- Grupo No.2……. 20
- Grupo No.3…….20
- Grupo No.4…….20

Para que el peso total asignado sea 100.

Grupo No 1: Criterios de mérito científico.

1. Valor científico de la propuesta.
   Peso.....
2. Calidad de la investigación.
   Peso.....
3. Contribución científica.
   Peso.....
4. Responsabilidad científica del estudiante.
   Peso.....

Grupo No 2: Criterios de implantación.

4. Necesidad de empleo de la propuesta.
   Peso.....

5. Posibilidades de aplicación.

Peso.....

Grupo No 3: Criterios de flexibilidad.

6. Adaptabilidad a proyectos productivos de la facultad 1.

Peso.....

7. Uso de los recursos necesarios para la aplicación de estas metodologías.

Peso.....

Grupo No 4. Criterios de impacto.

8. Impacto en el área para la cual está destinada la propuesta.

Peso.....

9. Organización en el proceso de desarrollo.

Peso.....

Plantilla para la recogida de Información.

Modelo para la recogida de información referente a la calificación de los criterios.

Nombre y Apellidos del Evaluador……

Criterios de medida que se evalúan en una escala de 1 - 5

Grupo No 1: Criterios de mérito científico.

1. Valor científico de la propuesta.

Evaluación.....

2. Calidad de la investigación.

Evaluación.....

3. Contribución científica.

Evaluación.....

4. Responsabilidad científica del estudiante.

Peso.....

Grupo No 2: Criterios de implantación.

4. Necesidad de empleo de la propuesta.

   Evaluación.....

5. Posibilidades de aplicación.

   Evaluación.....

Grupo No 3: Criterios de flexibilidad.

6. Adaptabilidad a proyectos productivos de la facultad 1.

   Evaluación.....

7. Uso de los recursos necesarios para la aplicación de estas metodologías..

   Evaluación.....

Grupo No 4. Criterios de impacto.

8. Impacto en el área para la cual está destinada la propuesta.

   Evaluación.....

9. Organización en el proceso de desarrollo.

   Evaluación.....

Categoría final.

___ Excelente: Alta novedad científica, con aplicabilidad y resultados relevantes.

___ Bueno: Novedad científica, resultados destacados.

___ Aceptable: Suficientemente bueno con reservas.

___ Cuestionable: No tiene relevancia científica y los resultados son malos.

___ Malo: No aplicable.

Valoración final

Sugerencias del experto para mejorar la calidad del proyecto

Elementos críticos que deben mejorarse.