Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

METODOLOGÍAS ÁGILES INCORPORADAS A LAS NECESIDADES DE LAS EMPRESAS QUE DESARROLLAN SOFTWARE EN GUATEMALA

Héctor Alberto Heber Mendía Arriola

Asesorado por la Inga. Claudia Rojas Morales

Guatemala, abril de 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

METODOLOGÍAS ÁGILES INCORPORADAS A LAS NECESIDADES DE LAS EMPRESAS QUE DESARROLLAN SOFTWARE EN GUATEMALA

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA

POR

HÉCTOR ALBERTO HEBER MENDÍA ARRIOLA
ASESORADO POR LA INGA. CLAUDIA ROJAS MORALES

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, ABRIL DE 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADORA	Inga. Ligia María Pimentel Castañeda
EXAMINADORA	Inga. Virginia Victoria Tala Ayerdi
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

METODOLOGÍAS ÁGILES INCORPORADAS A LAS NECESIDADES DE LAS EMPRESAS QUE DESARROLLAN SOFTWARE EN GUATEMALA,

tema que me fuera asignado por la Dirección de la Escuela de Ciencias y Sistemas, el 16 de Febrero de 2004.

Héctor Alberto Heber Mendía Arriola

DEDICATORIA

A DIOS Y A LA VIRGEN MARIA

Por haberme dado una vida llena de bendiciones y por haberme permitido llegar a este momento de mi vida.

A MI FAMILIA

A mis padres, mis hermanos y mis tíos; por haberme apoyado e instruido ya que sin ellos nunca lo hubiese conseguido.

AGRADECIMIENTOS

A DIOS Y A LA VIRGEN MARIA

Por haberme dado la fortaleza en los momentos difíciles y guiarme en el camino de la vida.

A MI FAMILIA

Por su apoyo incondicional durante toda mi vida, sus palabras de aliento y su cariño.

A MIS AMIGOS

Edgar González, Francisco Orozco, Gabriela Hernández, Juan Miguel Indekeu, Kristhian Herrera, Luis Alonzo, Marlon Tarax, Melissa García, Ricardo Giron, Ronald Alvarado y Walter Minchez por su amistad y los conocimientos compartidos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

GLOSARIO

CASE	Computer Aided *Software* Engineering. Las herramientas CASE son las que se orientan a la mejor comprensión de los modelos de empresas, sus actividades y el desarrollo de los sistemas de información.
Check-list	Lista utilizada para la verificación de actividades a realizar en un proyecto.
Consultor	Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.
Framework	Es una estructura que da soporte a un proyecto de *software* para ser organizado o desarrollado.
ISO	International Standards Organization. Organización dedicada a la estandarización de normas internacionales.
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).
Stakeholder	Es una persona que puede afectar o ser afectada por una acción, tienen interés en el éxito de la organización.

Tester	Encargado de pruebas. Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.
Tracker	Encargado de seguimiento. 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.
UML	Unified Modeling Language. Lenguaje de modelación unificado; permite modelar sistemas y sus interacciones.
Heurística	Es un algoritmo que ofrece dos objetivos fundamentales, buenos tiempos de ejecución y buenas soluciones, usualmente las óptimas.

RESUMEN

Un aspecto importante en el desarrollo de *software* es que siempre es necesario utilizar una metodología de desarrollo, la cual guíe en todo momento su creación incluso después de su implantación.

Algunos problemas que afronta el desarrollo, es que los costos temporales y económicos son frecuentemente muy imprecisos, la productividad difiere de la demanda del mercado y la baja calidad del *software* con relativa frecuencia.

Los métodos ágiles son adaptables en lugar de predictivos, son orientados a la gente y no orientados al proceso. Hay diversas prácticas inherentes en las distintas metodologías que se tratan, aunque el trabajo con iteraciones que sean lo más cortas posibles es común para todas. Todo debe de ser realizado siguiendo los estándares de desarrollo, para facilitar su lectura y modificación por cualquier miembro del equipo de desarrollo.

Los largos ciclos de desarrollo de los métodos tradicionales son incapaces de adaptarse al cambio, hay que hacer ciclos de desarrollo más cortos ya que la prioridad más alta es satisfacer al cliente mediante tempranas y continuas entregas de *software,* que le aporte un valor.

El poder responder a los cambios de requerimientos en forma robusta y en buen tiempo, logrando todo esto sin sobrecargar al equipo, para que el trabajo este hecho, representa una gran ventaja en cualquier campo.

Asegurar el progreso con *software* funcional, apuntando a la entrega del sistema que el cliente quiere al final del proyecto, no lo que busca al principio.

El uso de un método ágil no es para todos. Hay que tener en cuenta varias cosas si se decide a seguir por este camino. Sin embargo es evidente que estas nuevas metodologías son extensamente aplicables y deben ser usadas por más personas de las que actualmente lo consideran.

Lo que se desea dejar claro es que los procesos ágiles son buenos cuando sus requisitos son inciertos o volátiles. Si no se tiene requisitos estables, entonces no está en la posición de tener un plan estable y seguir un proceso planeado. En estas situaciones un proceso adaptable puede ser menos cómodo, pero será más eficaz.

OBJETIVOS

General

Mostrar cómo las nuevas metodologías de desarrollo pueden ayudar a los programadores de Guatemala a desarrollar de una mejor forma sus proyectos.

Específicos

Identificar y mostrar cada una de las distintas metodologías ágiles.

Mostrar los beneficios y las desventajas de estas metodologías, y si es algo que se debería usar: sea como desarrollador o como cliente de *software*.

Mostrar qué pasos se deben seguir en cada una de éstas para poder implementarlas en nuestro país.

Comparar las metodologías ágiles con las metodologías utilizadas actualmente en las empresas desarrolladoras de *software* en Guatemala.

INTRODUCCIÓN

Este trabajo intenta plantear y dar a conocer cuáles son las nuevas metodologías de desarrollo.

Las metodologías ágiles poseen ventaja ante la burocracia de las metodologías monumentales ya que estos nuevos métodos buscan un punto medio entre ningún proceso y demasiado proceso, proporcionando simplemente suficiente proceso para que el esfuerzo valga la pena.

Los métodos ágiles afirman que ningún proceso nunca podrá ocultar o sustituir las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo. Explícitamente, puntualizan el trabajar a favor de la naturaleza humana en lugar de en su contra, y enfatizan que el desarrollo de *software* debe ser una actividad agradable.

La parte más importante y difícil, es saber con precisión dónde estamos. Necesitamos un mecanismo honesto de retroalimentación que pueda decirnos con precisión cuál es la situación a intervalos frecuentes.

La llave para obtener esta retroalimentación es el desarrollo iterativo. Esta no es una idea nueva. El desarrollo iterativo ha estado durante algún tiempo bajo muchos nombres: incremental, evolutivo, escenificado, espiral... muchos nombres. La clave del desarrollo iterativo es producir frecuentemente versiones que funcionen, del sistema final que tengan un subconjunto de los rasgos requeridos. Esta es una de las principales características que poseen todas las metodologías.

El uso de un método ágil no es para todos. Hay que tener en cuenta varias cosas si se decide seguir este camino, esto se explica más a fondo en el último capítulo.

1. EL POR QUÉ DE LAS METODOLOGÍAS

1.1. Introducción

En este capítulo se tratan los aspectos más relevantes que respaldan la razón por la cual se debe de utilizar una metodología de desarrollo para la creación de los sistemas. Además de esto se hace mención a ciertos aspectos comunes en los que se basan las metodologías ágiles.

1.2. ¿Qué es una metodología aplicada al desarrollo de software?

Una metodología orientada al desarrollo de *software* se refiere a un conjunto de métodos que se siguen dentro del desarrollo de un *software*.

Un método se puede describir como un procedimiento para alcanzar determinado fin, en el área en la que nos encontramos éstos nos enseñan cómo construir técnicamente el *software*. Estos métodos abarcan:

- Descripción del problema y modelado del negocio: es la identificación de las necesidades del sistema y las deseadas por el usuario.
- Planificación y estimación del proyecto: se realiza una estimación del trabajo a realizar, recursos necesarios y el tiempo que tomará, y la viabilidad que el proyecto posee para nosotros.
- Análisis de requisitos del sistema y del *software*: es como una narración de la visión del sistema a lo largo de un conjunto de características funcionales que son necesidades del sistema.

- Diseño de estructuras de datos, programas y procedimientos: es una descripción de cómo se va a realizar el sistema en la fase de codificación.
- Codificación: es la producción del código fuente y la integración con los componentes externos, aquí es donde se genera un archivo ejecutable.
- Pruebas: son los elementos que garantizan la calidad del *software* y son una revisión final de las especificaciones, del diseño y de la codificación.
- Mantenimiento: es la revisión y actualización del sistema.
- Implantación: coincide con la entrega del sistema y el entrenamiento del cliente.
- Configuración de la administración: implementación del proyecto y de cada iteración.
- Administración: vela por la conclusión de cada iteración en la fecha dada.
- Ambiente: definición de los recursos de *software* y hardware para el proyecto

Al final lo que todo esto busca es mejorar la calidad del *software* y de reducir su coste, a la vez de entregar el producto en los plazos establecidos.

El *software* evoluciona a través de muchas versiones, a medida que se corrigen errores, se mejora el funcionamiento y se responde a las modificaciones que surgen en los requisitos. Cada nueva versión se crea a través de un proceso de desarrollo de *software*.

El *software* no siempre se ha desarrollado de forma controlada, y en la actualidad hay algunos sistemas que presentan grandes dificultades para su mantenimiento.

El organismo de normalización ISO (*International Standards Organization*) ha definido los requisitos de un sistema de gestión de calidad de carácter general que cubre el desarrollo de cualquier producto (ISO 9001) y ha publicado directrices específicas para aplicar esa norma al desarrollo de *software* (ISO 9000-3). Una organización que ponga en práctica un sistema de gestión de calidad según esa norma puede ser auditada y recibir una certificación formal de su proceso de desarrollo.

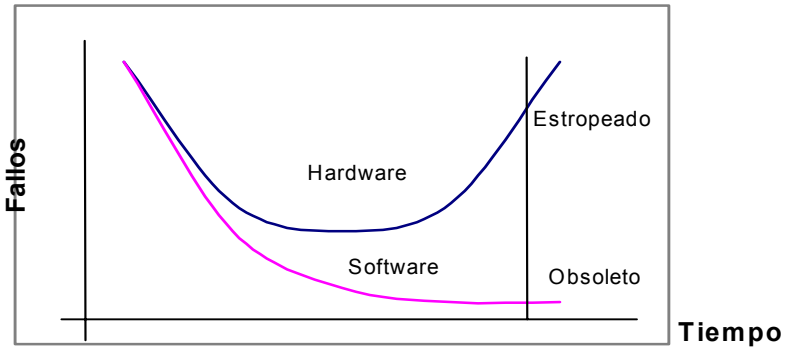
1.3. Características del Software

El proceso creativo del hardware lleva a la obtención de una forma física, el *software* es un elemento del sistema que es lógico, no es físico como el hardware. Por ello tiene unas características propias:

- El *Software* se desarrolla, no se fabrica en un sentido clásico. Para conseguir un producto de calidad se debe tener un buen diseño. Los costos del *software* se encuentran en la ingeniería, el *software* no es un producto de fabricación. Actualmente se tiende a utilizar las herramientas CASE (*Computer Aided Software Engineering*) para ayudar al desarrollo.

- El *Software* no se estropea, los agentes externos (temperatura, humedad, etc.) no le afectan.

Figura 1. Fallos del Software vrs. Hardware



En el Hardware existe una zona de mortalidad infantil hasta llegar a una zona de pocos fallos en la que se estaciona, para luego volver a empezar a fallar por envejecimiento de éste. Con el *Software* en cambio sólo existen problemas durante el desarrollo y puesta en marcha de éste como se puede observar en la figura anterior.

Lo comentado anteriormente sería lo ideal, pero normalmente se suelen efectuar cambios en el *Software* y cuantos más cambios se produzcan más se incrementarán los fallos hasta que se llegue a un punto en el que sea más rentable comenzar otro programa nuevo que adaptar el que tenemos. El coste para solucionar fallos en el *Software* es más elevado que en el Hardware.

- La mayoría del *Software* se construye a la medida, en lugar de utilizar componentes existentes. Se puede comprar *software* ya desarrollado para casos muy particulares.

Existe un gran número de personas que desarrollan *software* pero muy pocos de estos utilizan algún método para apoyar este desarrollo, esto se debe a que el desarrollo requiere de muy pocos recursos económicos caso contrario que sucede con la construcción del Hardware.

1.4. ¿Por qué utilizar una metodología para desarrollar software?

El concepto surgió en unas reuniones de trabajo organizadas por la Organización del Tratado del Atlántico Norte (OTAN) en 1968 y 1969 para estudiar lo que entonces se describía como “la crisis del *software*”. Había demasiados proyectos de desarrollo de soporte lógico que experimentaban fallos, los cuales se atribuían al rápido aumento en la escala y complejidad del *software* en cuestión.

Se reconoció que era necesario un planteamiento más sistemático en el desarrollo de *software*, que debía basarse en principios de ingeniería ya establecidos.

Las áreas de informática que no utilizan metodologías de desarrollo soportadas por herramientas CASE, podrían compararse a empresas constructoras cuyos métodos de construcción se reducen a la experiencia de sus operarios y cuyas herramientas constructivas fueran los tradicionales picos, palas, carretillas, etc. Aunque sus equipos humanos estuvieran integrados por excelentes jefes de obra y oficiales de albañilería, sus "métodos y técnicas artesanales" les impedirían abordar competitivamente cualquier proyecto de construcción actual, con independencia de que el mismo se llevase a cabo con los más modernos materiales.

Existen 3 problemas principales que afectan al desarrollo del *software*:

- La planificación y estimación de costes temporales y económicos son frecuentemente muy imprecisos.
- La productividad del *software* difiere de la demanda del mercado.

- Baja calidad del *software* con relativa frecuencia.

Para solucionar estos problemas podemos combinar métodos completos para todas las fases del desarrollo del *software*; podemos utilizar herramientas para automatizar estos métodos y mejorar la calidad del *software*, y así conseguir una disciplina para el desarrollo del *software*.

La razón por la que se debe utilizar una metodología de desarrollo se basa en los siguientes objetivos cuantitativos y cualitativos:

Cuantitativos

- Ganancia de productividad[1] de los analistas

La ganancia de la productividad de un Analista que lleva a cabo sus Análisis con la ayuda del CASE es superior al 30% y el periodo de entrenamiento y dominio de la herramienta es corto, normalmente inferior a tres meses. Esta ganancia de productividad es aun mayor cuando en un proyecto participan múltiples analistas, en estas situaciones, muy frecuentes en proyectos de tamaño medio y grande, las herramientas CASE se convierten además en excelentes herramientas de trabajo en grupo.

- Disminución de los costes de puesta en marcha de los nuevos sistemas desarrollados

Uno de los principales problemas que están teniendo la mayoría de empresas es el excesivo tiempo de la puesta a punto de los programas en los nuevos proyectos en desarrollo. Gran parte de esta problemática está directamente relacionada con un análisis y diseño inicial defectuoso e incompleto.

[1] Capacidad de producción por unidad de trabajo.

- Disminución de los costes de mantenimiento de las aplicaciones

Estos beneficios son los más importantes a largo plazo, para conseguirlos necesitamos que nuestros sistemas hallan sido llevados a cabo con el soporte del CASE ó si partimos de sistemas ya existentes, que su análisis y diseño se documente en la enciclopedia de la herramienta CASE utilizada.

Cualitativos

- Disminución de los tiempos de desarrollo y de mantenimiento de los sistemas existentes

Para muchas empresas, tan importante como es la disminución de los costes de desarrollo, es la disminución del tiempo de tener disponible los sistemas que se precisan para llevar a cabo sus estrategias de negocio.

- Mayor calidad de los sistemas desarrollados

Verificar que los requisitos establecidos en cualquier proyecto informático se cumplan correctamente

- Mejorar la documentación de los sistemas informáticos y facilitar que esté permanentemente al día

La documentación de los sistemas es un proceso semiautomatizado, puesto que la fuente de la misma es la propia documentación que se genera en cada una de las etapas del desarrollo.

- Mejor seguimiento y gestión de proyectos

Disponer de los mecanismos para incorporar las extensiones que cada organización precise, como *check-list* de actividades, objetos y componentes obtenidos en el proceso de desarrollo.

1.5. La dificultad de medir el software

Si se tiene un proceso en donde las personas que dicen cómo hacer el trabajo son distintas de las personas que realmente lo hacen, los líderes necesitan alguna manera de medir cuán eficaces son los que lo hacen.

Esto es particularmente pertinente al *software* debido a la dificultad de aplicar medidas al *software*. A pesar de nuestros mejores esfuerzos somos incapaces de medir las cosas más simples sobre el *software*, como la productividad. Sin buenas medidas para estas cosas, cualquier clase de control externo está condenado.

La introducción de gestión medida sin buenas medidas tiene sus propios problemas. Se señala que cuando se mide el rendimiento se tiene que conseguir que todos los factores importantes estén bajo medida. Cualquier cosa que se olvide tiene el resultado inevitable que los hacedores alterarán lo que hacen para producir las mejores medidas, incluso si eso claramente reduce la verdadera efectividad de lo que hacen. Este trastorno de la medida es el mayor problema de la gestión basada en medidas.

Al final se tiene que escoger entre la gestión basada en métricas y la gestión delegatoria (donde los realizadores deciden cómo hacer el trabajo). La gestión basada en métricas es más adecuada para el trabajo simple repetitivo, con bajos requisitos de conocimiento y rendimientos fácilmente medibles, exactamente lo contrario al desarrollo de *software*.

El punto de todo esto es que los métodos tradicionales han operado bajo la asunción de que la gestión basada en métricas es la manera más eficaz de administrar.

La comunidad ágil reconoce que las características del desarrollo de *software* son tales que la gestión basada en métricas lleva el trastorno de la medida a niveles muy altos. Es realmente más eficaz usar un estilo delegatorio de administración, que es el tipo de acercamiento central desde el punto de vista ágil.

1.6. Manejando un proceso orientado a la gente

La orientación a la gente se manifiesta de varias maneras diferentes en los procesos ágiles, lo que lleva a efectos diferentes, no todos consistentes.

Uno de los elementos clave es la aceptación de un proceso en lugar de la imposición de un proceso. A menudo los procesos de *software* se imponen desde la gerencia. Como tales se les resiste a menudo, particularmente cuando la gerencia ha estado fuera del desarrollo activo un buen tiempo. Aceptar que el proceso requiere un compromiso, y como tal se necesita el involucramiento activo de todo el equipo.

Esto termina con el resultado interesante de que sólo los desarrolladores pueden escoger seguir un proceso adaptable. Esto es particularmente cierto para la XP (programación extrema), que requiere mucha disciplina para ejecutarse. Aquí es donde Crystal es un complemento efectivo ya que apunta a una disciplina mínima.

Otro punto es que los desarrolladores deben poder tomar todas las decisiones técnicas. XP llega al corazón de esto cuando en su proceso de planeación establece que sólo los desarrolladores pueden estimar cuánto tiempo tomará hacer un trabajo.

Tal liderazgo técnico es un gran cambio para muchas personas en posiciones gerenciales. Tal acercamiento requiere compartir una responsabilidad donde desarrolladores y gerencia tienen un mismo lugar en la dirección del proyecto. La gerencia aún juega un papel, pero reconoce la pericia de los desarrolladores.

Una razón importante para esto es el ritmo del cambio de tecnología en nuestra industria. Después de unos años el conocimiento técnico se vuelve obsoleto. Esta vida media de habilidades técnicas no tiene comparación en cualquier otra industria. Incluso los técnicos tienen que reconocer que entrar a la gerencia significa que sus habilidades técnicas se marchitarán rápidamente.

Los exdesarrolladores necesitan reconocer que sus habilidades técnicas desaparecerán rápidamente y necesitan confiar y depender en los desarrolladores actuales.

Los procesos predecibles requieren componentes que se comporten de manera predecible. Sin embargo las personas no son componentes predecibles. El error de este enfoque es que las "personas" son altamente inconstantes y no lineales, con modos únicos de éxito y fracaso. Esos factores son de primer orden, factores no despreciables. El fracaso de los diseñadores de procesos y metodologías para responder a esto contribuye a la clase de trayectorias de proyectos no planeados que vemos a menudo.

Si uno espera que todos sus desarrolladores se junten en unidades de programación compatibles, no intentará tratarlos como individuos. Esto baja la moral y por lo tanto la productividad.

Las personas buscan el mejor lugar para estar, y esto nos favorecerá ya que se obtienen unidades de programación compatibles o afines. Decidir que las personas son de primero es una gran decisión, que requiere mucha determinación.

Antes se pensaba que la gente que hace el trabajo no es la mejor gente para entender la mejor manera de hacer el trabajo. Últimamente se ha visto cada vez más lo falso que es esto en el desarrollo de *software*. A las personas brillantes y capaces les atrae cada vez más el desarrollo de *software*, tanto por su ostentación como por ganancias potencialmente mayores.

Cuando se quiere contratar y retener a gente capaz, hay que reconocer que son profesionales competentes. Como tales son los mejores para decidir cómo dirigir su trabajo técnico.

1.7. Controlando un proceso impredecible

La parte más importante y difícil, es saber con precisión donde estamos. Necesitamos un mecanismo honesto de retroalimentación que pueda decirnos con precisión cuál es la situación a intervalos frecuentes.

La llave para obtener esta retroalimentación es el desarrollo iterativo. Ésta no es una idea nueva. El desarrollo iterativo ha estado durante algún tiempo bajo muchos nombres: incremental, evolutivo, escenificado, espiral, etc. La clave del desarrollo iterativo es producir frecuentemente versiones que funcionen del sistema final, que tengan un subconjunto de los rasgos requeridos. Estos sistemas son cortos en funcionalidad, pero por otra parte deben ser fieles a las demandas del sistema final. Deben ser totalmente integrados y tan cuidadosamente probados como una entrega final.

El punto es que no hay nada como un sistema probado, integrado para traer una dosis poderosa de realidad en cualquier proyecto. Los documentos pueden esconder toda clase de fallas. El código no probado puede esconder bastantes defectos. Pero cuando las personas realmente se sientan delante de un sistema y trabajan con él, entonces las fallas se vuelven aparentes: tanto las relativas a defectos como las relativas a los requisitos mal entendidos.

El desarrollo iterativo también tiene sentido en los procesos predictivos. Pero es esencial en los procesos adaptables porque un proceso adaptable necesita poder tratar con los cambios en los rasgos requeridos. Esto lleva a un estilo de planear donde los planes a largo plazo son muy fluidos, y los únicos planes estables son a corto plazo hechos para una sola iteración. El desarrollo iterativo da un fundamento firme en cada iteración que puede usarse para basar los planes posteriores.

Uno de los grandes peligros es pretender que se puede seguir un proceso predecible cuando no se puede. La gente que trabaja en metodologías no es buena en identificar condiciones límite: los lugares donde la metodología pasa de apropiada a inapropiada.

La mayoría de los metodologistas quieren que sus metodologías sean usadas por todos, de modo que no entienden ni publican sus condiciones límite. Esto lleva a la gente a usar una metodología en malas circunstancias, como usar una metodología predictiva en una situación imprevisible.

La estimación es difícil por muchas razones, en parte porque el desarrollo de *software* es una actividad de diseño, difícil de planear y cuantificar, en parte porque los materiales básicos cambian rápidamente, en parte por lo mucho que depende de los individuos involucrados, y los individuos son difíciles de predecir y cuantificar.

De cualquier modo dejar ir la previsibilidad no significa que hay que volver al caos ingobernable. Más bien hace falta un proceso que pueda dar control sobre la imprevisibilidad. De eso se trata la adaptabilidad.

Una pregunta importante es cuánto debe durar una iteración. Diferentes personas dan respuestas diferentes. XP sugiere iteraciones de entre una y tres semanas. SCRUM sugiere un mes. Crystal estira aun más. La tendencia, de cualquier modo, es hacer cada iteración tan corta como se pueda manejar. Esto proporciona la retroalimentación más frecuente, así que usted sabe más a menudo donde se encuentra.

1.8. Separación del diseño y la construcción

La inspiración usual para las metodologías han sido disciplinas como las ingenierías civil o mecánica. Tales disciplinas enfatizan que hay que planear antes de construir. Los ingenieros trabajan sobre una serie de esquemas que indican precisamente qué hay que construir y cómo deben juntarse estas cosas. Muchas decisiones de diseño, como la manera de controlar la carga sobre un puente, se toman conforme los dibujos se producen. Los dibujos se entregan entonces a un grupo diferente, a menudo una compañía diferente, para ser construidos. Se supone que el proceso de la construcción seguirá los dibujos. En la práctica los constructores se encuentran con algunos problemas, pero éstos son normalmente poco importantes.

Como los dibujos especifican las piezas y cómo deben unirse, actúan como los fundamentos de un plan de construcción detallado. Dicho plan define las tareas que necesitan hacerse y las dependencias que existen entre estas tareas. Esto permite un plan de trabajo y un presupuesto de construcción razonablemente predecibles. También dice en detalle cómo deben hacer su trabajo las personas que participan en la construcción. Esto permite que la construcción requiera menos pericia intelectual, aunque se necesita a menudo mucha habilidad manual.

Así que lo que vemos aquí son dos actividades fundamentalmente diferentes. El diseño, que es difícil de predecir y requiere personal caro y creativo, y la construcción que es más fácil de predecir. Una vez que tenemos el diseño, podemos planear la construcción. Una vez que tenemos el plan de construcción, podemos ocuparnos de la construcción de una manera más predecible. En ingeniería civil la construcción es mucho más costosa y tardada que el diseño y la planeación.

Así el acercamiento de muchas metodologías es: queremos un plan de trabajo predecible que pueda usar gente del más bajo nivel. Para hacerlo debemos separar el plan de la construcción. Por consiguiente necesitamos entender cómo hacer el diseño de *software* de modo que la construcción pueda ser sencilla una vez que el plan esté hecho.

¿Qué forma toma este plan? Para muchos, éste es el papel de notaciones de diseño como el UML. Si podemos hacer todas las decisiones significativas usando UML, podemos armar un plan de construcción y entonces podemos dar planes a los programadores como una actividad de construcción.

Pero aquí surgen preguntas cruciales. ¿Es posible armar un plan que sea capaz de convertir el código en una actividad de construcción predecible? Y en tal caso, ¿Es la construcción suficientemente grande en costo y tiempo para hacer valer la pena este enfoque?

Todo esto trae a la mente más preguntas. La primera es la cuestión de cuán difícil es conseguir un diseño UML en un estado que pueda entregarse a los programadores. El problema con un diseño tipo UML es que puede parecer muy bueno en el papel, pero resultar seriamente fallido a la hora de la programación.

Los modelos que los ingenieros civiles usan están basados en muchos años de práctica guardados en códigos ingenieriles. Además los problemas importantes, como el modo en que juegan las fuerzas, son dóciles al análisis matemático. La única verificación que podemos hacer con los diagramas UML es la revisión cuidadosa. Mientras esto es útil trae errores al diseño que sólo se descubren durante la codificación y pruebas.

Otro problema es el costo comparativo. Cuando se construye un puente, el costo del esfuerzo en el plan es aproximadamente un 10% del total, siendo el resto la construcción. En *software* la cantidad de tiempo gastada codificando es mucho, mucho menor. Sugiere que para un proyecto grande, sólo 15% del proyecto son código y pruebas unitarias, una inversión casi perfecta de las proporciones de la construcción del puente. Aun cuando se consideren las pruebas parte de la construcción, el plan es todavía 50% del total. Esto genera una pregunta importante sobre la naturaleza del diseño en *software* comparado con su papel en otras ramas de la ingeniería.

2. DESCRIPCIÓN DE LAS METODOLOGÍAS ÁGILES

2.1. Introducción

A continuación se explican detalladamente los aspectos más relevantes de las metodologías ágiles, características, diferencias, etc. Además, para cada una de las metodologías se presenta un caso de ejemplo en el cual se pueden emplear éstas mismas.

2.2. ¿Qué es una metodología ágil?

Las metodologías imponen un proceso disciplinado sobre el desarrollo de *software* con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería.

Las metodologías de ingeniería han estado presentes durante mucho tiempo. No se han distinguido precisamente por ser muy exitosas. La crítica más frecuente a estas metodologías es que son burocráticas. Hay tanto que hacer para seguir la metodología que el ritmo entero del desarrollo se retarda.

Como una reacción a estas metodologías, un nuevo grupo de metodologías ha surgido en los últimos años. Durante algún tiempo se conocían como metodologías ligeras, pero el término aceptado ahora es metodologías ágiles. Para mucha gente el encanto de estas metodologías ágiles es su reacción ante la burocracia de las metodologías monumentales. Estos nuevos métodos buscan un punto medio entre ningún proceso y demasiado proceso,

proporcionando simplemente suficiente proceso para que el esfuerzo valga la pena.

El resultado de todo esto es que los métodos ágiles cambian significativamente algunos de los énfasis de los métodos ingenieriles. La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. De muchas maneras son más bien orientados al código: siguiendo un camino que dice que la parte importante de la documentación es el código fuente.

Los métodos ágiles son adaptables en lugar de predictivos. Los métodos de ingeniaría tienden a intentar planear una parte grande del proceso del *software* en gran detalle para un plazo largo de tiempo, esto funciona bien hasta que las cosas cambian. Así que su naturaleza es resistirse al cambio. Para los métodos ágiles, no obstante, el cambio es bienvenido. Intentan ser procesos que se adaptan y crecen en el cambio, incluso al punto de cambiarse ellos mismos.

Los métodos ágiles son orientados a la gente y no orientados al proceso. La meta de los métodos de ingeniaría es definir un proceso que funcionará bien con cualquiera que lo use. Los métodos ágiles afirman que ningún proceso nunca podrá ocultar las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo. Explícitamente puntualizan el trabajar a favor de la naturaleza humana en lugar de en su contra y enfatizan que el desarrollo de *software* debe ser una actividad agradable.

Los representantes de cada una de estas metodologías fueron invitados a un taller en Snowbird, Utah en febrero de 2001, en donde se produjo una declaración de los principios y valores comunes de los procesos ágiles.

- La prioridad más alta es satisfacer al cliente mediante tempranas y continuas entregas de *software* que le aporte un valor.
- Aceptar requisitos cambiantes, no importando si es tarde en el desarrollo.
- Entregar *software* funcional frecuentemente, de un par de semanas a un par de meses, con la preferencia de que sea el tiempo más corto.
- Negociadores y diseñadores deben trabajar juntos diariamente en todo el proyecto.
- Construir el proyecto en torno a individuos motivados. Hay que darles el ambiente y apoyo que requieren, y la confianza de hacer el trabajo.
- El método más eficiente y eficaz de llevar información dentro del equipo de desarrollo es la conversación cara a cara.
- El *software* que funciona es la medida principal de progreso.
- Los procesos ágiles promueven un desarrollo sostenible. Los patrocinadores, diseñadores y usuarios deben estar disponibles para mantener un paso constante de trabajo indefinidamente.
- Atención continua a la excelencia técnica y un buen plan refuerzan la agilidad.
- La simplicidad[2] es un punto esencial.
- Las mejores arquitecturas, requisitos y planes emergen de los propios equipos organizados.
- A intervalos regulares el equipo refleja en cómo volverse más eficaz, entonces hay que cambiar y ajustar su conducta de acuerdo a esto.

[2] El arte de llevar hasta el máximo la cantidad de trabajo que no se hace.

Varias metodologías encajan bajo el estandarte de ágil. Mientras todas ellas comparten muchas características, también hay algunas diferencias significativas las cuales se destacan en este capítulo.

2.3. XP (eXtreme Programming – Programación Extrema)

2.3.1. ¿En qué consiste la programación extrema?

De todas las metodologías ágiles, ésta es la que ha recibido más atención. De alguna manera, la popularidad de XP se ha vuelto un problema, pues ha acaparado la atención fuera de las otras metodologías y sus valiosas ideas.

La XP empieza con cuatro valores: Comunicación, Retroalimentación, Simplicidad y Coraje. Construye sobre ellos una docena de prácticas que los proyectos XP deben seguir. Muchas de estas prácticas son técnicas antiguas, tratadas y probadas, aunque a menudo olvidadas por muchos, incluyendo la mayoría de los procesos planeados. Además de resucitar estas técnicas, la XP las teje en un todo sinérgico[3] dónde cada una refuerza a las demás.

Una de las características principales, es su fuerte énfasis en las pruebas. Mientras todos los procesos mencionan la comprobación, la mayoría lo hace con muy poco énfasis. Sin embargo la XP pone la comprobación como el fundamento del desarrollo, con cada programador escribiendo pruebas cuando escriben su código de producción. Las pruebas se agregan en el proceso de integración continua y construcción lo que rinde una plataforma altamente estable para el desarrollo futuro.

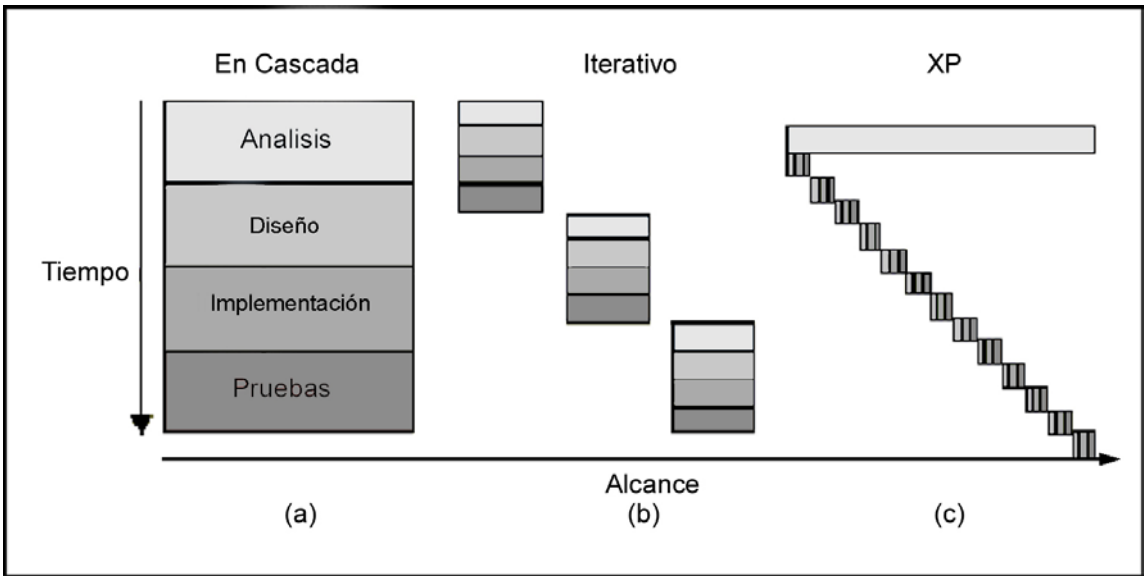
[3] Acción de dos o más causas cuyo efecto es superior a la suma de los efectos individuales.

En esta plataforma XP construye un proceso de diseño evolutivo que se basa en refactorar un sistema simple en cada iteración. Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras. El resultado es un proceso de diseño disciplinado, lo que es más, combina la disciplina con la adaptabilidad de una manera que indiscutiblemente la hace la más desarrollada de entre todas las metodologías adaptables.

2.3.2. Ciclo de vida

Como se ha descrito, los largos ciclos de desarrollo de los métodos tradicionales son incapaces de adaptarse al cambio, hay que hacer ciclos de desarrollo más cortos y esto es lo que presenta la siguiente figura.

Figura 2. Ciclos de Desarrollo entre Metodologías



Evolución de los largos ciclos de desarrollo en cascada (a) a ciclos más cortos (b) y a la mezcla que hace XP (c)

2.3.3. Fases de la metodología XP

Hay diversas prácticas inherentes a la Programación Extrema, en cada uno de los ciclos de desarrollo del proyecto, los cuales se describen a continuación.

2.3.3.1. Planificación

XP plantea la planificación como un permanente diálogo entre la parte empresarial y técnica del proyecto, en la que los primeros decidirán el alcance ¿qué es lo realmente necesario del proyecto?, la prioridad ¿qué se debe de hacer en primer lugar?, la composición de las versiones ¿qué debería incluir cada una de ellas?, y la fecha de las mismas.

En cuanto a los técnicos, son los responsables de estimar la duración requerida para implementar las funcionalidades deseadas por el cliente, de informar sobre las consecuencias de determinadas decisiones, de organizar la cultura de trabajo y, finalmente, de realizar la planificación detallada dentro de cada versión. XP no es sólo un método centrado en el código, sino que sobre todo es un método de gestión de proyectos *software*.

2.3.3.1.1. Se redactan las historias de usuarios

Las historias de usuario tienen el mismo propósito que los casos de uso, pero no son lo mismo. Las escriben los propios clientes, tal y como ven ellos las necesidades del sistema. Por tanto serán descripciones cortas y escritas en el lenguaje del usuario, sin terminología técnica. 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 las pruebas de aceptación. Una o más de estas pruebas se utilizarán para verificar que las historias de usuario han sido implementadas correctamente.

Una de las mayores equivocaciones a cerca del uso de las historias de usuario es la diferencia que existe entre éstas y la tradicional especificación de requisitos. La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionarán los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario. El nivel de detalle de las historias de usuario debe ser el mínimo posible que permita hacerse una ligera idea de cuánto costará implementar el sistema. Cuando se llegue a la fase de implementación, los desarrolladores podrán acudir al cliente para ampliar detalles.

Los desarrolladores deberán hacer una estimación de cuánto tiempo, idealmente, les llevará implementar cada historia de usuario. Las condiciones ideales son aquellas en las que se codifica la historia de usuario sin otras distracciones y sabiendo exactamente qué es lo que hay que implementar. Como resultado deberíamos obtener un periodo ideal de 1, 2 ó 3 semanas. Más de 3 semanas implica que debemos dividir la historia de usuario en partes. Menos de 1 semana implica que la historia de usuario es demasiado sencilla y tendremos que unir dos o más de ellas.

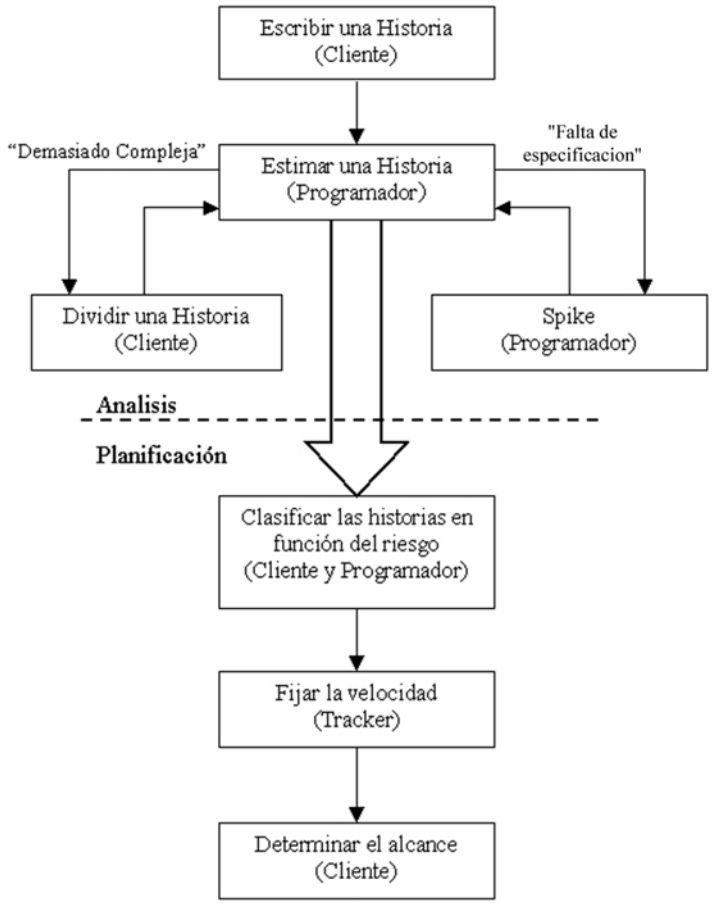
2.3.3.1.2. Crear un plan de entregas

Las historias de usuario servirán para crear el plan estimado de entrega. Se convocará una reunión para crear el plan de entregas. El plan de entregas se usará para crear los planes de iteración para cada iteración. Es en este momento cuando los técnicos tomarán las decisiones técnicas y los comerciales

las decisiones comerciales. En esta reunión estarán presentes tanto desarrolladores como los usuarios. Con cada historia de usuario previamente evaluada en tiempo de desarrollo ideal, el cliente las agrupará en orden de importancia.

Un tiempo ideal es cuánto tiempo costaría implementar dicha historia si nos enfocamos específicamente al trabajo de la misma, incluyendo la parte de prueba correspondiente. De esta forma se puede trazar el plan de entregas en función de estos dos parámetros: tiempo de desarrollo ideal y grado de importancia para el cliente. Las iteraciones individuales son planificadas en detalle justo antes de que comience cada iteración. A modo de esquema:

Figura 3. Plan de Entregas



2.3.3.1.3. Se controla la velocidad del proyecto

La velocidad del proyecto es una medida de cuan rápido se está desarrollando. La velocidad del proyecto se usa para determinar cuántas historias de usuario pueden ser implementadas antes de una fecha dada (tiempo), o cuánto tiempo es necesario para llevar a cabo un conjunto de historias (alcance).

Por ejemplo elegimos de forma arbitraria que las velocidades individuales son de 2.5 (es decir que cada programador puede hacer 2.5 días de trabajo ideal en una iteración de 5 días), si se tienen dos programadores en el proyecto nuestra velocidad de equipo es de 5 (2 x 2.5 = 5), así una historia de usuario de tres días ideales requerirá tres días. Cuando se realiza una planificación por alcance se divide el número total de semanas entre la velocidad de proyecto para determinar en cuántas iteraciones estará disponible el sistema.

2.3.3.1.4. Se divide el proyecto en iteraciones

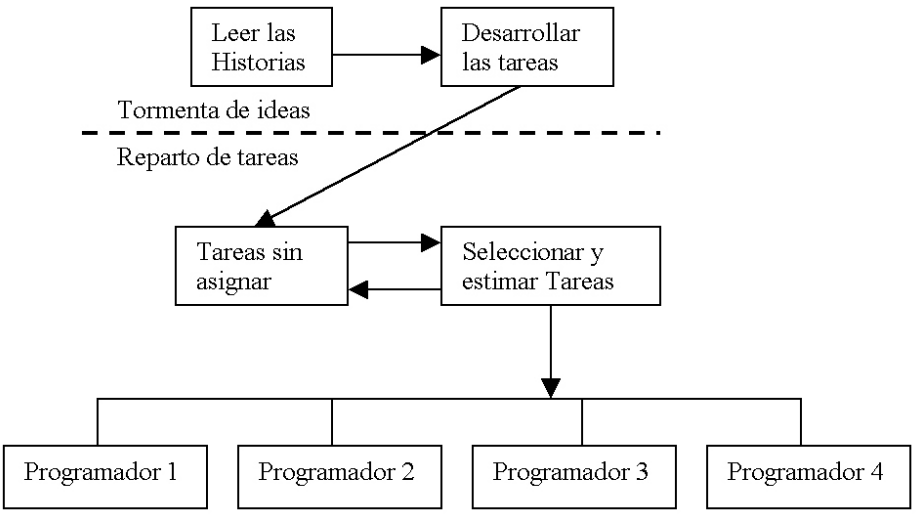
Cada iteración corresponde a un periodo de tiempo de desarrollo del proyecto de entre una y tres semanas. De esta forma, un proyecto, se divide en una docena de iteraciones, más o menos. Al principio de cada iteración se debería convocar una reunión para trazar el plan de iteración correspondiente. Está prohibido intentar adelantarse e implementar cualquier cosa que no esté planeada para la iteración en curso. Habrá suficiente tiempo para añadir la funcionalidad extra cuando sea realmente importante según el plan de entregas.

Se usará la velocidad del proyecto para determinar si una iteración está sobrecargada. La suma de los días que costará desarrollar todas las tareas de la iteración no debería sobrepasar la velocidad del proyecto de la iteración anterior. Si la iteración está sobrecargada, el cliente deberá decidir que historias de usuario retrasar a una iteración posterior. Si, por el contrario, la iteración tiene huecos se rellenará con otras historias de usuario.

2.3.3.1.5. Trazar un plan de iteración

El plan de iteración consiste en seleccionar las historias de usuario que, según el plan de entregas, corresponderían a esta iteración esto se hace al inicio de cada iteración. También se eligen qué pruebas de aceptación fallidas se corregirán. Un plan de iteración puede verse como:

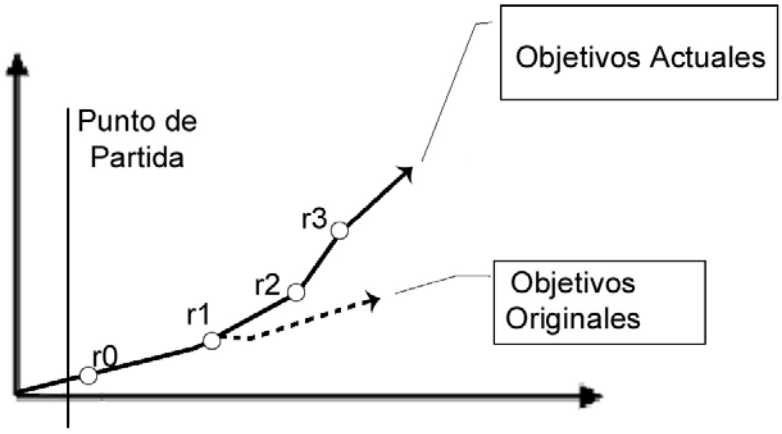
Figura 4. Plan de Iteración



Cada historia de usuario se transformará en tareas de desarrollo. Cada tarea de desarrollo corresponderá a un periodo ideal de uno a tres días de desarrollo.

Es necesario mantener vigiladas la velocidad del proyecto y el movimiento de historias de usuario. Puede ser necesario volver a calcular las historias de usuario y negociar el plan de entrega cada de tres a cinco iteraciones. Como estaremos siempre implementando las historias de los usuarios más importantes para el cliente, estaremos haciendo lo máximo posible por nuestro cliente y la dirección del proyecto. En cada iteración iremos alcanzado unas metas:

Figura 5. ¿Cómo van Variando las Metas?



2.3.3.1.6. Rotación del personal

Las rotaciones evitarán que las personas se conviertan en sí mismas en un cuello de botella. Si sólo una persona de nuestro equipo es capaz de trabajar en un área concreta, existirá un riesgo enorme si esa persona nos deja por cualquier circunstancia. De esta forma, las rotaciones permitirán que todo el mundo conozca cómo funciona el sistema en general y ayudarán a realizar un reparto más equitativo del trabajo. Además el hecho de que se asignen por

parejas nos permitirá entrenar a un nuevo miembro, simplemente dividiendo el grupo original en dos.

2.3.3.1.7. Cada día se convoca una reunión de seguimiento

La comunicación entre las diferentes partes que intervienen en un proyecto resulta fundamental para su desarrollo. Esto se consigue gracias a las reuniones. Podremos analizar los problemas y las soluciones. Se recomienda que éstas sean frecuentes, de poca duración y de ser posible delante de la pantalla del ordenador.

Según la metodología XP, se recomienda que sean diarias; recordemos que los usuarios se considerarán parte integrante del equipo de desarrollo del proyecto. La reunión de seguimiento de cada mañana debe usarse para sacar a la luz los problemas, las soluciones y centrar el objetivo del equipo.

2.3.3.1.8. Corrección de la propia metodología XP cuando falla

Deberemos corregir el proceso cuando éste falle. Cuando se inicia con un proyecto, seguiremos la metodología XP, pero debemos cambiar aquello que no funcione. Además los cambios que se realicen deberán ser comunicados al resto del equipo, todo el mundo debe estar al corriente de los cambios. Esto no significa que se cambiará lo que no guste, sino que se cambiará aquello que no funciona con el problema en particular que se pretende resolver.

2.3.3.2. Diseño

XP establece unas recomendaciones o premisas a la hora de abordar esta etapa.

2.3.3.2.1. Simplicidad

La simplicidad es la llave, siempre costará menos tiempo en implementar un diseño sencillo que uno complejo. Por lo que, trataremos siempre de realizar las cosas de la manera más sencilla posible. Si alguna parte de la implementación resulta especialmente compleja, debiese de ser replanteada (divide y vencerás). Así, cualquier cambio o modificación será mucho más sencillo. En ocasiones, realizar un diseño sencillo puede resultar una tarea especialmente difícil

2.3.3.2.2. Elección de una metáfora para el sistema

Una metáfora para el sistema es una historia que todo el mundo puede contar acerca de cómo funciona el sistema (Kent Beck), la tarea de elegir una metáfora para el sistema nos permitirá mantener la coherencia de nombres de todo aquello que se va a implementar.

El nombre de los objetos o partes de nuestro sistema es muy importante. La tarea de “poner nombre”, es sencilla a simple vista, no lo es tanto. Debemos elegir un sistema de nombres que permita que cualquiera que lo vea adivine la relación entre el objeto y aquello que representa.

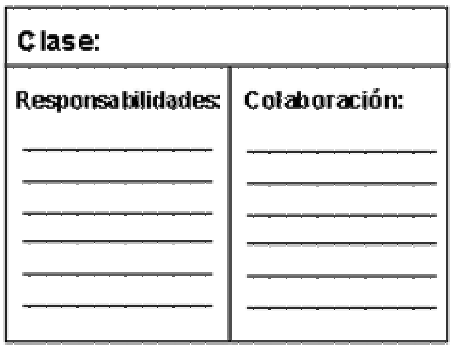
Por ejemplo, decir que una computadora debería de aparecer como un escritorio, o que el cálculo de la planilla es una hoja de calculo; Éstas son metáforas ya que no decimos literalmente “el sistema es una hoja de cálculo”.

Debemos encontrar una imagen de aquello que pretendemos representar en el mundo real. Así se podrá tener una imagen mental más clara.

2.3.3.2.3. Usar tarjetas CRC en las reuniones de diseño

Para poder diseñar el sistema como un equipo deberemos cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración (CRC), las tarjetas CRC permitirán desprendernos del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Las tarjetas CRC permiten que el equipo completo contribuya en la tarea del diseño. Una tarjeta CRC representa un objeto. El nombre de la clase se coloca a modo de título en la tarjeta, las responsabilidades se colocan a la izquierda, y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente.

Figura 6. Tarjeta CRC



2.3.3.2.4. Crear soluciones puntuales para reducir riesgos

Un programa *Spike*, es un programa muy simple que explora una posible solución al problema, es similar a como trabajan los prototipos. A partir de estos pequeños programas iremos construyendo la solución a nuestro problema. El

sistema se pone por primera vez en producción en, a lo sumo, unos pocos meses, antes de estar completamente terminado. Las sucesivas versiones serán más frecuentes (entre un día y un mes).

2.3.3.2.5. No añadir funcionalidad en las primeras etapas

Debemos evitar caer en la tentación de ir añadiendo funcionalidades según se nos vayan ocurriendo, aún incluso que sepamos exactamente cómo implementarlas. Es decir, debemos centrarnos en la tarea que se ha fijado para hoy, y hacerla lo mejor posible. Programaremos lo que se ha fijado, y no perderemos el tiempo en desarrollar código que no se sabe si será utilizado.

2.3.3.2.6. Reaprovechar cuando sea posible

Cuando eliminamos redundancia, eliminamos funcionalidad inútil, y rejuvenecemos antiguos diseños, estamos reciclando código. El reciclaje, dentro del ciclo de vida de un proyecto, ahorra tiempo e incrementa la calidad. El reciclaje implicará mantener el código limpio y fácil de comprender, modificar y ampliar. Esto puede resultar un poco costoso al principio, pero resulta fundamental a la hora de realizar diseños futuros.

2.3.3.3. Desarrollo

Esta etapa debe reunir las siguientes características o cualidades:

2.3.3.3.1. El cliente está siempre disponible

Una de las pocas condiciones que impone la metodología XP es tener al usuario siempre disponible. No sólo para ayudar al equipo de desarrollo, sino formando parte de él. Todas las fases que se realizan en un proyecto XP

requieren de comunicación con el usuario, preferiblemente cara a cara, en persona, sin intermediarios. Durante la reunión del plan de entregas, el usuario propondrá qué historia de usuario se incluye en cada plan. También se negociarán los plazos de entrega.

El usuario o cliente tomará las decisiones que le afecten para alcanzar los objetivos de su negocio. También es necesario que el cliente colabore en la realización de las pruebas. Estas pruebas comprobarán que el sistema está listo para pasar a la fase de producción. El usuario comprobará los resultados obtenidos y tomará decisiones en cuanto a la utilización o no del sistema realizado.

2.3.3.3.2. Se debe escribir código de acuerdo a los estándares

El código ha de ser desarrollado siguiendo los estándares de desarrollo para facilitar su lectura y modificación por cualquier miembro del equipo de desarrollo. Es decisiva, para poder plantear con éxito la propiedad colectiva del código. Ésta sería impensable sin una codificación basada en estándares que haga que todo el mundo se sienta cómodo con el código escrito por cualquier otro miembro del equipo.

2.3.3.3.3. Desarrollar primero las unidades de prueba

Cuando las pruebas son creadas antes que el código, la implementación del código será mucho más rápida. El tiempo empleado en desarrollar una prueba y algo de código para probarlo es aproximadamente el mismo tiempo que se emplea en crear exclusivamente dicho código. La creación de las unidades de pruebas ayuda al programador a tener una visión acerca del cómo, en definitiva, del comportamiento del programa. Además, aún estamos a

tiempo de dar marcha atrás, ya que el programador no ha concluido la implementación.

Esta manera de trabajar resulta especialmente beneficiosa en el diseño de complicados sistemas de *software* (cualquier característica de un programa para la que no haya una prueba automatizada, simplemente no existe), y es que éste es sin duda el pilar básico sobre el que se sustenta XP.

Otros principios son susceptibles de ser adaptados a las características del proyecto, de la organización, del equipo de desarrollo.

Aquí no hay discusión posible: si no se hacen pruebas, no estamos haciendo XP. Para ello, deberemos emplear algún *framework* de pruebas automático, como JUnit o cualquiera de sus versiones para diferentes lenguajes. No sólo eso, sino que se deben de escribir las pruebas incluso antes de que se escriba la clase a probar.

2.3.3.3.4. Todo el código debe programarse por parejas

Todo el código que formará parte del plan, será desarrollado por dos personas que trabajarán de forma conjunta en un ordenador. De esta manera, se incrementará la calidad del *software* desarrollado sin afectar al tiempo de entrega. Partimos de la base de que este equipo de dos personas posee unos conocimientos similares en cuanto a la tarea que van a realizar, es decir, están aproximadamente al mismo nivel.

Mientras uno de ellos se encarga de pensar la táctica con la que se va a abordar el problema, el otro se encargará de pensar las estrategias que

permiten llevar dichas tácticas a su máximo exponente. Ambos roles son intercambiables.

2.3.3.3.5. Sólo una pareja se encargará de integrar el código

Una forma de hacer esto es tener una maquina dedicada a la integración. Cuando la maquina este libre, una pareja con código para integrar, se sienta, carga la versión actual, carga sus cambios (probando y resolviendo cualquier colisión) y ejecuta las pruebas hasta que funcionan 100% correctas, en un tiempo determinado solo hay una pareja integrando el código.

En esta etapa pueden aparecer problemas debidos a la integración de los módulos que se han desarrollado y no han sido probados todavía, la corrección de estos errores los hace la pareja que los originó ya que la última dejó el sistema funcionando al 100%, si no se logran que las pruebas estén al 100%, se tira lo que se hizo y se comienza de nuevo. Las pruebas deberán ser completas, en el sentido de que, un fallo de las mismas podría derivar que determinados errores pasarán inadvertidos.

2.3.3.3.6. Integración frecuente

Los programadores deberán actualizar sus módulos con las versiones más recientes del trabajo realizado tan pronto como les sea posible. De esta manera, todo el mundo trabajará siempre con la última versión. Dicha actualización es responsabilidad de cada pareja de programadores.

Esta integración se llevará a cabo cuando el éxito para su prueba correspondiente sea del 100%, o cuando se trate de una parte que constituye

un todo funcional, en cuanto esté acabada. Esta frecuencia con la que se inserta el nuevo código nos permitirá una rápida detección de los problemas de compatibilidad.

2.3.3.3.7. Todo el código es común a todos

Esta filosofía nos permite que cualquiera contribuya al desarrollo de cualquier parte del proyecto. Cualquier programador podrá cambiar una línea de código para añadir funcionalidad o eliminar algún fallo. Las personas dejan de ser un cuello de botella, en cuanto a la programación. Cualquiera puede realizar un cambio en el mismo siempre y cuando beneficie a la arquitectura del mismo. La responsabilidad del funcionamiento recaerá sobre el equipo al completo.

2.3.3.3.8. Dejar las optimizaciones para el final

No optimizaremos el código hasta el final. Nunca trataremos de averiguar cuales serán los posibles cuellos de botella del programa. “Hacer el trabajo, hacerlo bien, y entonces hacerlo rápido”.

2.3.3.3.9. No trabajar más de 40 horas semanales

Trabajar horas extras absorbe el espíritu y la motivación del equipo. Aquellos proyectos que requieren horas extras para acabarse a tiempo pueden convertirse en un problema. En lugar de esto, utilizaremos las conocidas reuniones de plan de entregas para cambiar los objetivos del proyecto.

También es una mala idea incorporar nueva gente al proyecto, una vez que este ya ha comenzado, con el fin de que ayuden a la finalización de esté.

Debido a que solo retrasaran más el proyecto ya que no tienen conocimiento sobre lo que se esta haciendo y como se esta haciendo y les tomara tiempo entenderlo y solo consumirán recursos, por lo el proyecto solo se retrasaría más. Se trabajará un máximo de 40 horas semanales. Nadie es capaz de trabajar 60 horas a la semana y hacerlo con calidad.

2.3.3.4. Pruebas

2.3.3.4.1. Todo el código debe de ir con su unidad de pruebas

Las unidades de *test* o pruebas constituyen unos de los pilares básicos de la Extreme Programming (XP), uno de los errores que se suele cometer es pensar que podemos dejar la construcción de los *test* para los últimos meses en la realización de un proyecto. Descubrir todos los errores que pueden aparecer lleva tiempo, y más si dejamos la depuración de todos para el final. Las unidades de *test* están directamente relacionadas con el concepto de posesión del código. En cierta manera, una parte del código no será reemplazado si no supera los *test* que existen para ese código.

Después de cada modificación, podremos emplear los *test* para verificar que un cambio en la estructura no introduce un cambio en la funcionalidad. Sin embargo, si se añade nuevas capacidades a nuestro código, tendremos que rediseñar la unidad de *test*, para adaptarse a la nueva funcionalidad. De esta manera, la probabilidad de que exista un fallo en ambos (*test* y código) es menor. Si creásemos los *test* después de la creación del código tenderíamos a hacerlos utilizando nuestro código como un generador, trasladando los fallos de uno al otro.

De aquí la importancia de la creación de las unidades de prueba antes que el código, para que sean independientes de este último. Las pruebas se convierten en una herramienta de desarrollo, no un paso de verificación que puede despreciarse si a uno le parece que el código está bien. De alguna forma, se ofrece una alternativa a los tradicionales “print” que todo el mundo ha usado y que después son eliminados del programa final.

2.3.3.4.2. Hacer todas las pruebas antes de implantar

Las unidades de *test* serán incluidas junto con el código que verifican dentro del repositorio. El código no se considerará completo si éste no consta de su unidad de *test* correspondiente. El código será implantado cuando supere sus correspondientes unidades de *test*.

2.3.3.4.3. Ante un fallo, una unidad de pruebas

Cuando se detecte un fallo, se debe crear una unidad de pruebas para protegernos del mismo. De esta manera, la localización del mismo será mucho más fácil por parte de los programadores. Este nuevo *test* será empleado para aislar el fallo y depurarlo.

2.3.3.4.4. Se deben ejecutar pruebas de aceptación a menudo y publicar los resultados

Las pruebas de aceptación son creadas a partir de las historias de usuario. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. El cliente o usuario especifica los aspectos a probar cuando una historia de usuario ha sido correctamente implementada.

Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento. Una prueba de aceptación es como una caja negra. Cada una de ellas representa una salida esperada del sistema. Es responsabilidad del cliente verificar la corrección de las pruebas de aceptación y tomar decisiones acerca de las mismas. Una historia de usuario no se considera completa hasta que no supera sus pruebas de aceptación.

Esto significa que debe desarrollarse un nuevo *test* de aceptación para cada iteración o se considerará que el equipo de desarrollo no realiza ningún progreso. Las pruebas de aceptación deberían automatizarse ya que se deben pasar frecuentemente. La puntuación de las pruebas de aceptación se hará pública a todo el equipo. La garantía de calidad (*Quality Assurance* - QA), es una parte esencial en el proceso de XP.

La realización de este tipo de pruebas y la publicación de los resultados deben ser lo más rápido posible, para que los desarrolladores puedan realizar con la mayor rapidez posible los cambios que sean necesarios. A las pruebas de aceptación también se las conoce con el nombre de pruebas de funcionalidad, y constituyen la garantía de que los requerimientos fijados por los usuarios han sido reflejados en el sistema.

2.3.4. Caso

Requerimientos iniciales

Una empresa que se dedica a la venta de equipo electrónico de todo tipo (cámaras digitales, televisores, radios, pantallas, proyectores, computadoras, etc.) desea tener presencia en Internet, el empresario le dice al desarrollador

que lo principal para él es mostrar los productos que tiene a la venta para que así las personas los puedan conocer, y por lo tanto esta parte del proyecto la desea lo antes posible.

Requerimientos finales

Después cuando ya esté implantada la primera parte quiere agregar una sección en el sitio en donde se pueda tener la atención al cliente (reparación y mantenimiento de equipo), búsqueda de productos y sus precios.

Cambio de los requerimientos (durante la fase de diseño)

Una semana después el empresario decide que lo mejor sería crear una sección de venta en línea y que esta parte sea la primera que esté lista del proyecto, esto conlleva a la creación de módulos de seguridad de transferencia de información y el módulo de mantenimiento de los productos disponibles.

Cambio en los requerimientos (al concluir la primera iteración)

Después decide que quiere nuevas secciones para cierto tipo de clientes en el sitio como “Soluciones Empresariales”, “Computadoras”, “Música”, “Car Audio”, “Regalos”, etc. Esto implica cambiar la estructura que posee el sitio.

Nuevos requerimientos (al concluir la tercera iteración)

Poco tiempo después el empresario piensa que también sería ideal que la página esté disponible en inglés y en español. Todos estos cambios se han dado después de mes y medio de haber iniciado el proyecto.

2.3.4.1. Descripción de la solución

Se creará el plan de entregas a partir de las historias de los usuarios ya clasificadas y analizadas por tiempo de desarrollo y grado de importancia para

el cliente. La solución se entregará por medio de iteraciones que están planificadas de dos a tres semanas. Al inicio de cada iteración se convocará a una reunión para trazar el plan de iteración correspondiente esto permitirá la adaptabilidad.

Al concluir cada iteración se realizará la integración para que todo el equipo pueda trabajar con la última versión liberada. Se realizarán todas las pruebas correspondientes antes de integrar, cada vez que se agreguen nuevas funcionalidades al proyecto se rediseñarán las pruebas, para adaptarse a las nuevas funcionalidades.

Todos los cambios que se realicen serán comunicados al resto del equipo, para esto se definirán canales de comunicación para hacer llegar la información tales como memorando, correo electrónico, y reuniones al final del día.

Se implementarán tarjetas CRC para trabajar con una metodología basada en objetos y además permitir que el equipo contribuya en el diseño. Durante todo el proyecto se crearán programas *Spike*. En algunas partes del sistema se utilizaran componentes que ya posee la empresa lo cual acelerara el desarrollo. El cliente ha garantizado su completa colaboración hacia el equipo de desarrollo, con las reuniones de entregas y las realizaciones de las pruebas.

Se desarrollará el proyecto de acuerdo a los estándares internacionales para que así cualquier persona lo pueda comprender y poder dar seguimiento al trabajo o mantenimiento, además el código se realizará en parejas para incrementar la productividad y calidad del desarrollo como lo indica la metodología en cuestión.

Todo el proyecto se realiza con páginas dinámicas PHP y con la base de datos MySQL, esto debido a que la empresa desarrolladora tiene experiencia utilizando estas herramientas, lo cual presenta un valor para el negocio y los clientes no presentaron ningún inconveniente al respecto.

Como sistema operativo para el servidor se utilizará Linux, como las páginas serán PHP se utilizará un servidor Apache. Otra característica que poseen estas herramientas mencionadas es que son libres de licencias y esto ayudará a reducir los costos totales del proyecto en los que tienen que incurrir los clientes.

2.3.4.2. ¿Por qué XP?

Uno de los principales motivos del por qué se debe de utilizar XP para el desarrollo es este proyecto es que la metodología en cuestión trabaja por medio de iteraciones, como es un método auto adaptable se hacen revisiones continuas sobre el proceso al final de cada iteración, esto con el fin de corregir y cambiar lo que ya no funcione o no sea necesario.

Todo el diseño se centra en la iteración actual y no se hace nada anticipadamente para necesidades futuras, no se debe perder el tiempo en hacer cosas que no se sabe si servirán. Debido a que van cambiando las metas, si se da el caso en que hay que votar el trabajo hecho, sólo se pierde el esfuerzo de la iteración y no el esfuerzo del proyecto.

El resultado es un proceso de diseño disciplinado, lo que es más, combina la disciplina con la adaptabilidad de una manera que indiscutiblemente es necesaria en este proyecto. Para que el proyecto tenga éxito a pesar de los requisitos volátiles que se han dado es que el cliente este dispuesto en todo

momento para colaborar, no sólo con el equipo de desarrollo sino formando una parte integral de éste mismo. XP anima a la gente a afinar el proceso.

Además de todo esto el proyecto no requiere un gran número de personal por parte del equipo de desarrollo, no más de veinte como lo sugiere la propia metodología.

Una ventaja adicional es la forma en que se trabajan las pruebas, esto ayudara a garantizar la calidad del producto, reducirá costos y tiempo. Los casos de pruebas se ejecutaran según avanza el proyecto.

2.4. DSDM (*Dynamic Systems Development Method*)

2.4.1. Historia

DSDM (Método de Desarrollo de Sistema Dinámico) empezó en Gran Bretaña en 1994 como un consorcio de compañías del Reino Unido que querían construir sobre Desarrollo Rápido de Aplicaciones y desarrollo iterativo. Habiendo empezado con 17 fundadores ahora tiene más de mil miembros y ha crecido fuera de sus raíces británicas. Siendo desarrollado por un consorcio, tiene características diferentes a muchos de los otros métodos ágiles. Tiene una organización de tiempo completo que lo apoya con manuales, cursos de entrenamiento, programas de certificación y demás.

2.4.2. ¿Qué es DSDM?

La clave está en entregar lo que el negocio necesita cuando lo necesita. El método empieza con un estudio de viabilidad y negocio. El estudio de viabilidad considera si DSDM es apropiado para el proyecto. El estudio de

negocio es una serie corta de talleres para entender el área de negocio dónde tiene lugar el desarrollo. También propone arquitecturas de esbozos del sistema y un plan del proyecto.

El resto del proceso forma tres ciclos entretejidos: el ciclo del modelo funcional produce documentación de análisis y prototipos, el ciclo de diseño del modelo realiza el esquema del sistema para uso operacional, y el ciclo de implantación se ocupa del despliegue al uso operacional.

DSDM tiene principios subyacentes que incluyen una interacción activa del usuario, entregas frecuentes, equipos autorizados, pruebas a lo largo del ciclo. Como otros métodos ágiles usan ciclos de plazos cortos de entre dos y seis semanas. Hay un énfasis en la alta calidad y adaptabilidad hacia requisitos cambiantes.

DSDM es notable por tener mucha de la infraestructura de las metodologías tradicionales más maduras, al mismo tiempo que sigue los principios de los métodos ágiles. DSDM no es un método en cierto sentido más bien es una estructura enfocada a entregar una solución de calidad en forma rápida.

2.4.3. La filosofía de DSDM

- El desarrollo es un esfuerzo de equipo. Se debe combinar el conocimiento de los clientes sobre los requisitos del negocio con las habilidades técnicas de nuestros profesionales.

- Alta calidad demanda capacidad para una robustez técnica.

- El desarrollo puede ser incremental, no todo se tiene que entregar enseguida, y entregar algo más temprano es a menudo más valioso que entregar todo más tarde.

- La ley de disminuir ingresos se aplica, se debe gastar en el desarrollo de recursos que tengan más valor para el negocio.

2.4.4. Los principios

2.4.4.1. Envolvimiento del usuario activo, es imperativo

Los usuarios son participantes activos en el proceso de desarrollo. Si no se envuelven estrechamente a los usuarios en todo el ciclo de vida del desarrollo ocurrirán retrasos y los usuarios concluirán que es por los diseñadores y/ o manejo.

2.4.4.2. Se debe autorizar al equipo para tomar decisiones

Los equipos DSDM constan de diseñadores y usuarios. Deben poder hacer decisiones para refinar los requisitos y posibles cambios. Deben poder estar de acuerdo en cierto nivel de funcionalidad, desempeño, etc.

Para esto es primordial que los equipos tengan conocimiento sobre el negocio para el cual se está creando el sistema, para no tomar decisiones que afecten de alguna manera al proyecto o a los usuarios.

2.4.4.3. El enfoque frecuentemente es la entrega de productos

Un producto basado es más flexible que una actividad basada. El trabajo de un equipo DSDM se concentra en productos que se pueden entregar deliberadamente de acuerdo a un período de tiempo. Para mantener cada período de tiempo corto, el equipo puede decidir fácilmente que actividades son necesarias y suficientes para alcanzar los productos correctos.

Los productos incluyen elementos de desarrollo internos y no sólo la entrega de sistemas, como la creación o adaptación de componentes, clases globales, métodos, etc.

2.4.4.4. Conocimiento amplio del negocio

El enfoque de DSDM es entregar adelantadamente los requisitos esenciales del negocio dentro de los requerimientos de tiempo. Se constituye que un negocio cambiante requiere concesión dentro de un límite de tiempo.

Al poseer una mayor capacidad para el negocio propuesto es mucho más fácil poder adaptar el proyecto a los cambios que puedan surgir ya que algunos se pueden prever y tenerlos presentes si en caso surgieran. Esto formaría parte del plan de contingencias del proyecto, también podría darse el caso de un requerimiento totalmente nuevo (no previsto), y en este caso el conocimiento que tengamos sobre el negocio nos dará la ventaja suficiente para superar el cambio sin que afecte el desarrollo del sistema.

2.4.4.5. El desarrollo iterativo e incremental son necesarios

DSDM deja que los sistemas crezcan incrementalmente. Por consiguiente los diseñadores pueden hacer uso completo de la

retroalimentación de los usuarios. Además se pueden entregar y satisfacer necesidades del negocio que sean inmediatas pero que son parciales.

Se construye dentro del trabajo en los procesos DSDM; así, el desarrollo puede proceder más rápidamente durante la iteración. Al crear el proyecto en forma incremental se puede converger fácilmente en una solución exacta del negocio ya que con cada iteración se van ajustando las necesidades requeridas.

2.4.4.6. Todos los cambios durante desarrollo son reversibles

Se debe de controlar la evolución de todos los productos, todo debe estar en un estado conocido todo el tiempo. Rastreo hacia atrás es una característica de DSDM. Sin embargo en algunas circunstancias puede ser más fácil reconstruir que retroceder. Esto depende de la naturaleza del cambio y el ambiente en el que se hizo.

2.4.4.7. Los requisitos son de alto nivel

Los lineamientos principales son los requisitos de alto nivel y están "congelados" y están de acuerdo con el propósito y alcance del sistema, esto da margen para detallar la investigación de lo que los requerimientos implican. Más adelante, se puede establecer más lineamientos principales en el desarrollo, aunque el alcance no debe cambiar significativamente.

2.4.4.8. Pruebas integradas en todo el ciclo de vida

No se trata la comprobación como una actividad separada. Se desarrolla incrementalmente dentro del sistema, también se prueba y repasa por ambos

diseñadores y usuarios incrementalmente para asegurar que el desarrollo avanza no sólo en la dirección correcta del negocio sino que también en un solo sentido.

2.4.4.9. Colaboración y cooperación entre todos los participantes son esenciales

La naturaleza de los proyectos DSDM significa que hay requisitos a bajo nivel y que no necesariamente se arreglan cuando empieza el proyecto. A corto plazo la dirección que un proyecto toma debe decidirse rápidamente sin que existan restricciones en los recursos para que el cambio de los procedimientos se pueda dar.

Los stakeholders incluyen no sólo el personal de negocio y desarrollo dentro del proyecto, sino que también a otros empleados tal como el soporte técnico o gerentes de recursos.

2.4.5. Caso

La NASKA (North America Sport Karate Association) ha decidido digitalizar toda la información que posee sobre los torneos que tienen a cargo entre estos se encuentra el US Open el cual es el de mayor importancia para la asociación, entre los datos que poseen está la información de los participantes en cada una de las distintas categorías, clasificados, campeones, etc. Toda la información se encuentra en fichas llenadas a mano por los organizadores.

Requerimientos

La empresa que ha contratado la asociación se dedica al desarrollo de aplicaciones para PC's terminales (cliente-servidor), pero los organizadores han especificado que desean que la aplicación este como un portal en la web,

donde se tiene acceso a toda la información digitalizada de los eventos anteriores, el ranking que tiene cada participante dentro de esta asociación y como ha variado a lo largo del tiempo.

También se podrá consultar la información de las actividades futuras (fechas, lugares, horario, patrocinadores, etc.), estadísticas, reglamento y proyecciones.

Características relevantes

Una característica adicional e importante para este proyecto es que faltan menos de 2 meses para el US Open, y los organizadores desean que para esta fecha el sistema esté listo y así poder utilizarlo. La solución también deberá de ayudar para poder obtener los resultados de una forma más rápida ya que los ganadores se definen por acumulación de puntos, y este proceso también se lleva a mano.

2.4.5.1. Descripción de la solución

La arquitectura está formada por un servidor Web Windows 2003 con Microsoft Internet Information Server (IIS), el cual dará páginas ASPX para obtener la información en forma dinámica, para lo cual se ingresará el código del asociado en el caso de que se quiera consultar la información que se posee de él.

Se generarán distintos tipos de reportes dependiendo de la información que ingrese el visitante como fechas, categorías, tipo de evento (abierto o circuito cerrado), ranking, etc.

La base de datos se tendrá en Microsoft® Access 2000, la cual proporcionará la información necesaria para la creación de las páginas, ésta se actualizará constantemente según ocurran las distintas actividades en las que esté envuelta la asociación.

2.4.5.2. ¿Por qué DSDM?

Una de las razones por que resulta provechoso implementar este proyecto utilizando la metodología en cuestión es que ésta enfatiza que se debe de entregar lo que el negocio necesita cuando lo necesita.

Adicional a esto se puede agregar el énfasis que se realiza en la alta calidad hacia los requerimientos variables, y al cambio de la estructura del sistema.

Dado que se lanza una versión ligera (no completa) del sistema para la parte que más urge del sistema, después ésta puede sufrir cambios al momento de que se concluya el sistema pero se habrá cubierto la urgencia principal del proyecto.

Otra característica es que los desarrolladores nunca han trabajado en este tipo de proyectos y no conocen como el sistema debe de manejar los torneos (acumulación de puntos, clasificaciones, categorías, descalificaciones, amonestaciones, etc.), es por eso que se combinan los conocimientos de los clientes sobre los requisitos del negocio para ayudar a los desarrolladores.

Y lo que puede ser la característica más importante es que esta metodología apoya con manuales y cursos de capacitación para que así se pueda solventar el problema de la especialidad de los desarrolladores en aplicaciones de cliente-servidor y puedan crear el sistema sobre una

infraestructura para la web, con esto vendrá un beneficio a mediano plazo para la empresa ya que el conocimiento adquirido no solo servirá para este proyecto sino que se reutilizara en proyectos posteriores.

2.5. Crystal Clear

2.5.1. ¿Qué es Crystal Clear?

El creador de esta metodología es Alistair Cockburn el cual ha estado trabajando en metodologías desde que la IBM le encargó escribir sobre metodologías a inicios de los 90. No obstante, su acercamiento no es como la mayoría de los metodologistas. En lugar de partir solamente de su experiencia personal para construir una teoría de cómo deben hacerse las cosas, él complementa su experiencia directa con la búsqueda activa de proyectos y ver cómo trabajan.

Crystal Clear es una metodología para equipos pequeños de 4 a 6 personas, comparte con la XP una orientación humana, pero esta centralización en la gente se hace de una manera diferente. Se considera que las personas encuentran difícil seguir un proceso disciplinado, así que más que seguir la alta disciplina de la XP, se explora una metodología menos disciplinada que aun podría tener éxito, intercambiando conscientemente productividad por facilidad de ejecución. Esto es lo que nos lleva a pensar que Crystal es menos productivo que la XP, pero más personas serán capaces de seguirlo.

Las estrategias de proyecto se hacen decidiendo qué cuestiones son predecibles, impredecibles pero que se pueden resolver o irresolubles, decidiendo cuales de éstas son propuestas por información o por flexibilidad y como asignar recursos para cada una de ellas.

Una propuesta de dinero por información (DPI) es en la que el equipo puede elegir gastar recursos ahora para ganar información cuanto antes. Si la información no es considerada de valor suficiente los recursos se aplicarán a otro trabajo. El problema es cuánto desea gastar el equipo a cambio de esta información. En una propuesta de dinero por flexibilidad (DPF) es cuando el equipo puede optar por ahorrar los recursos para tener más flexibilidad en el futuro.

Las cuestiones predecibles pueden ser investigadas usando técnicas de disipación, una cuestión así puede ser el crear un calendario de trabajo. Las cuestiones impredecibles pero que se pueden resolver, pueden ser investigadas con técnicas de estudio como pueden ser los prototipos y simuladores que evalúan el rendimiento del sistema, éstas son propuestas DPI ya que se invierte esfuerzo y dinero para crear el prototipo para obtener información o aclarar determinados aspectos del proyecto.

Las cuestiones irresolubles suelen ser de origen sociológico, como si un estándar emergente va a ser aceptado por el mercado o cuanto tiempo van a permanecer los empleados claves en la empresa. Éstas son de vital importancia y por lo tanto no pueden ser resueltas con una propuesta DPI sino con una DPF.

Los equipos ágiles y los equipos orientados a un plan, intrínsicamente utilizarán diferentes estrategias para estas cuestiones. Los equipos ágiles se prepararán para absorber estos cambios, mientras que los equipos orientados a un plan tienden, por definición, a hacer planes para afrontarles.

Existen características como poder reemplazar documentación escrita con interacciones cara a cara, se puede reducir la confianza en productos de

trabajo escritos y mejorar la probabilidad de entregar el sistema, lo que beneficia ambas partes contratistas y contratados. Se pueden entregar en forma mas frecuente funcionalidades, probando partes del sistema.

2.5.2. Las propiedades

A continuación se describirán las propiedades para los equipos pequeños exitosos. Crystal Clear requiere entrega frecuente, comunicación osmótica[4], mejora reflexiva y fácil acceso a los usuarios o expertos. Todas estas propiedades no son sólo para proyectos de equipos pequeños, a excepción de la comunicación osmótica. Cada una se iguala a la magnitud necesaria en los grandes proyectos.

2.5.2.1. Entrega frecuente

Simple y sencillamente ésta es la propiedad más importante en todos los proyectos, largos o pequeños, ágiles o no, y es que los usuarios finales prueben el código cada cierto tiempo (pocos meses), las ventajas que se obtienen por esto son:

- El equipo obtiene una depuración de su proceso de desarrollo y hábitos de trabajo.

- Los desarrolladores mantienen su foco, rompiendo los bloqueos de la indecisión.

[4] Ósmosis: Mutua influencia entre dos personas o grupos de personas, sobre todo en el campo de las ideas.

- Los usuarios obtienen una oportunidad de descubrir si sus requisitos originales son los que ellos necesitan actualmente y obtener su retroalimentación dentro del desarrollo.

- Los equipos consiguen una buena moral debido a los cumplimientos pertinentes.

- Los patrocinadores dan retroalimentación crítica sobre el valor del progreso del equipo.

Todas estas ventajas provienen de una simple práctica, la entrega frecuente. No deben de haber periodos largos que sobrepasen los cuatro meses ya que éstos no ofrecen seguridad. Dos meses son mucho más seguros, y con un desarrollo basado para la Web, no es extraordinario que la entrega sea mensual o semanal.

En ocasiones la "entrega" significa que el *software* es implantado por una parte del equipo al final de cada iteración. Esto sucede con el *software* implantado para Web y donde el grupo de usuarios es relativamente pequeño, y posiblemente el equipo de diseño es local, es decir que son parte de los clientes.

En ocasiones los usuarios no aceptan las actualizaciones del *software*, por lo regular es la mayoría de veces. Si el sistema se entrega frecuentemente, no les será tan difícil a los usuarios aceptarlo y adaptarse al cambio. Si no se hace esto, es debido a que existe un problema de integración o un problema de implantación, este problema se encuentra cuando ya es muy tarde, al momento de implantar el sistema.

Hay dos posibles soluciones:

- Integre pero no implante. En este caso es dejar la implantación como un elemento no depurado del proceso.

- Encontrar un usuario amistoso, uno a quién no le ocasione molestia probar el *software*, ya sea por curiosidad o por cortesía.

La segunda solución es la mejor de las dos porque el equipo consigue practicar la implantación, y puede también conseguir una retroalimentación útil proveniente del usuario amistoso. Sin embargo, ambas permiten que el equipo realice la integración de sistema y prepare el sistema para la entrega. Actualmente, hay una cierta confusión sobre la diferencia entre las integraciones, las iteraciones y los "release" (lanzamientos), que son las versiones que ven los usuarios. La entrega frecuente es sobre los releasing del *software* para los usuarios, poniéndolo en sus máquinas, iterando todo no sólo el proceso.

Simplemente la ejecución de una integración de sistema no constituye una iteración, al final de una integración, el sistema todavía está en el dominio de los desarrolladores, y no de los usuarios. La integración frecuente debe ser una norma, que debe de suceder cada hora, cada día, o en el peor de los casos, cada semana. Los mejores equipos actualmente ejecutan en forma automatizada la construcción y prueba del código, hasta llegar al punto de que no pasan más de 30 minutos desde una revisión hasta que se fijan los resultados de la siguiente prueba automatizada.

La iteración se refiere al equipo, sobre una base regular, terminando una sección del trabajo, integrando el sistema, divulgando el resultado encima de la cadena de la gerencia, haciendo su reflexión periódica (debería de ser así), y

muy importante, consiguiendo la parte emocional de que se va ha terminar el trabajo. Esto último es importante porque define un ritmo emocional para el proyecto, algo que es importante para nosotros como seres humanos. Una iteración puede ser a partir de una hora a un año de largo, en principio. La longitud de la iteración se elige en la práctica generalmente a partir de dos semanas a tres meses.

La longitud de una iteración por lo general es fija e inamovible. La razón principal de hacer esto es que hay una tentación de ampliar una iteración cuando el equipo se retrasa. Esto ha demostrado generalmente ser una mala estrategia, lo mejor es trabajar sin cambiar la fecha, y forzar al equipo a entregar lo que tiene que terminar en ese tiempo.

Los requisitos son a veces bloqueados durante una iteración, o sea que no se permiten modificaciones de éstos. Esta combinación de bloquear el tiempo y los requisitos se conoce como Encaje de Tiempo. La fijación de los requisitos da al equipo una cierta paz mental mientras trabajan, asegurándolos que no serán molestados mientras que entregan algo. En ambientes hostiles, la fijación y la paz mental son críticas. En ambientes bien comportados, la fijación de los requisitos no es necesaria.

2.5.2.2. Comunicación osmótica

La comunicación osmótica significa que en los flujos de información se deben de oír a los miembros del equipo, se toma la información relevante por ósmosis. Esto se logra normalmente reuniéndolos a todos en el mismo salón, de modo que cuando una persona hace una pregunta, otros en el cuarto pueden estar de acuerdo o estar en desacuerdo, contribuyendo a la discusión o continuando con su trabajo. Cuando la comunicación osmótica está presente,

las preguntas y las respuestas fluyen naturalmente y asombrosamente hay poca molestia por parte del equipo.

Una de las razones por las que puede funcionar la comunicación osmótica en los proyectos de Crystal Clear es que con una estructura muy pequeña la gente se oye por casualidad. Con la comunicación osmótica, el coste de comunicaciones es muy bajo, los errores se corrigen extremadamente rápido y el conocimiento se propaga rápidamente. La gente aprende los valores del proyecto, características del proyecto, programación, diseño, pruebas, manejo de nuevas herramientas y trucos. Se consigue rápidamente la sincronización en las metas y el estado del proyecto.

La comunicación osmótica es una característica de la mayoría de los proyectos ágiles. Es dominante en proyectos pequeños. Aunque tiene valor incluso para proyectos más grandes, es por supuesto, cada vez más difícil de lograr el objetivo conforme crece el tamaño del equipo. Este tipo de comunicación contiene sus propios peligros: demasiada plática y preguntas constantes al desarrollador más experto del equipo. Generalmente, el equipo se autorregula en estos dos peligros, no haciendo preguntas al azar, o más respecto al período de foco del individuo.

2.5.2.3. Mejora Reflexiva

Este punto se refiere al mejoramiento continuo por parte del equipo, enumeran en que han estado trabajando y en que no (dentro del marco del proyecto) discuten en qué pueden trabajar mejor, y después realizan esos cambios en la iteración siguiente. Es decir refleje y mejore. El equipo no tiene

que pasar mucho tiempo haciendo este trabajo - a menudo una hora cada una o dos semanas. El mecanismo de la mejora reflexiva permite que se ajusten los cambios que se encuentran en el proceso de desarrollo. Cada pocas semanas, una vez al mes, o dos veces según el periodo de incremento de la entrega, los desarrolladores se reúnen para discutir cómo están trabajando las cosas.

Éste es un taller de la reflexión, o iteración retrospectiva, como alguna gente prefiere llamarla. Este tipo de taller captura los hábitos de trabajo o convenciones que mantienen y lo que ellos desean intentar de forma diferente en la próxima iteración del trabajo.

En una reunión de un grupo de usuarios de Crystal Clear la gente debe discutir cómo ellos desarrollaron sus convenciones de trabajo, qué habían experimentado con estos y cómo se sentían sobre esos experimentos.

2.5.2.4. Acceso fácil a los usuarios expertos

El acceso continuo a los usuarios expertos provee al equipo:

- Requisitos actualizados.
- Retroalimentación rápida en sus decisiones del diseño.
- Retroalimentación rápida en la calidad del producto acabado.
- Un lugar para implantar y para probar las entregas frecuentes.

Los tres métodos más comunes de acceso a los usuarios son:

- Uno o más usuarios experimentados relacionados directamente con el equipo de desarrollo. Esto no es muy posible, pero no se descarta la posibilidad. Funciona periódicamente a través del equipo de proyecto que se establece directamente en la comunidad de usuario, o de cierta manera colocarse con el usuario experto. Estos equipos tienden a ser muy exitosos.

- Reuniones y entrevistas semanales o casi semanales con el usuario. Esto es lo más parecido a una norma en la metodología. Es probable que el usuario sobrecargue al equipo con la información en las primeras semanas; los desarrolladores no pueden continuar con la información que es dada de esta forma. Al poco tiempo, los diseñadores necesitan menos tiempo de los usuarios, mientras que desarrollan el código. Eventualmente, como el usuario vaya proporcionando los nuevos requisitos y la información se revisa el *software* hecho. Este ritmo podría tomar una, dos o tres horas a la semana del usuario experto.

- Enviar a los desarrolladores para hacer que los usuarios los capaciten por un período. Aunque esto puede sonar ilógico, algunos equipos de desarrollo han decidido enviar a sus programadores con cualquier usuario experto para volverlos sus aprendices. Los desarrolladores vuelven con una mejor apreciación del trabajo y un respeto por los usuarios, además de cómo su nuevo *software* puede cambiar las vidas laborales de los usuarios.

El acceso fácil a los usuarios expertos proporciona una red de seguridad para el equipo así como ser una ventaja competitiva. Es probable que esto sea un factor crítico del éxito para un equipo pequeño.

2.5.2.5. Pruebas automatizadas

Muchos equipos de proyectos entregan sistemas en los que se emplean absolutamente bien las pruebas manuales que se consideraban un factor crítico para el éxito del sistema. Sin embargo, hay algo que destaca sobre la automatización de pruebas y es que cada programador que ha tenido que utilizar una herramienta para automatizar las pruebas ha jurado nunca trabajar sin una de estas herramientas otra vez.

El hecho es que las pruebas automatizadas son una parte importante para la paz mental ya que rara vez presenta algún problema la automatización. Las pruebas automatizadas significan que la persona puede comenzar a correr las pruebas, no teniendo que mirar las pantallas, y regresar para encontrar los resultados de las pruebas.

Si cada programador y probador crean individualmente un sistema de pruebas automatizadas, después pueden escribir un archivo para poner las pruebas juntas, y se obtiene un sistema más grande de pruebas automatizadas. Incluso si el sistema combinado de pruebas tiene que funcionar todo el fin de semana, los resultados de las pruebas estarán esperando la mañana de lunes. Las pruebas automatizadas de este tipo pueden ser fijas en la integración y construir un *script*, de modo que todas funcionen en cada estructura del sistema, y los resultados pueden ser enviados por correo o publicados en una página Web para que todos vean.

Hay un punto relacionado con la prueba automatizada: cuándo escribir la prueba. Crystal Clear no asigna ninguna regla para esto. Tradicionalmente, los programadores y los probadores escriben las pruebas después de que se escribe el código, esto está muy bien, si lo hacen. Aunque, no tienen mucha

energía para escribir pruebas después de que escribe el código, en el caso de los programadores, resulta muy útil en este caso imitar a XP ya que se escriben las pruebas antes del código y se evita la corrupción de las mismas.

La mejor manera para comenzar con las pruebas automatizadas es descargar una copia específica del lenguaje que seria <lenguaje>unit (designado a menudo *unit). Hay JUnit para los programadores de Java, CppUnit para los programadores de C++, etcétera. Hay versiones * de la unidad para C, VB, y muchos otros lenguajes.

2.5.2.6. Integración frecuente y configuración administrativa

Relacionado con la configuración de la gerencia, los equipos más eficaces integran su código varias veces al día. Esto permite que los desarrolladores permanezcan más de cerca en la sincronización uno con el otro, la parte de las comunicaciones cercanas y la retroalimentación rápida es esencial en el desarrollo ágil. Integrar cada pocos días es también aceptable, pero los equipos divulgan comunicaciones mejoradas uno con otro mientras que el período de la integración disminuye.

La configuración de la gerencia y la prueba automatizada van de acuerdo al supuesto proyecto ágil. Con esas dos características, el equipo puede agregar, revisar y quitar código, incorporando requisitos que cambian sin aviso. No teniéndolas en un lugar específico (particularmente las pruebas), el equipo tendrá siempre un trabajo adicional para ocuparse con los cambios rápidos según la base del código.

Con la configuración de la gerencia disponible, y la prueba preferiblemente también automatizada, el equipo puede integrar su sistema tan a menudo como lo deseen.

Existen equipos que realizan integración continua, funcionando con código automatizado basados en construir y probar de forma directa desde un servidor independiente.

Hay que probar con diversas frecuencias de integración, y encontrar el paso de trabajo ideal para el equipo. Es necesario incluir este punto como parte de la mejora reflexiva.

2.5.2.7. Seguridad

Los problemas de seguridad inician cuando se habla de algo que le está incomodando, si se le dice al administrador que el horario es poco realista, diciéndole a un colega que su diseño (o el código) necesita una mejora, o iguala a veces diciendo a un colega que necesita tomar una ducha más a menudo o sea que no se tiene paz mental. La seguridad es importante porque con ella, el equipo puede descubrir y reparar sus debilidades. Sin ella, no podemos crecer, y las debilidades continuarán dañando al equipo.

La seguridad se construye sobre cierto nivel de la confianza, que implica el dar el poder a otra persona sobre usted, con un riesgo de daño personal. La confianza es el grado en el que uno se siente cómodo con dar el poder a otra persona. Alguna gente confía en otras por inercia, esperando que no pase nada antes de quitar la confianza. Otros están poco dispuestos a confiar en otros, esperando hasta que den evidencia de que no serán lastimadas antes de que

les den la confianza. La presencia de la confianza se correlaciona positivamente con el funcionamiento del equipo.

Las maneras en las cuales una persona puede ser lastimada producen diversas formas de confianza: alguien puede ser que ensucie su asignación, debido a la carencia de la capacidad o de la confiabilidad. Esta persona puede ser que actúe para lastimar, puede ser que mienta, encubra la información, ande divulgando información privada del proyecto, etc. Aceptar la exposición a uno de éstos riesgos es una forma de confianza.

Es importante que la gente pueda hablar y actuar libremente. Se revelará la información en una forma más común, lo cual apresurará el descubrimiento de defectos en el proyecto. Por lo tanto, la seguridad es la característica crítica a lograr.

Una vez que se establezca la seguridad, un tipo de alegría puede emerger. La gente puede emprender la competición amistosa uno con otro. Pueden discutir en alta voz, incluso al borde de luchar, sin tomarlo en forma personal. En el caso donde alguien lo toma en forma personal, lo clasifican hacia fuera y fijan cosas rectas otra vez. Durante las sesiones reflexivas de mejora, la gente está más dispuesta a hablar de su corazón, o de las opiniones, de modo que otras personas puedan ampliarlas.

La seguridad no debe ser confundida con la cortesía. Algunos equipos pueden aparentar tener seguridad pero en realidad están siendo corteses entre sí, porque son pocos los que están dispuestos a demostrar desacuerdo. Cubriendo sus desacuerdos con cortesía y la conciliación, no detectan y no reparan los errores que están presentes, y esto puede llegar al extremo de dañar el proyecto.

2.5.3. Caso

Una empresa se dedica a realizar boletines de carácter político uno se publica mensualmente y otro en forma quincenal, los dos por el medio escrito desde hace ocho años y desde hace tres años lo hacen por Internet. Estas publicaciones se difunden en todo el país y toda Centroamérica y parte de México. Lo que la empresa busca es distribuir la información que obtienen de distintas fuentes a todos sus suscriptores, sin tener en ningún momento relación con algún partido político.

La empresa tiene contratado un servicio de Host en el extranjero (Florida, USA), esta empresa se ha encargado de diseñar, crear y mantener el portal. Cada vez la empresa ha tenido más problemas técnicos, ya que les resulta difícil realizar las actualizaciones de las páginas con los nuevos documentos editados, este problema afecta la imagen de la empresa y además presenta un impacto económico en las ganancias de la empresa periodística.

Requerimientos

Por todo esto la empresa ha decidido tener su propio servidor y crear su propio departamento de desarrollo el cual tendrá a cargo las mismas responsabilidades que los subcontratados (diseñar, crear y mantener el sitio).

Dado que la empresa ha decidido hacerse cargo de todo, los nuevos empleados tienen que administrar el servidor que contendrá el portal, esto incluye la configuración del mismo, además del diseño y la creación de las páginas para la empresa.

Al momento de desligarse por completo de la empresa subcontratada el nuevo sitio deberá de estar listo, por lo menos las opciones más importantes. Algo importante que destaca es que todo se comenzará desde cero y no se contará con el código de las páginas actuales ni cualquier tipo de componentes, plantillas, etc. Ya que el contrato que tiene la empresa periodística con la empresa de *host* no les otorga ningún derecho sobre lo antes mencionado.

Anteriormente cuando se deseaba subir un documento éste tenía que ser enviado al *Webmaster* y después él se hacía cargo se subir el documento al sitio, este es uno de los procesos que se desea mejorar. Lo que se pretende ahora es que los usuarios autorizados puedan subir sus documentos desde cualquier lugar y automáticamente aparezcan en el sitio.

2.5.3.1. Descripción de la solución

Se contratará un equipo de cuatro personas (un administrador, un técnico y dos programadores) para la realización del proyecto. Se contratará un servicio de conexión a Internet de un ISP local.

La solución se realizará utilizando páginas dinámicas JSP (Java Server Pages) ya que esta tecnología permite modificar las páginas en tiempo de corrida. Esto se debe a que una vez que un usuario sube un documento éste se tiene que visualizar en la página de documentos cargados, y la tecnología java también permite de forma fácil poder implementar la parte de subir y descargar archivos (protocolo FTP: File Transfer Protocol).

Dado el tipo de páginas que se utilizarán, el Web Server que se manejará será Macromedia JRun Server, se realizará la conexión con la base de datos por JDBC (Java Data Base Connection), este manejador ya está incluido en

Java 2 SDK (Standar Development Kit) que será la versión con la que estarán trabajando los desarrolladores.

Se manejará la base de datos con MySQL, para llevar el control de los usuarios de la empresa, el listado de los documentos que se poseen y el control de las distintas características que sean necesarias.

Dado que el tamaño de cada documento no es muy grande, semanalmente se realizará una copia de seguridad de todos los documentos los cuales se almacenarán en CD's, para así protegerse de cualquier problema que pudiese ocurrir con la pérdida de alguna información.

2.5.3.2. ¿Por qué Crystal Clear?

Una de las razones por la que se puede implementar Crystal Clear es el tamaño de equipo de desarrollo el cual es relativamente pequeño y con esto resulta fácil la utilización de la comunicación osmótica, también se da la oportunidad de tener interacciones frente a frente entre los desarrolladores y los clientes (en este caso los dos están en el mismo lugar), y por lo tanto se obtiene una rápida depuración del trabajo.

El acceso a los expertos es casi inmediato por la situación en que se encuentran los equipos, con esto se consigue una retroalimentación bastante rápida con requisitos actualizados. Ya que el proyecto es para la Web, da la ventaja de que los desarrolladores estén más sincronizados entre ellos, se integra en el tiempo más corto posible. El punto de Crystal Clear es permitir a los miembros de los equipos que utilicen sus mejores técnicas de trabajo, y no atarlos a un tipo de trabajo con el cual no están a gusto.

2.6. FDD (Feature Driven Development)

2.6.1. ¿Qué es FDD?

El Desarrollo Manejado por Rasgos, fue desarrollado por Jeff De Luca y Peter Coad. Como las otras metodologías adaptables, se enfoca en iteraciones cortas que entregan funcionalidad tangible. En el caso del FDD las iteraciones duran dos semanas como máximo.

Los desarrolladores entran en dos tipos: dueños de clases y programadores jefe. Los programadores jefe son los desarrolladores más experimentados. A ellos se les asignan los distintos rasgos a construir, sin embargo ellos no los construyen solos. Sólo identifican qué clases se involucran en la implantación de un rasgo y juntan a los dueños de dichas clases para que formen un equipo para desarrollar ese rasgo. El programador jefe actúa como el coordinador, diseñador líder y mentor mientras los dueños de clases hacen la mayor parte de la codificación del rasgo.

FDD es extremadamente efectivo en proyectos largos con una lógica del negocio compleja. Y no es efectiva en proyectos pequeños y con una simple lógica del negocio. Esto no significa que no se deba usar FDD para ayudar a la administración para hacer un proyecto de pequeña escala pero es mejor dando un soporte fuerte al código y a las revisiones de diseño

La clave para FDD es el primer proceso, y la clave para el primer proceso es tener a un gran modelador de objetos y un gran gerente de proyectos. No importa que proceso use, es necesario tener a las personas correctas en todos los sentidos, ni la mejor metodología puede ayudar cuando no se tiene a las personas ideales.

2.6.2. El punto de inicio

Existen distintas formas de empezar un proyecto con FDD, pero la base siempre es la misma, lo primero es pensar que el proyecto es un conjunto de rasgos, a partir de esto se generará todo el proyecto.

La definición del proyecto es una lista de rasgos o funciones

Uno de los puntos generales, en cada proyecto es necesario definir qué hace o la funcionalidad que provee. Esto se hace listando todas las funciones (o rasgos) del sistema, con esto se está definiendo que es lo que se requiere para hacer todo el proyecto. Una de las cosas que hay que tomar en cuenta para realizar esta parte es usar el mismo lenguaje que el cliente, no se deben de usar términos técnicos debido a que no los entienden ni tampoco les importan.

Plan de desarrollo basado en rasgos

Una vez que se sabe qué es lo que tiene que construir, debe de seguir un orden lógico lo cual es hacer el plan de desarrollo, en otras palabras ya sabiendo qué es lo que se debe de hacer hay que pensar cómo se debe de hacer. Puede haber algunos rasgos que dependan de otros, estos obviamente son los que se hacen de primero. Las tareas que sean paralelas se pueden hacer en cualquier orden, se sabe que se tiene un plan de desarrollo que se seguirá paso a paso durante el progreso del proyecto.

Reuniones semanales del estado del proyecto por rasgos completados

Si se tiene una lista de lo que se necesita tener hecho y cuando tenerlo, es más fácil seguir el progreso del proyecto y realizar el reporte de avances para el cliente. Es muy efectivo decirle al cliente que rasgos han sido

completados y en cuales se esta trabajando actualmente, preferiblemente que sea cada semana.

2.6.3. Los Procesos

El FDD tiene cinco procesos. Los primeros tres se hacen al principio del proyecto. Los últimos dos se hacen en cada iteración. Cada proceso se divide en tareas y se da un criterio de comprobación. Cada uno de estos procesos se compone de: criterio de entrada, tareas, verificación y criterio de salida, esto ayuda a tener el proyecto bajo control con un proceso perfectamente sistematizado[5].

Cada tarea lleva un conjunto de personas que desarrollan distintas actividades dependiendo del rol que tengan dentro del proyecto. La mayoría de estas actividades son obligatorias ya que nos ayudan a llevar el control del desarrollo del proyecto.

2.6.3.1. Desarrollando un modelo global

Las actividades iniciales de un proyecto son definir a los miembros bajo la dirección de un modelador con experiencia que será el Jefe de Arquitectura.

Se forman pequeños equipos mezclando desarrolladores y expertos del área, cada uno de éstos forma su propio modelo que debe soportar el problema propuesto. Cada equipo presenta su modelo para que sea visto y discutido, uno de los modelos propuestos, o combinación de los modelos, se selecciona según el acuerdo general así que se hace el modelo para aquella área de dominio específico.

[5] Que esta ordenado siguiendo una estructura lógica.

Criterio de entrada

El experto del área (pertenece a los clientes), programadores jefe y el jefe de arquitectura han sido seleccionados.

Tareas

Formación del equipo – Administrador de proyecto - Requerido

La formación del equipo compromete en forma permanente a los que serán miembros de éste, no importando si son pertenecientes a los clientes o del grupo de desarrolladores, básicamente son los expertos del área y el jefe de programadores. Otros equipos de proyectos son rotados por medio de las sesiones de modelado para que así cada uno tenga la oportunidad de participar y ver el proceso en acción.

Dominio del entorno – Equipo de modelado – Requerido

Un experto del área da una visón global de la parte que será modelada. Esto también debe de incluir información como características principales y actores, no es necesario realizar toda la implementación.

Estudio de documentos – Equipo de modelado - Opcional

El equipo estudia la referencia de las posibilidades o los documentos de requerimientos semejante al modelado de objetos, requerimientos funcionales (por casos de uso), modelo de datos y guías de usuarios.

Desarrollo del modelo – Equipo de modelado - Requerido

Se forman grupos dividiendo el equipo de modelado en no más de tres integrantes, cada pequeño grupo compondrá un modelo en apoyo del área en

cuestión. Un miembro de cada equipo presenta al grupo el modelo propuesto para el área en la que trabaja.

Refinar el modelo global de objetos – Jefe de arquitectura y equipo de modelado – Requerido

Cada cierto tiempo, el modelo global de objetos es actualizado con las figuras del nuevo modelo producidas por las iteraciones del desarrollo del modelo.

Notas del modelo – Jefe de arquitectura y programadores jefe – Requerido

Se escriben notas en detalle, figuras del modelo y alternativas significantes para tener referencias del proyecto en el futuro.

Verificación

Valoración interna y externa – Equipo de modelado y negocios - Requerido

La autovaloración o valoración interna es alcanzada por la participación activa de los expertos del área. Una de las necesidades básicas, es hacer una evaluación externa para referirse al negocio de los usuarios para ratificar o aclarar las cosas que pueden afectar al modelo.

Criterio de salida

Los resultados del proceso es el modelo de objetos. Modelo de clases (métodos y atributos), diagramas de secuencias (sí en caso hay) y el modelo de notas

2.6.3.2. Construir una lista de rasgos

Una actividad inicial para todo el proyecto es identificar todos los rasgos para apoyar las exigencias. El equipo comprende las actividades del negocio y los pasos dentro de cada una de las actividades del negocio, formando así la lista de rasgos categorizada.

La clasificación de más alto nivel para la lista de rasgos proviene de la división del dominio realizada por los expertos del área en el primer proceso.

Criterio de entrada

El experto del área o domino, los programadores jefe y el jefe de arquitectura han sido seleccionados.

Tareas

Formar el equipo para la lista de rasgos – Administrador de proyecto y administrador de desarrollo – Requerido

El equipo esta comprendido principalmente por los programadores más importantes del equipo de modelación del proceso anterior.

Construir la lista de rasgos – Equipo de lista de rasgos - Requerido

El equipo identificará el conjunto de rasgos que usarán a partir del conocimiento obtenido previamente. Ésta es simple, la descomposición funcional en los ámbitos que viene de la división de dominio por dominio.

Los rasgos son funciones granulares expresadas en términos usados por el cliente. Por ejemplo, calcular el total de una venta, calcular la cantidad total vendida por un punto de venta al por menor para una descripción de un artículo.

Los rasgos son granulares conforme a la regla en la cual un rasgo no tomará más de dos semanas para ser completado, dos semanas son el límite

superior por lo regular los rasgos toman menos tiempo. Cuando una actividad económica tiende a ser más grande que dos semanas, la actividad es dividida en pequeños pasos que entonces se convierten en rasgos.

Verificación

Valoración interna y externa – Equipo de lista de rasgos y negocios - Requerido

La autovaloración interna es alcanzada por la participación activa de los miembros de equipo de modelado. Una de las necesidades básicas, es hacer una evaluación externa para referirse al negocio de los usuarios para ratificar o aclarar las cosas que pueden afectar la lista de rasgos.

Criterio de Salida

El resultado del proceso es la lista de rasgos. Para cada ámbito, una lista de las actividades del negocio y para cada actividad del negocio, el rasgo para satisfacer esa actividad.

2.6.3.3. Planeación por Rasgos

El jefe del proyecto, el gerente de desarrollo y los programadores principales (jefes) planean el orden en el cual los rasgos van a ser implementados, basados en las dependencias que posean unos con otros, también por medio del equipo de desarrollo se decidirá sobre la complejidad que posean los rasgos para que puedan ser implementados.

Las tareas principales en este proceso no son una secuencia estricta, como muchas actividades de planificación éstas son consideradas en conjunto, en un escenario típico se debe considerar la secuencia de desarrollo, luego considerar la asignación de actividades de negocio a los programadores y

hacerlo así, considere qué clases deben ser asignadas a qué desarrolladores y sólo a éstos.

Cuando este equilibrio es alcanzado en la secuencia de desarrollo y la asignación de actividades de negocio entonces la propiedad de clase esta finalizada.

Criterio de entrada

El proceso de construcción de la lista de rasgos ha sido terminado.

Tareas

El equipo de planeación – Administrador del proyecto - Requerido

La planeación del equipo comprende al gerente de desarrollo y a los programadores

Determinar la secuencia de desarrollo – Equipo de planificación – Requerido

El equipo de planificación asignará una fecha (el mes y el año) para la finalización de cada actividad del negocio. La identificación de las actividades del negocio y la fecha de conclusión (y la secuencia de desarrollo) están basadas en:

- Dependencia entre los rasgos en términos de las clases relacionadas.
- Equilibrio de carga a través de los responsables de las clases.
- La complejidad en el rasgo a implementar.
- Presentación de las actividades de alto riesgo o complejas del negocio.

- Consideración de cualquier problema externo (visible) por medio de prototipos, puntos de retroalimentación y productos enteros que satisfacen tales problemas.

Asignar las actividades del negocio a los programadores – Equipo de planificación - Requerido

El equipo de planificación asignará a los programadores como los responsables de las actividades de negocio. La asignación está basada sobre:

- La secuencia de desarrollo.
- Dependencia entre rasgos en términos de las clases involucradas.
- Equilibrio de carga a través de los responsables de las clases
- La complejidad de los rasgos que van a ser implementados.

Asignación de clases a los desarrolladores - Equipo de planificación - Requerido

El equipo de planificación asignará a los desarrolladores como responsables de las clases, los desarrolladores pueden tener múltiples clases asignadas. La asignación de las clases a los desarrolladores está basada en:

- Equilibrio de carga a través de los desarrolladores.
- La complejidad de las clases.
- El uso de las clases (por ejemplo empleo alto).
- La secuencia de desarrollo.

Verificación

Autovaloración – Equipo de planeación - Requerido

Como la planificación es una actividad de equipo, una autovaloración es alcanzada por la participación activa de los programadores, el gerente de desarrollo y el jefe del proyecto.

Criterio de salida

- Actividades del negocio con fechas de terminación (mes y año).
- Programadores asignados a las actividades del negocio.
- Ámbitos con fechas de terminación (mes y año) sacados de la última fecha de terminación de sus actividades respectivas de negocio.
- La lista de clases y los desarrolladores que las poseen (la lista de dueños de clases)

2.6.3.4. Diseñar por Rasgos

Un número de rasgos son calendarizados para ser desarrollados, esto está a cargo de un programador jefe. El programador jefe selecciona los rasgos para el desarrollo de su "bandeja de entrada" de rasgos asignados. Él puede escoger múltiples rasgos que resultan usar las mismas clases,

El programador jefe entonces forma un equipo de rasgo por la identificación de los dueños de las clases (desarrolladores), este equipo entonces produce diagramas de secuencias para los rasgos asignados. El programador jefe entonces refina el Modelo de Objeto basado en el contenido de los diagramas de secuencias. Los desarrolladores escriben la clase y los métodos, además se debe de mantener una inspección constante en el diseño.

Criterio de entrada

El proceso de planeación esta completo.

Tareas

El equipo de rasgos – Programador jefe - Requerido

El programador jefe identifica las clases que probablemente van ha estar implicadas en el diseño de este conjunto de rasgos, en consecuencia actualiza la base de datos de rasgo. De la lista de los responsables de las clases, el programador jefe identifica a los desarrolladores que formarán el equipo de rasgos. Como la parte de este paso, el jefe crea un nuevo paquete de diseño para el rasgo como parte del paquete de trabajo.

Estudiar los documentos referidos – Equipo de rasgos - Opcional

El equipo de rasgos estudia el documento referido para el rasgo que va a ser diseñado, todas las notas de confirmación, diseño de pantallas, datos específicos de la interfaz del sistema externo y cualquier otra documentación de apoyo. Ésta es una tarea opcional basada en la complejidad del rasgo y/o sus interacciones.

Desarrollo de los diagramas de secuencias – Equipo planificación - Requerido

El desarrollo de diagramas de secuencias es requerido para el rasgo que va a ser diseñado. Los archivos de los diagramas deberían ser comprobados en el sistema de control de la versión. Cualquier diseño alternativo, diseño de decisiones, clasificación de requerimientos y apuntes también son registrados y escritos encima del diseño de alternativas en la sección del diseño del paquete.

Refinar el modelo de objetos – Programador jefe - Requerido

El programador jefe crea un equipo de rasgos de área para el rasgo en cuestión. El objetivo del equipo de rasgos de área es el progreso en el trabajo

del equipo de rasgo que puede ser compartido y es visible entre el equipo de rasgo pero no es visible al resto del proyecto.

El programador jefe refina el modelo y añade clases, métodos, atributos y/o hace cambios a las clases existentes, esto lo hace basado en el diagrama de secuencia definido para el rasgo. Esto a causa de los archivos fuentes en el lenguaje que se esté trabajando, actualizados por el equipo de rasgos de área. El jefe crea diagramas en un formato de estándar para todos, estos archivos deberían de ser comprobados en el sistema de control de versión.

Escribir clases y prólogos de métodos – Equipo de rasgos - Requerido

La utilización de los archivos fuentes del lenguaje utilizado son actualizados por la refinación del modelo de objetos, la tarea en el equipo de rasgo de área es compartida, el responsable del desarrollo de cada clase escribe la clase y prólogos de los métodos para cada detalle definido por el rasgo y diagramas de secuencia. Esto incluye tipos de parámetros, tipos de ciclos, excepciones y mensajes. Una vez que cada desarrollador ha completado esta tarea, el jefe genera el API y la documentación.

Verificación

Inspección de diseño – Equipo de rasgos – Requerido

Se sostiene una inspección de diseño con los miembros del equipo de rasgos o con otros miembros de proyecto. La decisión de que se haga la inspección dentro del equipo de rasgos o con otros miembros del equipo de proyecto es del programador jefe. Cada miembro de equipo añade a su lista de tareas la nueva tarea calendarizada, las clases se catalogan por los cambios hechos a las mismas.

Criterio de salida

El resultado del proceso es un paquete de diseño satisfactoriamente inspeccionado. El paquete de diseño comprende:

Una nota de cubierta, o papel, que se integra y en el que se describe el diseño tal y como está al día en las propias revisiones. Los requerimientos referidos (sí existiera alguno) en forma de documentos y todas las notas de confirmación relacionados a la documentación de apoyo.

- Los diagramas de secuencias.
- Las alternativas de diseño (sí existiera alguna)
- El modelo de objeto con las nuevas actualizaciones para las clases, métodos y atributos.
- Calendarización de tareas por hacer según la lista de tareas para los detalles de acciones sobre clases afectadas para cada miembro de equipo.

2.6.3.5. Construyendo por rasgos

Comenzando con el paquete de diseño, los responsables de las clases de desarrollo ponen en práctica los detalles necesarios para su clase y así apoyar el diseño para este rasgo. El código desarrollado es la unidad probada y el código inspeccionado – el orden de trabajo de cual debe de ser concluida es determinado por el jefe programador. Después de una inspección de código acertada, el código se pasa a la parte de construcción.

Criterio de Entrada

El diseño rasgos de procesos ha sido completado. Y el diseño del paquete ha sido inspeccionado exitosamente.

Tareas

Implementando las clases y métodos – Equipo de rasgos - Requerido

El desarrollo de la clase por su responsable es implementado según las características necesarias para satisfacer los requisitos necesarios del rasgo al que pertenece.

Inspección de código – Equipo de rasgos – Requerido

La inspección de código por los miembros del equipo de rasgos o por otro miembro del proyecto es mantenida antes o después de las tareas de unidad de prueba. La decisión de hacer la inspección por el equipo de rasgos o con otro miembro del equipo es del jefe programador, éste también toma la decisión de hacer la inspección antes o después de la unidad de pruebas.

Unidad de pruebas – Equipo de rasgos - Requerido

El desarrollo de la clase por su responsable o propietario prueba su código y se debe de asegurar que cumple con todos los requerimientos para satisfacer la clase.

Promover la construcción – Programador jefe y equipo de rasgos - Requerido

Las clases sólo pueden ser promovidas a construcción después de la inspección exitosa del código. El jefe programador sigue en forma individual las clases que han sido promovidas, y así obtener la retroalimentación de los desarrolladores, y este es el punto de integración para el rasgo completo.

Verificación

Inspección de código y Unidad de Pruebas – Programador Jefe y Equipo de Rasgos - Requerido

Una inspección de código exitosa más la completación exitosa de la unidad de pruebas es la verificación de la salida de este proceso.

Criterio de Salida

Los resultados de este proceso son:

- Clases o métodos con la inspección de código exitosa.
- Clases que son promovidas a la construcción.
- La completación de una función (rasgo) de valor para el cliente.

2.6.4. Caso

Una empresa se dedica a la creación de proyectos de animación por computadora (principalmente en 3D); han trabajado en la realización de comerciales para la televisión, cortos animados, videos de toda clase, etc. Además de esto también tienen una amplia experiencia en la edición de películas mezclando tomas reales con las imágenes generadas por computadora.

La empresa es reconocida internacionalmente por su amplia experiencia en el ramo, y por su alta calidad en el trabajo que realizan. A la empresa en cuestión se le ha contratado para que realicen toda la animación, integración y edición primaria de un documental de vida salvaje en la época prehistórica. Para la obtención de este importante contrato debieron someterse a un proceso de selección con otras empresas internacionales.

Requerimientos generales

Dado el tipo de trabajo que se esta realizando (documental prehistórico), entre las solicitudes se encuentra la creación de criaturas extinguidas, climas extremos (días con tormentas, soleados, nublados, etc.), escenarios con gran vegetación, escenarios desérticos, etc.

Actualmente la empresa no cuenta con personal suficiente para desarrollar el proyecto en el tiempo establecido que es de aproximadamente 7 meses, para lo cual decidieron subcontratar a 3 empresas con las que ya han trabajado.

La primera empresa localizada en Australia tendrá a cargo el desarrollo de los escenarios, de los cuales 7 deben de ser planos (solo se utilizarán como fondo) y 15 deben de ser en 3D ya que se debe de poder moverse dentro de ellos, algunos más grandes que otros.

Cada uno de estos se realizará según las especificaciones dadas por la empresa contratadora, los escenarios van desde un simple follaje, áreas rocosas, riachuelos, cuevas y cascadas. Los 15 escenarios deben de poseer una animación altamente natural como sombras, movimiento de las hojas, texturas de los objetos, etc.

La segunda empresa ubicada en Japón tiene a cargo el desarrollo de los climas: días soleados totalmente despejados, creación de 4 distintos tipos de cielo con nubes los cuales deben de poseer movimiento, 2 crepúsculos, 3 amaneceres estos dos últimos han de durar de 5 a 10 minutos.

Además deben de crear la animación de una tormenta la cual desde su inicio hasta su punto más fuerte debe de ser de 5 minutos con 12 segundos.

Cada uno de los climas se unirá a los distintos escenarios creados por la otra empresa, por lo que todos trabajarán con un mismo estándar.

La tercera empresa con sede en Inglaterra tiene a cargo el desarrollo de 57 modelos animales, los cuales deben de estar listos para su utilización. Entre los cuales habrá 17 clases distintas de animales, algunos modelos sólo variarán en color, altura, complexión, etc. Los modelos serán aves, mamíferos y reptiles, cabe destacar que éstos deben de contar con una gran calidad en sus texturas y la animación para poder obtener el mayor realismo en el documental.

2.6.4.1. Descripción de la solución

La empresa contratada directamente entregará distintos cortos con las exigencias estipulan para el edición final. Pero antes de esto deben de integrar todo el trabajo generado por sus empresas subcontratadas.

Debido al alto poder de procesamiento que se necesita para la integración de cada una de las partes se creará un sistema de cluster de 7 máquinas de última generación para poder laborar de una forma ideal dentro del proyecto.

Todos los nodos del cluster tendrán un sistema operativo Linux, y todas las empresas trabajarán con el programa de animación Light Wave, el cual provee de todos los elementos para la creación de este tipo de proyecto creación de texturas, modelos, escenarios, etc.

Se realizarán reuniones periódicas para revisar el avance del trabajo de cada una de las empresas subcontratadas y discutir cualquier cuestión que pueda surgir, esto se hará por medio de video conferencias cada viernes de la

semana a las cero horas del meridiano, se creará un canal de mensajería instantánea el cual estará activo todo el día para poder responder cualquier comentario o cuestión que pueda surgir en determinado momento.

Para compartir archivos la empresa contratada ha decidido contratar un servidor de host que tenga soporte para FTP para que se tenga acceso a los archivos, ya que algunos son demasiado grandes como para que se puedan enviar por correo. La información debe de viajar encriptada y para poder acceder al servidor las empresas deben de tener un usuario y una clave.

2.6.4.2. ¿Por qué FDD?

Dada la complejidad del proyecto y la gran duración que tiene, en comparación con muchos proyectos se crea un ambiente ideal para trabajar con esta metodología. La empresa contratada directamente posee la experiencia suficiente para conseguir la coordinación de las otras empresas, ya que cuenta con personal altamente calificado con muchos años en el medio.

Cada una de las solicitudes se agregará en una lista para llevar el control de las actividades a realizar. Lo primero que se debe de tener es el trabajo que genera la primera empresa ya que a partir de los escenarios se comenzarán a adaptar el resto de los elementos, las otras dos empresas desarrollarán en paralelo.

El criterio de comprobación para cada una de las tareas la dará la empresa contratada, otro punto que apoya a la utilización de esta metodología es que cada empresa se tratará como un equipo, no se controlará la estructura interna que posee cada una de estas empresas.

La estructura con la que trabajarán será la misma para todos características como sistema métrico decimal, el programa de desarrollo, etc. Conforme se va dando el avance se estará refinando la estructura del proyecto, las iteraciones están definidas por un conjunto de escenarios, ambientes y modelos que deben de ser entregados por todas las empresas.

La empresa contratada planea el orden en el cual los rasgos van a ser implementados, basados en las dependencias que poseen uno con otros, las empresas subcontratadas determinarán la complejidad del trabajo para estimar el tiempo de entrega, después de esto se calendarizarán los rasgos.

La única diferencia real que posee este tipo de proyectos a los de otros proyectos de desarrollo comunes es la parte de mantenimiento.

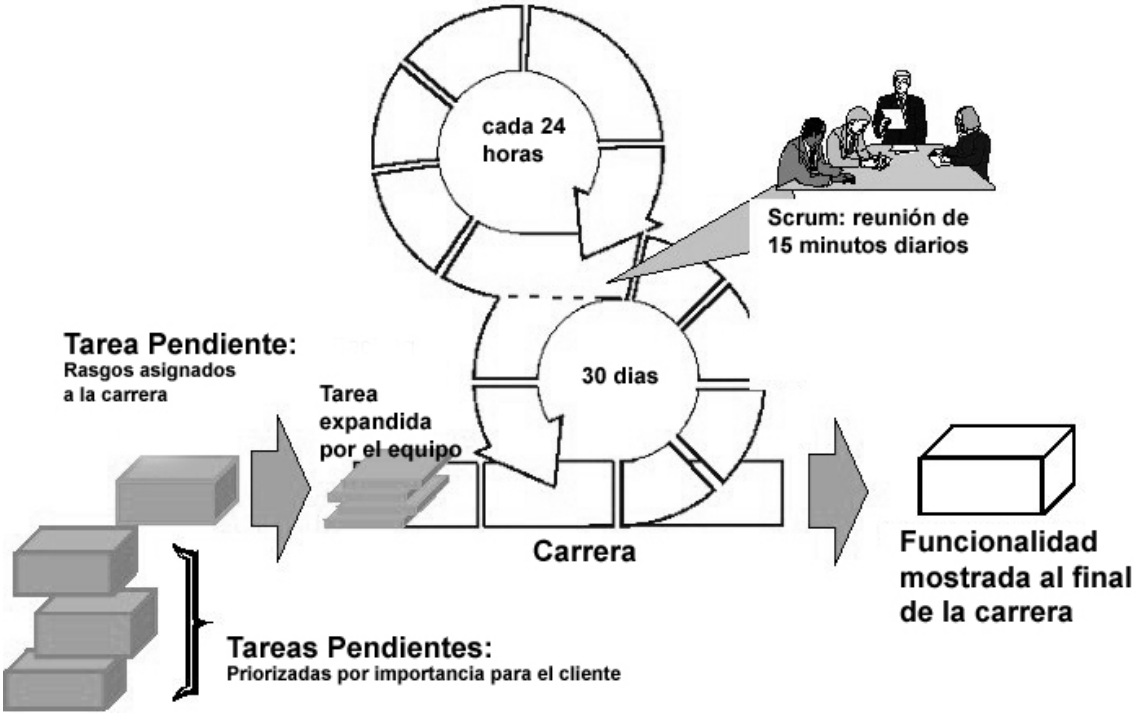
2.7. Scrum

Scrum ha estado durante algún tiempo en los círculos orientados a objetos. Ken Schwaber y Mike Beedle escribieron el primer libro de Scrum, pero Jeff Sutherland fue la primera persona en aplicar sus conceptos. De nuevo se enfoca en el hecho de que procesos definidos y repetibles sólo funcionan para atacar problemas definidos y repetibles con gente definida y repetible en ambientes definidos y repetibles.

Scrum divide un proyecto en iteraciones (a las que se les llaman carreras cortas) de 30 días. Antes de que comience una carrera se define la funcionalidad requerida para esa carrera y entonces se deja al equipo para que la entregue. El punto es estabilizar los requisitos durante la carrera.

Sin embargo la gerencia no se desentiende durante la carrera corta, todos los días el equipo sostiene una junta corta (quince minutos), llamada Scrum, dónde el equipo discute lo que hará al día siguiente. En particular muestran a los bloques de la gerencia: los impedimentos para progresar que se atraviesan y que la gerencia debe resolver. También informan lo que se ha hecho para que la gerencia tenga una actualización diaria de dónde va el proyecto.

Figura 7. Flujo de Actividades de Scrum



La literatura de Scrum se enfoca principalmente en la planeación iterativa y el seguimiento del proceso. Es muy cercana a las otras metodologías ágiles en muchos aspectos y debe funcionar bien con las prácticas de código de la XP.

2.7.1. La filosofía de Scrum

El centro de Scrum es el acercamiento a la creencia de que la mayoría de los sistemas de desarrollo tienen una mala filosofía básica. La falla de los proyectos, sistemas inapropiados y herramientas de productividad poco efectivas son pruebas de que el proceso de desarrollo necesita mas rigor, si podemos hacer que todos los desarrolladores nos sigan, los problemas se irán.

Scrum declara que el proceso de desarrollo de los sistemas es un proceso impredecible y complicado que sólo se puede describir en forma global. Además define que el proceso del desarrollo de los sistemas es un conjunto de actividades separadas de las cuales tenemos cierto conocimiento, las herramientas de uso fácil y las técnicas nos ayudan a obtener un mejor desarrollo en equipo para construir los sistemas. Desde el momento en que se separan las actividades el control de cada una adopta un riesgo inherente.

Los siguientes problemas pueden ocurrir como resultado de una mala filosofía y malas técnicas de desarrollo:

- El tiempo en el que se debe de entregar el sistema, a menudo es irrelevante o requiere cambios significantes.

- La administración actualmente piensa que pueden predecir el costo, la calendarización y la funcionalidad que debe de ser entregada.

- Los desarrolladores y administradores de proyectos están forzados a vivir una mentira. Ellos tienen que pretender que pueden planear, predecir y entregar, y trabajan de la mejor forma que conocen para entregar el sistema.

Scrum se aplica a las organizaciones internas, y de esta forma puede revelar los problemas previamente mencionados. El acercamiento es llamado Scrum, y comienza aceptando que esto es complicado; también inicia con la premisa que se puede predecir o hacer el plan de entregas, cuando se haya entregado. La metodología es aplicable para sistemas nuevos o existentes que pueden usar una interfaz clara o por objetos y componentes.

2.7.2. Reglas de Scrum

Scrum

Es un proceso iterativo e incremental para desarrollar *software* en ambientes caóticos. Scrum consiste en una serie de carreras de 30 días, cada carrera produce un ejecutable. Entre las carreras, todos los interesados participan evaluando el progreso y reevaluando los requerimientos técnicos y de negocios.

El ritmo de Scrum es la clave de su éxito. La administración determina la prioridad para cada carrera, su determinación está influenciada por la prioridad de las entregas y los requerimientos. Durante la carrera, el equipo produce el mejor *software* posible: dejando el caos, manteniéndose fuera del caos, dejando el caos, manteniéndose fuera del caos, dejando el caos, manteniéndose fuera del caos, etc.

Backlog

Es una lista priorizada de todo el trabajo que tiene que ser completado y liberando el producto según la prioridad:

- Sólo una persona tiene a cargo priorizar y mantener la lista de Backlog.

- Cualquier interesado puede participar demandando que Backlog debe ser puesto en la lista.

Entre las carreras, todas las partes envueltas y el equipo de ingeniería se reúne para determinar qué trabajo debe ser completado en la próxima carrera, y como debe ser el ejecutable generado

Sprint

Un corto gran esfuerzo de trabajo duradero aproximadamente durante 30 días durante el cual un ejecutable y otras características son construidos por un equipo de ingeniería, el cual es definido por el Backlog asignado.

Las características de las carreras son;

- No debe de durar más de 30 días.

- La carrera es emprender por un cruce funcional de equipo comprendido de no más de 9 miembros.

- Cada carrera tiene una meta específica.

- Un ejecutable demostrando la meta que será completada por el equipo durante la carrera.

- Si alguna fuerza externa determina que la carrera esta trabajando erróneamente, la carrera es detenida y reiniciada con un nuevo Backlog y propósito.

- Una vez que inicia la carrera, un nuevo Backlog no puede ser agregado a la carrera excepto si, el administrador del proyecto determina que el nuevo Backlog refuerza la viabilidad del producto.

Reunión Scrum

La reunión diaria de Scrum es un nivel que verifica el conocimiento del equipo y actualizan a cada uno con respecto a lo que los demás están haciendo. Esto provee diariamente de un punto de donde debe de iniciarse el trabajo:

- Durante la carrera, el equipo maneja diariamente una reunión.

- La reunión es sostenida en el mismo lugar a la misma hora cada día de trabajo.

- La reunión no dura más de 30 minutos.

- El administrador de la reunión es definido.

- El administrador es responsable por preguntar a cada miembro del equipo las siguientes 3 preguntas:

  - ✓ Que ha hecho desde la última reunión Scrum.
  - ✓ Que ha impedido su trabajo.
  - ✓ Que tiene planeado hacer entre ahora y la próxima reunión Scrum.

- La conversación es restringida a los miembros del equipo para que respondan las preguntas.

- Las próximas reuniones pueden ser establecidas después de la reunión Scrum basada en las respuestas de las preguntas.

- El administrador es responsable por hacer las decisiones inmediatamente, si se requiere eliminar cosas que impidan el progreso.

- El administrador es responsable por hacer notar los impedimentos que deben ser resueltos externamente para la reunión y así eliminarlos.

2.7.3. Comenzando a usar Scrum

Scrum es una aplicación del sentido común al trabajo que utiliza técnicas de productividad aplicadas a la ingeniería de *software*. En la metodología se definen prioridades altas para hacer el trabajo y producir versiones.

Inicio del proceso

La definición del equipo consiste de personas que van ha estar asignadas al trabajo (desarrolladores) y personas que están interesadas, pero que no trabajan (clientes). Al momento de identificar a las personas que trabajarán se debe de recordar de lo siguiente:

- No más de 6 a 9 miembros por equipo.

- Si hay más miembros que manejar, es necesario dividirlos en equipos.

- Cada equipo se enfoca en un punto, autocontenido en el área de trabajo.

- Todos los miembros trabajan en esta área.

Definiendo al jefe de Scrum

El jefe de Scrum es la persona de conduce las reuniones de Scrum, realiza medidas empíricamente, toma decisiones. Dentro de las reuniones éste realiza el siguiente trabajo:

- Realiza las 3 preguntas expuestas en el inciso anterior a las personas involucradas.

- Debe de tener la posibilidad de hacer decisiones inmediatamente.

- Identificar los Backlog iniciales.

- Debe de resolver los problemas de trabajo independientemente de las circunstancias.

Identificando Backlog

Backlog es todo el trabajo relevante para un área del producto

- Listar el trabajo que conocemos que hay que hacer.

- Agruparlos en incrementos que no tomen más de 30 días.

- En las áreas donde el trabajo no puede ser completamente definido o puede cambiar se le darán más de 30 días, establecer un incremento para horizontes conocidos.

- Listar todo el trabajo sobresaliente que debe hacerse.

- Sólo una persona tiene a cargo la priorización de los Backlog.

- El equipo selecciona un Backlog para la carrera.

- Sólo este Backlog es trabajado durante esta carrera.

2.7.4. Caso

Una empresa se dedica a crear productos alimenticios de diverso tipo; esta empresa es líder en su ramo en el mercado de Guatemala. Hace 4 años ingresó al mercado una empresa extranjera del mismo ramo, esto le ha resultado muy difícil a la empresa nacional ya que no estaba preparada para esto.

La empresa extranjera ha ganado parte del mercado que poseía la nacional y además a generado nuevo mercado que no manejaba la nacional, pero a pesar de esto la empresa extranjera no ha generado tantas ganancias como lo esperaba ya que ha tenido que hacer muchas inversiones para lograr penetrar en el mercado.

Después de varias negociaciones durante un par de meses las dos empresas han decidido unirse, básicamente lo que hará la empresa nacional será representar a la extranjera y siempre manejará su propio producto y así las dos esperan beneficiarse con esto.

Requerimientos

Cada una de las empresas ya posee un sistema de manejo de su inventario, clientes, proveedores, etc. Por razones lógicas se deben de integrar los sistemas en uno solo, la empresa nacional posee su información en una

base de datos de Microsoft Access y la empresa extranjera los tiene en Microsoft SQL Server, pero la empresa extranjera no posee una aplicación para la interacción con la BDD mientras que la nacional tiene una aplicación deficiente programada en Microsoft Visual Basic.

Por lo tanto es necesario realizar una migración entre las bases de datos; y además de esto se ha decidido que se desarrolle una nueva aplicación para el manejo de toda la información, la cual será manejada por la empresa nacional.

Uno de los requerimientos especiales que posee la aplicación es que no solo deberá de servir a la empresa nacional sino que deberá de generar distintos reportes a la empresa extranjera de sus productos cada vez que ésta lo requiera. Por lo tanto se creará un portal en la web para realizar este trabajo y además se presentará distinta información de las empresas y sus productos, ya que de esta forma se podrá tener cierta independencia entre las empresas.

También es necesario capacitar al personal que manejará el nuevo *software* para que estén listos para su utilización cuando se termine de desarrollar la aplicación.

2.7.4.1. Descripción de la solución

Se ha decidido que los datos se almacenen en Microsoft SQL Server y por lo tanto es necesario migrar los datos de la nacional a la nueva base de datos que se manejará y adaptar los datos existentes de la empresa extranjera a los requerimientos actuales.

La aplicación se realizará en Java utilizando los componentes Swing, para lograr una buena interfase con el usuario. Al final lo que se obtendrá será un archivo con *.jar el cual es un archivo ejecutable.

Para poder utilizar los componentes Swing de Java es necesario trabajar con una versión 1.2 o superior, y es lo que se pretende hacer. Esto permitirá un GUI (*Graphic User Interface*) flexible, ya se quiere que la aplicación esté dentro de un entorno mixto. El sistema operativo del servidor será Windows y las computadoras terminales tendrán Linux, y por esto se utilizará tecnología Java ya que ésta es independiente de plataformas.

Con forme se vallan cumpliendo los objetivos de la aplicación se irá capacitando al personal para el manejo del mismo, aproximadamente un mes después del inicio del proyecto y después se realizará semanalmente. Se asignará a un equipo específico para este trabajo y además se enviará a una persona para capacitar al personal de la empresa extranjera en su país, esto se realizará una vez que se tengan concluidos los requerimientos de las páginas de los reportes.

El portal y las páginas de consultas estarán en una arquitectura de JSP (*Java Server Pages*), y por lo tanto se utilizará un servidor JRun Server para que sirva las páginas.

2.7.4.2. ¿Por qué Scrum?

Debido a que se está realizando una integración el departamento de informática quedará reducido a un número de 8 miembros, todos poseen

experiencia en el ámbito y todos poseen conocimientos sobre el entorno de desarrollo que se manejará.

Una característica importante es que no todos los miembros han trabajado juntos, y en este punto es donde esta metodología toma una gran importancia, debido a las reuniones diarias que se manejan permitirán abrir canales de comunicación para que el equipo se logre integrar y actualizar constantemente el estado del proyecto y el conocimiento de los miembros del equipo.

Dentro de la misma reunión la gerencia debe de resolver cualquier problema que pueda surgir, ya que en ese momento se encuentran todos los miembros del equipo de desarrollo, se puede contar con su asesoría o sus comentarios para el beneficio de la implementación del sistema.

Como la metodología trabaja por medio de iteraciones se definirá la lista de actividades según su prioridad para que éstas sean completadas lo antes posible para satisfacer las necesidades del negocio.

Al momento de desarrollar se utilizarán los principios de XP como se describe en la metodología con lo cual se obtendrán todos los beneficios que conlleva esta metodología.

4. DIFERENCIAS ENTRE MÉTODOS TRADICIONALES Y ÁGILES

1.9. Introducción

En este capítulo final se tratará de aclarar ciertas diferencias que poseen los métodos tradicionales y los ágiles además de aclarar los beneficios que provee cada orientación. El desarrollo de *software* no es una tarea fácil, prueba de ello es que existen diversas metodologías las cuales pueden ser difícilmente aplicables en determinados casos y debido ha esto es que este capítulo posee una gran importancia.

1.10. Métodos utilizados actualmente en Guatemala

Últimamente han surgido muchas empresas que han intentado desarrollar *software* y han fracasado debido a que no manejan ningún tipo de método para desarrollar *software*, algunas otras han logrado subsistir gracias a los esfuerzos heroicos de su personal y manejan sus propios métodos empíricos para el trabajo de proyectos posteriores. Sin duda alguna no es sorprendente que la mayoría de las empresas que no tiene un proceso establecido quiebren al momento de desarrollar su primer proyecto.

La mayoría de las personas que se aventuran a desarrollar sistemas son programadores con poca experiencia en la parte administrativa de un proyecto como la especificación de requerimientos, el análisis, la negociación de los términos de la entrega, los costos, etc.; y por esto poseen una gran probabilidad de fracaso.

Algunas otras empresas (las más antiguas), dicen trabajar con un método tradicional de cascada, que sin siquiera saberlo están trabajando con un proceso de iteraciones que han desarrollado ellos mismos que han aprendido de su amplia experiencia, esto sucede cuando en un proyecto cometen cualquier tipo de error se dan a la tarea de reestructurar la parte en donde cometieron el error y corregirlo, en otras palabras lo que hacen es codificar y corregir, en donde cada error lleva a una iteración.

Estas empresas trabajan de esta forma ya que todo su equipo está conformado por personas que llevan mucho tiempo en el ámbito y que nunca actualizaron sus conocimientos quedando estancados en el tiempo, con procesos antiguos y sin mencionar las herramientas que utilizan que son más antiguas todavía, esto no indica que son malas pero si obsoletas ante los requerimientos actuales, y esto los lleva a no poder optar a muchos proyectos actuales.

En Guatemala frecuentemente se encuentra el uso del método iterativo incremental el cual es uno de los más conocidos internacionalmente y debido ha esto está fuertemente documentado y es fácil encontrar un sin número de documentos relacionados con este método. Muchas de las empresas que poseen experiencia en el desarrollo de proyectos exitosos se manejan por medio del método en cuestión, pero hacen una combinación con otros métodos.

El método iterativo ha ganado mucha reputación debido a que si se sigue fielmente con un equipo adecuado logra los objetivos de los proyectos, pero hay que tomar en cuenta que seguirlo no es nada sencillo. Y es muy común encontrar un proceso iterativo incremental incompleto y por lo tanto deficiente el cual no aporta ningún beneficio para la elaboración del proyecto.

La mayoría de empresas exitosas se manejan por medio de un proceso híbrido, toman como base una metodología (por lo regular iterativo incremental) y para las deficiencias que no logran cubrir por la forma en que utilizan la metodología o cualquier otra razón utilizan otra metodología. En muchos proyectos es muy común utilizar prototipos para revisar especificaciones, éste es un caso que resulta muy útil para muchas empresas.

Algunas empresas en su intento de actualizarse dicen utilizar XP, pero en realidad lo hacen con el único fin de ganarse el favor de los clientes por utilizar un método innovador, este problema es originado por la gran popularidad que ha ganado esta metodología ágil.

El mayor problema que enfrentan estas empresas es que están en un punto intermedio donde pretenden aplicar distintas metodologías que difieren en la forma de hacer las cosas, y nunca entran de lleno a implementar XP, y deciden continuar con la forma que hacían las cosas pero sin dejar de promover que trabajan con XP.

1.11. Comparación entre las metodologías ágiles y tradicionales

1.11.1. Diferencias

Los métodos más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, también han presentado problemas en muchos otros.

Las metodologías ágiles tienen sus principales diferencias en los siguientes aspectos

Se deben de anteponer al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es 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 sobre la base de sus necesidades, dentro del cual se sentirán más cómodos.

Desarrollar software que funciona más que conseguir una buena documentación. La regla a seguir es “no producir documentos a menos que sean necesarios para tomar una 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 lo largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Mejorar la comunicación y el equipo. Una de los factores más importantes es la importancia que se les da a las personas. Como promover la comunicación y el trabajo en equipo siguiendo las siguientes prácticas: participación activa, modelación en conjunto, propiedad colectiva.

En base a lo anterior se define la siguiente lista

Metodologías ágiles

- Basadas en heurísticas provenientes de prácticas de producción de código
- Especialmente preparados para cambios durante el proyecto
- Reglas impuestas internamente (por el equipo)
- Proceso menos controlado, con pocos principios
- No existe contrato tradicional o al menos es bastante flexible
- El cliente es parte del equipo de desarrollo
- Grupos pequeños (<20 integrantes) y trabajando en el mismo sitio
- Pocos artefactos
- Pocos roles
- Menos énfasis en la arquitectura del *software*
- Se enfoca en el trabajo del equipo
- Comunicación diaria del estado del sistema

Metodologías Tradicionales

- Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
- Cierta resistencia a los cambios
- Proceso mucho más controlado, con numerosas políticas o normas
- Existe un contrato prefijado

- El cliente interactúa con el equipo de desarrollo mediante reuniones
- Reglas impuestas externamente
- Grupos grandes y posiblemente distribuidos
- Más artefactos
- Más roles
- La arquitectura del *software* es esencial y se expresa mediante modelos

1.11.2. Desventajas de las metodologías

1.11.2.1. Tradicionales

La inclusión de demasiados procesos de desarrollo, más actividades, más artefactos y más restricciones, puede resultar en un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto.

El enfoque tradicional no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Para muchos equipos de desarrollo el uso de metodologías tradicionales les resulta muy lejano a su forma de trabajo actual considerando las dificultades de su introducción e inversión asociada en formación y herramientas.

Los requerimientos y diseño son obviamente parte vital de cualquier proceso de *software*, en los procesos tradicionales se deben de tomar todos los requerimientos desde el inicio y modelar a detalle toda la arquitectura de todo el sistema y en un entorno cambiante se convierte en recursos desperdiciados.

En ocasiones la empresa requiere de un alto nivel de papeleo, por ejemplo el cambio de un requerimiento debe ser firmado por tres niveles administrativos.

1.11.2.2. Ágiles

Un proceso ágil requiere cierta disciplina, y en ocasiones es necesario trabajar duro y tener paciencia con los desarrolladores para obtener un beneficio completo.

Las metodologías ágiles comúnmente requieren un equipo calificado que puedan trabajar en todo. No se pueden tener elementos que trabajen a medias por falta de voluntad o capacidad.

La creación de documentos necesarios para la manutención del sistema puede convertirse en un problema ya que estos tienden a realizarse hasta el final del desarrollo o a menos que se este seguro de que son necesarios.

Puede volverse casi imposible manejar un proyecto si el equipo de trabajo es demasiado grande ya que los canales de comunicación se incrementan en gran medida y se crearía un descontrol en proceso y en el flujo de la información.

El no contar con un cliente que colabore con reuniones periódicas para la revisión de los avances y otras cosas podría poner en riesgo el proyecto.

1.11.3. Ventajas de las metodologías

1.11.3.1. Tradicionales

Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados, en otras palabras se puede afinar el proceso tanto como sea necesario.

El método iterativo incremental se puede ver como una guía de cómo utilizar UML en forma mas efectiva, modelando todo en forma visual, lo cual resulta de gran utilidad.

Se intenta abordar las tareas más riesgosas primero, con esto se logra reducir los riesgos del proyecto y tener un subsistema ejecutable tempranamente.

Esta fuertemente documentado, se encuentran fácilmente personas con experiencias en la utilización de este enfoque, sus lineamientos han sido probados incontable número de veces y por lo tanto los procesos son maduros y confiables.

1.11.3.2. Ágiles

Se debe de recopilar todo lo que se cree que será necesario para la próxima iteración y proceder con eso, de esta forma no se harán esfuerzos innecesarios.

Uno de los beneficios de los procesos ágiles sobre los equipos es que se dicen entre ellos lo que sucede y fácilmente pueden hablar y escuchar al cliente. Por ejemplo ambientes donde los programadores no están separados en cubículos.

En lugares donde se trabaja de forma ágil se trabaja con alta transparencia, permitiéndole al cliente ver lo que esta sucediendo en el proyecto en cualquier momento. Si un proyecto esta atrasado, los procesos ágiles dificultan poder ocultar esto del cliente, lo que se pretende evitar es tener un conjunto de sorpresas para el cliente, especialmente una en la que un día antes de la fecha limite de entrega se le tenga que decir al cliente que tomara 3 meses mas terminar.

El poder responder a los cambios de requerimientos en forma robusta y en buen tiempo, logrando todo esto sin sobrecargar al equipo de para que el trabajo este hecho, representa una gran ventaja en cualquier campo.

Asegurar el progreso con *software* funcional, apuntando a la entrega del sistema que el cliente quiere al final del proyecto, no lo que busca al principio.

1.12. ¿Por qué utilizar metodologías ágiles?

Ante la situación anterior, las metodologías ágiles aparecen como una posible respuesta para llenar este vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida, con una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Por otro lado, las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de *software*; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, nuevas tecnologías, etc.

Otra característica importante es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto. Ésta 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.

1.13. ¿Cuándo se deben de utilizar métodos ágiles?

El uso de un método ágil no es para todos. Hay que tener en cuenta varias cosas si se decide a seguir por este camino. Sin embargo es evidente que estas nuevas metodologías son extensamente aplicables y deben ser usadas por más personas de las que actualmente lo consideran.

En el ambiente actual, la metodología más común es codificar y corregir, aplicar más disciplina que caos seguramente ayudará al desarrollo del sistema, mucha de la ventaja de los métodos ágiles es de hecho su peso ligero. Los procesos más simples son más probables de ser seguidos cuando se está acostumbrado a ningún proceso en absoluto, como es la mayoría de casos.

Una de las limitaciones más grandes de estas nuevas metodologías es cómo manejan los equipos más grandes. Como muchas nuevas tendencias, éstas tienden a ser usadas primero a pequeña escala antes que a gran escala. También a menudo se han creado con énfasis en equipos pequeños. La XP explícitamente dice que está diseñada para equipos de no más de veinte personas. Hay que recordar que muchos equipos de *software* pueden reducirse en tamaño sin reducir su productividad total.

Otras tendencias ágiles están destinadas a equipos más grandes. La metodología FDD fue diseñada originalmente para un proyecto de cincuenta personas. Algunas empresas han usado proyectos influidos por la XP con equipos de aproximadamente 100 personas en tres continentes. Scrum se ha utilizado para manejar tamaños similares.

Lo que se desea dejar claro es que los procesos ágiles son buenos cuando sus requisitos son inciertos o volátiles. Si no se tiene requisitos estables, entonces no está en la posición de tener un plan estable y seguir un proceso planeado. En estas situaciones un proceso adaptable puede ser menos cómodo, pero será más eficaz. A menudo la barrera más grande aquí es el cliente.

Si se utiliza el camino adaptable, es vital confiar en los desarrolladores e involucrarlos en la toma de decisiones. Los procesos adaptables cuentan en que el responsable confía en los desarrolladores.

Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de *software*; aquellos en los cuales los

equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

Para resumir. Los siguientes factores sugieren utilizar un proceso ágil

- Requisitos inciertos o volátiles
- Desarrolladores responsables y motivados
- Clientes que entienden y se involucrarán.

Estos factores sugieren un proceso tradicional

- Un equipo de alrededor de cien o más
- Un precio fijo, o más correctamente un alcance o contrato fijo

CONCLUSIONES

1. No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de *software*. Toda metodología debe ser adaptada al contexto del proyecto, recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.

2. El mayor problema que enfrentan estas nuevas metodologías es la resistencia al cambio y el desconocimiento que presentan las empresas desarrolladoras, la primera debido al gran escepticismo que se genera alrededor de éstas, a causa de que algunas características carecen de un respaldo sólido, más que el propio conocimiento de sus creadores y la segunda por ignorancia.

3. Debido a la poca formalidad que existe en las empresas guatemaltecas en lo referente al desarrollo de sistemas, éstas refuerzan la desconfianza que existe en muchas otras empresas para la decisión de implementar algún tipo de sistema, lo cual afecta al desarrollo del país.

4. Una de las cualidades más destacables en una metodología ágil, es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implementación en el equipo de desarrollo. Cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios, el uso de tecnologías que no tengan un ciclo rápido de realimentación, o que no soporten fácilmente el cambio, etc. pueden llevar al fracaso del proceso.

5. Algunas empresas al momento de desarrollar, tergiversan los requisitos debido a su experiencia, influenciando a los clientes a tener productos que

no desean, y esto ocasiona que se generen problemas posteriores por entregar un producto no deseado, hay que tomar los requerimientos siempre como nuevos, y utilizar nuestra experiencia al momento de desarrollar.

6. Muchas empresas se manejan por medio de reglas no escritas, lo cual refuerza el caos que se genera dentro de las empresas. El hecho de que las metodologías sean orientadas a las personas y no a los procesos, les provee de una ventaja de los otros métodos de desarrollo que buscan hacer el trabajo más predecible, obteniendo con esto largas listas de requerimientos descritos por los desarrolladores y no por los clientes.

7. La constante retroalimentación que se obtiene de la comunicación entre el cliente y los desarrolladores, provee una mejora significativa en la adquisición de requerimientos, logrando un mejor producto.

8. Entre más personas estén involucradas en el proyecto, más grande debe ser la metodología.

RECOMENDACIONES

1. Puesto que en algunas metodologías no se logró enfatizar demasiado, resulta necesario realizar un estudio más profundo de cada una de ellas. Además, hay que realizar una investigación orientada hacia: métricas y evaluación del proceso, herramientas específicas, para apoyar las prácticas ágiles, aspectos humanos y de trabajo en equipo.

2. 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, claro que esto no significa permitir que hagan lo que deseen y realizar su trabajo cuando lo deseen. De nada sirven buenas notaciones y herramientas si no se proveen directivas para su aplicación.

3. No utilizar estas metodologías fuera de los alcances reales que poseen, porque de hacer esto de antemano estaríamos arriesgando el éxito del proyecto. Las empresas más exitosas de desarrollo saben cómo aplicar distintas metodologías en distintas situaciones.

4. En la medida de lo posible, tener a todo el personal trabajando en el mismo lugar, esto beneficiará en todo sentido al proyecto, ya que reduciremos los costos de comunicación en gran medida, y facilitaremos el aprendizaje del equipo.

5. Si el equipo de desarrollo es nuevo, les convendrá utilizar un proceso ágil y no uno tradicional, y lo pueden adaptar a sus necesidades debido a su flexibilidad, y por lo tanto el equipo estará contento con el método, lo cual debe ser un aspecto vital para el éxito del mismo.

BIBLIOGRAFÍA

1. Agile Modeling. Página de Internet http://www.agilemodeling.com/principles.htm

2. Agile Spain. Página de Internet http://www.agile-spain.com/index.php

3. Alistair Cockburn. Página de Internet http://alistair.cockburn.us,

4. Control Chaos. Página de Internet http://www.controlchaos.com

5. Cristal Methodologies. Página de Internet http://www.crystalmethodologies.org

6. Doug Rosenberg, Matt Stephens y Mark Collins-Cope. Agile Development with ICONIX Process, Estados Unidos de América, Apress 2005.

7. DSDM Consortium. Página de Internet http://www.dsdm.org

8. Feature Driven Development. Página de Internet http://www.featuredrivendevelopment.com

9. ISO 9000-3 1997 Guidelines *Software* Standard. Página de Internet http://praxiom.com/iso-9000-3.htm

10. ISO IEC 90003 2004 *SOFTWARE* STANDARD. Página de Internet http://praxiom.com/iso-90003.htm

11. ISO Internacional Organization for Standarization. Página de Internet http://www.iso.ch/iso/en/iso9000-14000/iso9000/iso9000index.html,

12. Jim Highsmith. Página de Internet http://www.adaptivesd.com

13. Manifesto for Agile *Software* Development. Página de Internet http://agilemanifesto.org, Diciembre 2004

14. Poppendieck LLC. Página de Internet http://www.poppendieck.com

15. Refactoring. Página de Internet http://c2.com/cgi/wiki?Refactoring,

16. *Software* Technology Support Center. Página de Internet http://www.stsc.hill.af.mil/index.html

17. Sticky Minds. Página de Internet http://www.stickyminds.com/

18. The New Methodology. Página de Internet http://martinfowler.com/articles/newMethodology.html

19. Wiki Extreme Programming. Página de Internet http://www.programacionextrema.org/cgi-bin/wiki.pl?ExtremeProgramming

20. Wikipedia, la enciclopedia libre. Página de Internet http://es.wikipedia.org/wiki/Portada