Metodología Ágil de Desarrollo de Software – XP

Borja López Yolanda
yoly.borja@gmail.com
ESPE, MEVAST

RESUMEN

Las metodologías ágiles han ganado bastante popularidad desde hace algunos años. Si bien son una muy buena solución para proyectos a corto plazo, en especial, aquellos proyectos en donde los requerimientos están cambiando constantemente, en proyecto a largo plazo, el aplicar metodologías ágiles no dan tan buenos resultados. El diseño de la arquitectura de software es una práctica muy importante para el desarrollo de software. Tener una buena arquitectura implica que nuestro sistema tiene atributos de calidad que nos van a dar un valor muy importante en el software. Si se definen actividades que fomenten el uso de métodos para el desarrollo de la arquitectura, en un proceso de desarrollo de software, se puede obtener muchos beneficios con respecto al producto que se desarrollo. Sin embargo, para las metodologías ágiles esas actividades no se consideran en forma importante.

Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

La razón de ser de este trabajo se basa en el análisis de algunos tipos de metodologías existentes, viendo sus principales características, conceptos y conclusiones.

ABSTRACT

Agile methodologies have gained a lot of popularity in recent years. While they are a very good solution for short-term projects, especially those projects where requirements are constantly changing, in long-term project, applying agile methodologies do not give such good results. The design of software architecture is a very important practice for software development. To have a good architecture means that our system quality attributes will give us a very important value in the software. If you define activities that encourage the use of methods for the development of architecture in a software development process, you can get many benefits with respect to the product being developed. However, to agile methodologies such activities are not considered as important.

This is the philosophy of agile methodologies, which give more value to the individual, to the collaboration with customers and incremental software development with very short interactions. This approach is showing its effectiveness in projects with changing requirements as required drastically reduce development time while maintaining high quality.

Agile methodologies are revolutionizing the way to produce software, while this generates much debate between its supporters and those of skepticism or belief do not see them as an alternative to traditional methodologies.

The rationale for this study is based on analysis methodologies Xp, its main features, concepts and conclusions.

JUSTIFICACION

La metodología ágil XP surge con el propósito de transformar la esencia del desarrollo de los procesos que se ejecutan al momento de llevar acabo la planificación y ejecución de un proyecto de creación de software esta metodología se enfoca en integrar en una mejora continua al usuario y al grupo de individuo que se encargaran de resolver la problemática percibida.

XP en comparación de las metodologías tradicionales es mas rápida, ya que conlleva menos protocolo, lo que evita que existan jerarquías dentro del grupo, lo cual permite que cada integrante del grupo pueda aportar en cualquier motivo, por la cual se

implementara o hará uso de esta metodología es, que la misma se enfoca en resultados a corto plazos es decir los resultados que se van obteniendo a lo largo de la modulación serán verificados al instante y de existir alguna anomalía o falta se hará las correcciones correspondientes. De esta forma rápida es posible obtener los resultados esperados a corto plazo y de manera eficiente el sistema tomara robustez en la utilización de los módulos se lograra de manera instantánea.

Los resultados son verificados por el usuario de manera que apruebe el producto y así obtener una seguridad y solidez en el desarrollo del sistema, de esta manera el resultado final lograra satisfacer en un gran porcentaje la expectativas y las necesidades del usuario.

1. INTRODUCCIÓN

La programación extrema o XP es una metodología de desarrollo que se englobaría dentro de las denominadas metodologías Ágiles en la que se da máxima prioridad a la obtención de resultados y reduce la burocracia que utiliza las metodologías tradicionales.

Generalmente el proceso de desarrollo llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque 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.

Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir de las buenas prácticas de la Ingeniería del Software, asumiendo el riesgo que ello conlleva.

En este contexto las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las Metodologías Ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

2. QUE ES XP

XP es una metodología ágil para el desarrollo de software y consiste básicamente en ajustarse estrictamente a una serie de reglas que se centran en las necesidades del cliente para lograr un producto de buena calidad en poco tiempo, centrada en potenciar las relaciones interpersonales como clave para el éxito del desarrollo de software.

La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo.

Promueve el trabajo en equipo, preocupándose en todo momento del aprendizaje de los desarrolladores y estableciendo un buen clima de trabajo.

Este tipo de programación es la adecuada para los proyectos con requisitos imprecisos, muy cambiantes y donde existe un alto riesgo técnico.

XP está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, dónde la comunicación sea más factible que en grupos de desarrollo grandes. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes.

3. VALORES DE XP

3.1. Comunicación

Prevalece en todas las prácticas de Extreme Programming. Comunicación cara a cara es la mejor forma de comunicación, entre los desarrolladores y el cliente. Método muy ágil. Gracias a esto el equipo esta pude realizar cambios que al cliente no le gustaron.

3.2. Simplicidad

La simplicidad ayuda a que los desarrolladores de software encuentren soluciones más simples a problemas, según el cliente lo estipula. Los desarrolladores también crean características en el diseño que pudieran ayudar a resolver problemas en un futuro.

3.3. Retroalimentación

La retroalimentación continua del cliente permite a los desarrolladores llevar y dirigir el proyecto en una dirección correcta hacia donde el cliente quiera.

3.4. Valentía

Requiere que los desarrolladores vayan a la par con el cambio, por que sabemos que este cambio es inevitable, pero el estar preparado con una metodología ayuda a ese cambio. Programa para hoy y no para mañana.

3.5. Respeto

El equipo debe trabajar como uno, sin hacer decisiones repentinas. Extreme Programming promueve el trabajo del equipo. Cada integrante del proyecto (cliente, desarrolladores, etc.) forman parte integral del equipo encargado de desarrollar software de calidad. El equipo debe trabajar como uno, sin hacer decisiones repentinas.

4. ROLES XP

Aunque en otras fuentes de información aparecen algunas variaciones y extensiones de roles XP, en este apartado describiremos los roles de acuerdo con la propuesta original de Beck.

4.1. Programador

El programador escribe las pruebas unitarias y produce el código del sistema. Debe existir una comunicación y coordinación adecuada entre los programadores y otros miembros del equipo.

4.2. Cliente

El cliente escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio. El cliente es sólo uno dentro del proyecto pero puede corresponder a un interlocutor que está representando a varias personas que se verán afectadas por el sistema.

4.3. Encargado de pruebas (Tester)

El 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.

4.4. Encargado de seguimiento (Tracker)

El encargado de seguimiento proporciona realimentación al equipo en el proceso XP. Su responsabilidad es verificar el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, comunicando los resultados para mejorar futuras estimaciones. También realiza el seguimiento del progreso de cada iteración y evalúa si los objetivos son alcanzables con las restricciones de tiempo y recursos presentes.

Determina cuándo es necesario realizar algún cambio para lograr los objetivos de cada iteración.

4.5. Entrenador (Coach)

Es responsable del proceso global. Es necesario que conozca a fondo el proceso XP para proveer guías a los miembros del equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

4.6. Consultor

Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto. Guía al equipo para resolver un problema específico.

4.7. Gestor (Big boss)

Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

5. MODELO XP

La metodología XP define cuatro variables para cualquier proyecto de software: *costo, tiempo, calidad y alcance.*

Además, se especifica que, de estas cuatro variables, sólo tres de ellas podrán ser fijadas arbitrariamente por actores externos al grupo de desarrolladores (clientes y jefes de proyecto). El valor de la variable restante podrá ser establecido por el equipo de desarrollo, en función de los valores de las otras tres. Este mecanismo indica que, por ejemplo, si el cliente establece el alcance y la calidad, y el jefe de proyecto el precio, el grupo de desarrollo tendrá libertad para determinar el tiempo que durará el proyecto.

Por esto, se trata de realizar ciclos de desarrollo cortos (llamados iteraciones), con entregables funcionales al finalizar cada ciclo. En cada iteración se realiza un ciclo completo de análisis, diseño, desarrollo y pruebas, pero utilizando un conjunto de reglas y prácticas que caracterizan a XP.

Típicamente un proyecto con XP lleva 10 a 15 ciclos o iteraciones. La siguiente figura esquematiza los

ciclos de desarrollo en cascada e iterativos tradicionales (por ejemplo, incremental o espiral), comparados con el de XP.

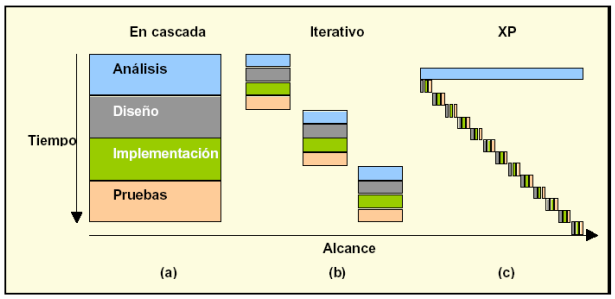

Fig. 1 Evolución de los largos ciclos de desarrollo en cascada (a) a ciclos iterativos más cortos (b) y a la mezcla que hace XP.

6. PROCESO XP

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

- El cliente define el valor de negocio a implementar.
- El programador estima el esfuerzo necesario para su implementación.
- El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
- El programador construye ese valor de negocio.
- Vuelve al paso 1.

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

Si bien el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en las siguientes Fases:

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

6.1. Fase I: Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

6.2. Fase II: Planificación de la Entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

El resultado de esta fase es un Plan de Entregas, o *"Release Plan"*

6.3. Fase III: Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está

compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son:

- historias de usuario no abordadas,
- velocidad del proyecto,
- pruebas de aceptación no superadas en la iteración anterior y
- tareas no terminadas en la iteración anterior.

Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

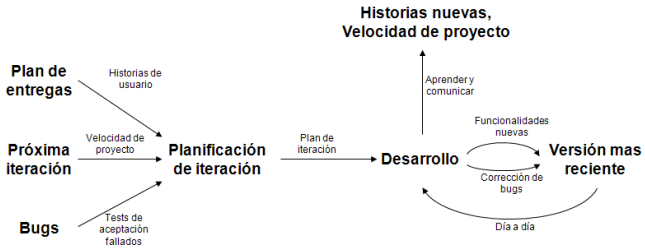

Fig. 2 Iteración

6.4. Fase IV: Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

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

En esta fase no se realizan más desarrollos funcionales, pero pueden ser necesarias tareas de ajuste ("fine tuning").

6.5. Fase V: Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

6.6. Fase VI: Muerte del Proyecto

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

7. REGLAS Y PRÁCTICAS

La metodología XP tiene un conjunto importante de reglas y prácticas. En forma genérica, se pueden agrupar en:

- Reglas y prácticas para la Planificación
- Reglas y prácticas para el Diseño
- Reglas y prácticas para el Desarrollo
- Reglas y prácticas para las Pruebas

7.1. Planificación

La metodología XP plantea la planificación como un dialogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores o gerentes. El proyecto comienza recopilando "Historias de usuarios", las que sustituyen a los tradicionales "casos de uso". Una vez obtenidas las "historias de usuarios", los programadores evalúan rápidamente el tiempo de desarrollo de cada una.

Si alguna de ellas tiene "riesgos" que no permiten establecer con certeza la complejidad del desarrollo, se realizan pequeños programas de prueba (*"spikes"*), para reducir estos riesgos. Una vez realizadas estas estimaciones, se organiza una reunión de planificación, con los diversos actores del proyecto (cliente, desarrolladores, gerentes), a los efectos de establecer un plan o cronograma de entregas (*"Release Plan"*) en los

que todos estén de acuerdo. Una vez acordado este cronograma, comienza una fase de iteraciones, en dónde en cada una de ellas se desarrolla, prueba e instala unas pocas "historias de usuarios".

Según Martín Fowler (uno de los firmantes del "Agile Manifesto"), los planes en XP se diferencian de las metodologías tradicionales en tres aspectos:

Simplicidad del plan. No se espera que un plan requiera de un "gurú" con complicados sistemas de gerenciamiento de proyectos.

Los planes son realizados por las mismas personas que realizarán el trabajo.

Los planes no son predicciones del futuro, sino simplemente la mejor estimación de cómo saldrán las cosas. Los planes son útiles, pero necesitan ser cambiados cuando las circunstancias lo requieren. De otra manera, se termina en situaciones en las que el plan y la realidad no coinciden, y en estos casos, el plan es totalmente inútil.

Los conceptos básicos de esta planificación son los siguientes:

7.1.1. Las Historias de Usuario

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales.

El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento historias de usuario pueden romperse, remplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Respecto de la información contenida en la historia de usuario, existen varias plantillas sugeridas pero no existe un consenso al respecto. En muchos casos sólo se propone utilizar un nombre y una descripción o sólo una descripción, más quizás una estimación de esfuerzo en días. Beck en su libro presenta un ejemplo de ficha (*customer story and task card*) en la cual pueden reconocerse los siguientes contenidos:

- Fecha,
- tipo de actividad (nueva, corrección, mejora),
- prueba funcional, número de historia,
- prioridad técnica y del cliente,
- referencia a otra historia previa,
- riesgo,
- estimación técnica,
- descripción,
- notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios.

7.1.2. Plan de entregas ("Release Plan")

El cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto (cliente, desarrolladores, gerentes, etc.).

XP denomina a esta reunión "Juego de planeamiento" (*"Planning game"*), pero puede denominarse de la manera que sea más apropiada al tipo de empresa y cliente (por ejemplo, Reunión de planeamiento, *"Planning meeting"* o *"Planning workshop"*) Típicamente el cliente ordenará y agrupará según sus prioridades las historias de usuario. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores.

Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario.

7.1.3. Plan de iteraciones ("Iteration Plan")

Las historias de usuarios seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden prestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración.

Cada historia de usuario se traduce en tareas específicas de programación.

Así mismo, para cada historia de usuario se establecen las pruebas de aceptación.

Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores. Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para

evaluar su corrección, así como para prever que no vuelvan a ocurrir.

7.1.4. Reuniones diarias de seguimiento ("Stand-up meeting")

El objetivo de tener reuniones diarias es mantener la comunicación entre el equipo, y compartir problemas y soluciones. En la mayoría de estas reuniones, gran parte de los participantes simplemente escuchan, sin tener mucho que aportar. Para no quitar tiempo innecesario del equipo, se sugiere realizar estas reuniones en círculo y de pie.

7.2. Diseño

La metodología XP hace especial énfasis en los diseños simples y claros. Los conceptos más importantes de diseño en esta metodología son los siguientes:

7.2.1. Simplicidad:

Un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione. Se sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando.

7.2.2. Soluciones "spike"

Cuando aparecen problemas técnicos, o cuando es difícil de estimar el tiempo para implementar una historia de usuario, pueden utilizarse pequeños programas de prueba (llamados "spike"), para explorar diferentes soluciones. Estos programas son únicamente para probar o evaluar una solución, y suelen ser desechados luego de su evaluación.

7.2.3. Recodificación

La recodificación (*"refactoring"*) consiste en escribir nuevamente parte del código de un programa, sin cambiar su funcionalidad, a los efectos de hacerlo más simple, conciso y/o entendible. Muchas veces, al terminar de escribir un código de programa, pensamos que, si lo comenzáramos de nuevo, lo hubiéramos hecho en forma diferente, mas clara y eficientemente. Sin embargo, como ya está pronto y "funciona", rara vez es rescrito.

Las metodologías de XP sugieren recodificar cada vez que sea necesario. Si bien, puede parecer una pérdida de tiempo innecesaria en el plazo inmediato, los resultados de ésta práctica tienen sus frutos en las siguientes iteraciones, cuando sea necesario ampliar o cambiar la funcionalidad. La filosofía que se persigue es, como ya se mencionó, tratar de mantener el código más simple posible que implemente la funcionalidad deseada.

7.2.4. Metáforas

Una "metáfora" es algo que todos entienden, sin necesidad de mayores explicaciones. La metodología XP sugiere utilizar este concepto como una manera sencilla de explicar el propósito del proyecto, y guiar la estructura y arquitectura del mismo. Por ejemplo, puede ser una guía para la nomenclatura de los métodos y las clases utilizadas en el diseño del código. Tener nombres claros, que no requieran de mayores explicaciones, redunda en un ahorro de tiempo.

Es muy importante que el cliente y el grupo de desarrolladores estén de acuerdo y compartan esta "metáfora", para que puedan dialogar en un "mismo idioma". Una buena metáfora debe ser fácil de comprender para el cliente y a su vez debe tener suficiente contenido como para que sirva de guía a la arquitectura del proyecto. Sin embargo, ésta práctica resulta, muchas veces, difícil de realizar.

7.3. Desarrollo

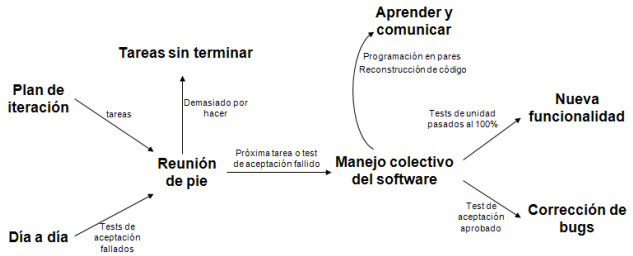

Fig. 2 Desarrollo

7.3.1. Disponibilidad del cliente

Uno de los requerimientos de XP es tener al cliente disponible durante todo el proyecto. No solamente como apoyo a los desarrolladores, sino formando parte del grupo.

El involucramiento del cliente es fundamental para que pueda desarrollarse un proyecto con la metodología XP.

Al comienzo del proyecto, el cliente debe proporcionar las historias de usuarios.

Pero, dado que estas historias son expresamente cortas y de "alto nivel", no contienen los detalles necesarios para realizar el desarrollo del código. Estos detalles deben ser proporcionados por el cliente, y discutidos con los desarrolladores, durante la etapa de desarrollo. No se requieren de largos documentos de especificaciones, sino que los detalles son proporcionados por el cliente, en el momento adecuado, "cara a cara" a los desarrolladores.

7.3.2. Uso de estándares

Si bien esto no es una idea nueva, XP promueve la programación basada en estándares, de manera que sea fácilmente entendible por todo el equipo, y que facilite la recodificación.

7.3.3. Programación dirigida por las pruebas ("Test-driven programming")

En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los tests, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo. La metodología XP propone un modelo inverso, en el que, lo primero que se escribe son los test que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario para pasar las pruebas previamente definidas.

Las pruebas a los que se refieren esta práctica, son las pruebas unitarias, realizados por los desarrolladores. La definición de estos test al comienzo, condiciona o "dirige" el desarrollo.

7.3.4. Programación en pares

XP propone que se desarrolle en pares de programadores, ambos trabajando juntos en un mismo ordenador. Si bien parece que ésta práctica duplica el tiempo asignado al proyecto (y por ende, los costos en recursos humanos), al trabajar en pares se minimizan los errores y se logran mejores diseños, compensando la inversión en horas. El producto obtenido es por lo general de mejor calidad que cuando el desarrollo se realiza por programadores individuales.

Adicionalmente, la programación en pares tiene las siguientes ventajas:

- La mayoría de los errores se descubren en el momento en que se codifican, ya que el código es permanentemente revisado por dos personas.

- La cantidad de defectos encontrados en las pruebas es estadísticamente menor.

- Los diseños son mejores y el código más corto.
- El equipo resuelve problemas en forma más rápida.
- Las personas aprenden significativamente más, acerca del sistema y acerca de desarrollo de software.
- El proyecto termina con más personas que conocen los detallas de cada parte del código.
- Las personas aprenden a trabajar juntas, generando mejor dinámica de grupo y haciendo que la información fluya rápidamente.
- Las personas disfrutan más de su trabajo.

7.3.5. Integraciones permanentes

Todos los desarrolladores necesitan trabajar siempre con la "última versión". Realizar cambios o mejoras sobre versiones antiguas causan graves problemas, y retrasan al proyecto. Es por eso que XP promueve publicar lo antes posible las nuevas versiones, aunque no sean las últimas, siempre que estén libres de errores. Idealmente, todos los días deben existir nuevas versiones publicadas. Para evitar errores, solo una pareja de desarrolladores puede integrar su código a la vez**.**

7.3.6. Propiedad colectiva del código

En un proyecto XP, todo el equipo puede contribuir con nuevas ideas que apliquen a cualquier parte del proyecto. Asimismo, cualquier pareja de programadores puede cambiar el código que sea necesario para corregir problemas, agregar funciones o recodificar. En otras metodologías, este concepto puede parecer extraño. Muchas veces se asume que, si hay algo de propiedad colectiva, la responsabilidad también es colectiva. Y que "todos sean responsables", muchas veces significa que "nadie es responsable".

7.3.7. Ritmo sostenido

La metodología XP indica que debe llevarse un ritmo sostenido de trabajo. Anteriormente, ésta práctica se denominaba "Semana de 40 horas". Sin embargo, lo importante no es si se trabajan, 35, 40 o 42 horas por semana. El concepto que se desea establecer con esta práctica es el de planificar el trabajo de manera de mantener un ritmo constante y razonable, sin sobrecargar al equipo.

Cuando un proyecto se retrasa, trabajar tiempo extra puede ser más perjudicial que beneficioso. El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto. En la medida de lo posible, se debería renegociar el plan de entregas (*"Release Plan"*), realizando una nueva reunión de planificación con el cliente, los desarrolladores y los gerentes. Adicionalmente, agregar más desarrolladores en proyectos ya avanzados no siempre resuelve el problema.

7.4. Pruebas

7.4.1. Pruebas unitarias

Las pruebas unitarias son una de las piedras angulares de XP. Todos los módulos deben de pasar las pruebas unitarias antes de ser liberados o publicados. Las pruebas deben ser definidas antes de realizar el código (*"Test-driven programming"*).

Que todo código liberado pase correctamente las pruebas unitarias es lo que habilita que funcione la propiedad colectiva del código. En este sentido, el sistema y el conjunto de pruebas debe ser guardado junto con el código, para que pueda ser utilizado por otros desarrolladores, en caso de tener que corregir, cambiar o recodificar parte del mismo.

7.4.2. Detección y corrección de errores

Cuando se encuentra un error ("bug"), éste debe ser corregido inmediatamente, y se deben tener precauciones para que errores similares no vuelvan a ocurrir. Asimismo, se generan nuevas pruebas para verificar que el error haya sido resuelto.

7.4.3. Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las historias de usuarios, en cada ciclo de la iteración del desarrollo. El cliente debe especificar uno o diversos escenarios para comprobar que una historia de usuario ha sido correctamente implementada.

Las pruebas de aceptación son consideradas como "pruebas de caja negra" (*"Black box system tests"*). Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Asimismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución.

Una historia de usuario no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información.

8. VENTAJAS Y DESVENTAJAS

8.1. Ventajas

Evidentemente, para que algo esté siendo tomado tan en cuenta como la XP, debe ofrecer una serie de ventajas a la hora de ponerlo en práctica que haga que el esfuerzo de entender y aplicar sus prácticas, sea insignificante con respecto a los beneficios obtenidos.

- Se consiguen productos usables con mayor rapidez.
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo. Se consigue integrar todo el trabajo con mucha mayor facilidad.
- Se atienden las necesidades del usuario con mayor exactitud. Esto se consigue gracias a las continuas versiones que se ofrecen al usuario.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de los test de forma previa a la codificación.
- Obtenemos código más simple y más fácil de entender, reduciendo el número de errores.
- Gracias a la filosofía del "pair programming" (programación en parejas), se consigue que los desarrolladores apliquen las buenas prácticas que se les ofrecen con la XP.
- Gracias al "refactoring" es más fácil el modificar los requerimientos del usuario.
- Conseguimos tener un equipo de desarrollo más contento y motivado. Las razones son, por un lado el que la XP no permite excesos de trabajo (se debe trabajar 40 horas a la semana), y por otro la comunicación entre los miembros del equipo que consigue una mayor integración entre ellos.

8.2. Desventajas

- Resulta muy complicado planear el proyecto y establecer el costo y la duración del mismo.
- No se puede aplicar a proyectos de gran escala, que requieran mucho personal, a menos que se las subdivida en proyectos más pequeños.
- Es más complicado medir los avances del proyecto, pues es muy complicado el uso de una medida estándar.
- Altas comisiones en caso de fallar.

9. CONCLUSIONES Y RECOMENACIONES

9.1. Conclusiones

- Es difícil estimar el costo y duración del proyecto por no existir una definición desde inicio.
- El cliente le da un valor agregado al proyecto ya que ayuda a que los requerimientos sean más compresibles y de fácil aprobación.
- Organiza a los integrantes del equipo para trabajar a un mismo ritmo divertido y con horario adecuado.
- Los desarrolladores deben estar involucrados en todas las funcionalidades del proyecto
- La programación en parejas ayuda a reducir costo y tiempos y mejorar la calidad del proyecto.
- Los métodos en cascada o espiral son los más adecuados para proyectos.
- Que requieran decenas de desarrolladores y en los que las especificaciones estén claramente determinadas desde el comienzo.

9.2. Recomendaciones.

- Para proyectos medianos, y en los que las especificaciones no se puedan obtener hasta luego de comenzado el proyecto, XP es la metodología recomendada.
- Usar XP en equipos pequeños de desarrollo que no superen a 20
- Se debe involucrar al cliente en el desarrollo del proyecto desde el inicio hasta el final ya que es indispensable que conozca y apruebe la metodología.
- Los desarrolladores deben compartan el código y ser responsables del código de todo el proyecto.

10. BIBLIOGRAFÍA

[1] **BECK** Kent, **MARTIN** Fowler. *Planning Extreme Programming* 2da. Edición, Boston, 2004.

[2] **SOMMERVILLE** Ian. *Ingeniería del Software* 7ma. Edición, Pearson Educación S.A, Madrid, 2006.

[3] **FERNANDEZ** Luis. *Una revisión sistemática de la adaptación del proceso software*, Revista Española de Innovación Calidad e Ingeniería del Software (REICES), Vol.3, No. 2, Pág. 21-38, Octubre 2007.

[4] **CASTRO** Paco, *Programación Orientada a Objetos*, Revista del Instituto Tecnológico de Informática, Vol. 1, No. 1, Pág. 4, Octubre 2004.

[5] **MARCK** C. Paulk, *Extreme Programming from a CMM Perspective*, Revista FOCUS, Vol. 1, No.1, Pág. 8, December 2002

[6] **ANONIMO**, *Detalles de Metodología Xp*, http://www.extremeprogramming.org/, 15 de Diciembre de 2012.

[7] **ANONIMO,** Extreme Programming: A Gentle Introduction, http://www.extremeprogramming.org/, 15 de Diciembre de 2013

[8] **CALDERON** Amaro, *Metodologías Ágiles*, http://seccperu.org/files/Metodologias%20Agiles.pdf**,** 15 de Diciembre de 2013

[9] **CASTILLO** Oswaldo, **FIGUEROA** Daniel, **SEVILLA** Héctor. Programación Extrema, http://programacionextrema.tripod.com/index.htm, 10 de diciembre 2013

[10] **VALVERDE** David, *Introducción a la Programación Extrema (XP)*, http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/, 11 de Diciembre 2013