eXtreme Programming / Programación Extrema

Introducción a Extreme Programming

Gerardo Fernández Escribano.

Trabajo realizado para la asignatura de Ingeniería de Software II.

Ingeniería Informática.

<gfe@airtel.net>

1. ¿Qué es eXtreme Programming ?

XP *(eXtreme Programming*) es una nueva disciplina para el desarrollo de software, que ha irrumpido recientemente con gran revuelo en el maremágnum de métodos, técnicas y metodologías existentes. Concretando más, se trata de una metodología «ligera», en contraposición a las metodologías «pesadas» como Métrica.

La Programación Extrema es una metodología de desarrollo de software que se basa en la simplicidad, la comunicación y la retroalimentación o reutilización del código desarrollado (reciclado de código). Cuenta con tan solo 6 años de vida, pero con un gran respaldo por parte de grandes empresas cómo la **Ford**, **DaimlerChrysler**, **First Union National Bank -USA-**, etc., que lo que buscan en definitiva es la reducción de costes.

1.1 Un cambio en la manera de programar

El software diseñado para ser simple y elegante no es menos valioso que aquel que es complejo y difícil de mantener. ¿Es cierto esto? XP se basa en esta idea. Un proyecto típico consume del orden de 20 veces más en recursos humanos que en hardware. Pongamos un ejemplo, si un proyecto cuesta 2 millones de dólares (o euros) al año en programadores, costará en equipos (mantenimiento, etc.) en torno a los 100.000 dólares (o euros).Supongamos por un momento que somos programadores listos y encontramos la forma de ahorrar a la compañía un 20 % de los gastos en los equipos, explotando nuestro conocimientos acerca de la arquitectura, redes, etc. Pero esto, hará que el código sea complejo y difícil de mantener. Sin embargo, supongamos ahora que en lugar de esto, construimos nuestros programas de forma que sean sencillos de entender y actualizar; consiguiendo un ahorro de personal en torno a un 10 %. Esta segunda opción gustará, sin duda, mucho más al cliente (concepto básico de proporciones).Otro aspecto a tener en cuenta es la localización de los temidos bugs, o fallos en el programa. XP pone especial hincapié en el testeo de nuestros programas. Crearemos test antes de la implementación, durante y cuando hayamos concluido. Cuando se detecte un fallo, se crearán nuevos test. No puede aparecer dos veces el mismo fallo. Una buena noticia para los usuarios será la apertura de los programadores ante los cambios en los requerimientos (XP es una mentalidad de trabajo). Los usuarios nos notificarán los cambios que sean necesarios para que nuestro sistema se adapte perfectamente a sus necesidades. Lo más importante es la calidad del software, por mucho que le mostremos a un usuario lo bonito de nuestros bucles y tabulaciones en el código fuente, jamás llegará a comprender el esfuerzo realizado en la construcción del mismo.

Planteemos el siguiente problema. Deseamos construir un puente sobre un río. Dicho puente comienza a construirse desde ambas orillas, con el fin de encontrarse en el punto medio. Pero cuando este hecho se produce, nos damos cuenta de que existen unos 60 cm de diferencia, por ejemplo. De la misma manera aplicamos esto al software, no pueden producirse estos problemas a la hora de la integración.

XP surge como solución a estos problemas. Se basa en observar qué es lo que hace que el desarrollo de un programa sea rápido o lento. XP es una metodología importante por dos razones. Primero y principal, porque constituye un método de control para las actividades de desarrollo de software que se han convertido en métodos operativos estándar. Y segundo, es una de las pocas y nuevas

metodologías ligeras desarrolladas para reducir el coste del software. XP va un paso más allá, definir un proceso simple y satisfactorio para la implementación de software.

2. Introducción a la metodología XP

Podríamos decir que XP nace «oficialmente» hace cinco años en un proyecto desarrollado por Kent Beck en *DaimlerChrysle*r, después de haber trabajado varios años con Ward Cunningham en busca de una nueva aproximación al problema del desarrollo de software que hiciera las cosas más simples de lo que nos tenían acostumbrados los métodos existentes. Para muchos, XP no es más que sentido común.

Kent definió cuatro grandes tareas a realizar en el desarrollo de todo proyecto: planificación, diseño, desarrollo y pruebas; teniendo siempre presente las cuatro características básicas que debe reunir un programador XP: simplicidad en el desarrollo, comunicación entre las partes implicadas, realimentación para poder reutilizar y coraje.

2.1 ¿En qué consiste XP? Objetivos

El objetivo principal que persigue XP es la satisfacción del cliente. Esta metodología fue diseñada para proporcionar el software que el cliente necesita cuando lo necesite. Deberemos responder de forma rápida a los cambios en las necesidades del cliente, incluso cuando estos cambios se produzcan al final del ciclo de vida de dicho software (simplicidad y realimentación).

El segundo objetivo es potenciar al máximo el trabajo en equipo. Tanto los Jefes de Proyecto, como los clientes y desarrolladores, son parte del equipo encargado de la implementación de software de calidad (comunicación). Esto implicará que los diseños deberán ser claros y sencillos. Y los clientes deberán disponer de versiones operativas cuanto antes para poder participar en el proceso creativo mediante sus sugerencias y aportaciones.

- El código será revisado continuamente, mediante la **programación en parejas** (dos personas por máquina).
- Se harán pruebas todo el tiempo, no sólo de cada nueva clase **(pruebas unitaria**s) sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos **(pruebas funcionale**s).
- Las pruebas de integración se efectuarán siempre, antes de añadir cualquier nueva clase al proyecto, o después de modificar cualquiera existente **(integración continu**a), para lo que nos serviremos de *frameworks de testin*g, como el xUni ó *JUnit*.
- Se (re)diseñará todo el tiempo **(refactoring**), dejando el código siempre en el estado más simple posible.
- Las iteraciones serán radicalmente más cortas de lo que es usual en otros métodos, de manera que nos podamos beneficiar de la retroalimentación tan a menudo como sea posible.

«Todo en el software cambia. Los requisitos cambian. El diseño cambia. El negocio cambia. La tecnología cambia. El equipo cambia. Los miembros del equipo cambian. El problema no es el cambio en sí mismo, puesto que sabemos que el cambio va a suceder; el problema es la incapacidad de adaptarnos a dicho cambio cuando éste tiene lugar.» Kent Beck.

2.2 Las cuatro variables

XP define cuatro variables para cualquier proyecto software: *coste*, *tiempo*, *calidad* y *alcance*.

Además, especifica que, de estas cuatro variables, sólo tres de ellas podrán ser fijadas por las fuerzas externas al proyecto(clientes y jefes de proyecto), mientras que el valor de la variable libre será establecido por el equipo de desarrollo en función de los valores de las otras tres.

¿Qué es lo novedoso aquí? Que normalmente los clientes y jefes de proyecto se creen capaces de fijar de antemano el valor de *todas* las variables: *«Quiero estos requisitos satisfechos para el día uno del mes que viene, para lo cual contáis con este equipo. ¡Ah, y ya sabéis que la calidad es lo primero!»*

Claro, cuando esto ocurre –y ocurre bastante a menudo, por desgracia– la calidad es lo primero que se esfuma de la ecuación. Y esto por una sencilla razón, que frecuentemente se ignora: nadie es capaz de trabajar bien cuando está sometido a mucha presión. XP hace a las cuatro variables visibles para todo el mundo–programadores, clientes y jefes de proyecto–, de manera que se

pueda jugar con los valores de la entrada hasta que la cuarta variable tenga un valor que satisfaga a todos (pudiendo escoger otras variables diferentes a controlar, por supuesto). Ocurre además que las cuatro variables no guardan entre sí una relación tan obvia como a menudo se quiere ver.

En este sentido, ya es de sobra conocido el dicho de que «nueve mujeres no pueden tener un hijo en un mes». XP hace especial énfasis en equipos de desarrollo pequeños (diez o doce personas como mucho) que, naturalmente, se podrán ir incrementando a medida que sea necesario, pero no antes, o los resultados serán generalmente contrarios a lo esperado.

Sin embargo, no pocos jefes de proyecto parecen ignorarlo cuando afirman, henchidos de orgullo, que *su* proyecto involucra a 150 personas, como un marchamo de prestigio que adherir a su currículo. Sí es bueno, en cambio, incrementar el **coste** del proyecto en aspectos como máquinas más rápidas, más especialistas técnicos en determinadas áreas o mejores oficinas para el equipo de desarrollo.

Con la **calidad** también sucede otro fenómeno extraño: frecuentemente, *aumentar la calidad conduce a que el proyecto pueda realizarse en menos tiempo*. En efecto, en cuanto el equipo de desarrollo se habitúe a realizar pruebas intensivas(y trataremos este punto muy pronto, pues es el pilar básico de XP) y se sigan estándares de codificación, poco apoco comenzará a avanzar mucho más rápido de lo que lo hacía antes, mientras la calidad del proyecto se mantiene asegurada –por las pruebas– al 100%, lo que conlleva mayor confianza en el código y, por tanto, mayor facilidad para adaptarse al cambio, sin estrés, lo que hace que se programe más rápido ... y así sucesivamente.

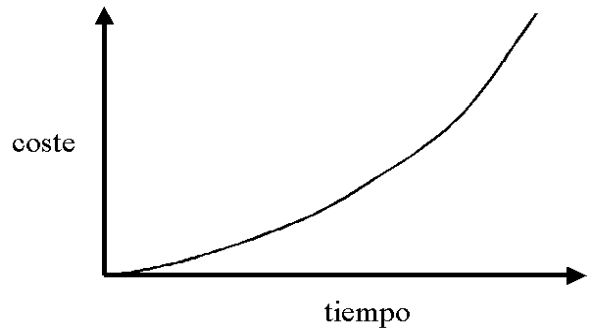
Frente a esto, está la tentación de sacrificar la calidad interna del proyecto –la que es apreciada por los programadores–para reducir el tiempo de entrega del proyecto, en la confianza de que la calidad externa –la que notan los clientes–-no se vea demasiado afectada. Sin embargo, ésta es una apuesta a muy corto plazo, que suele ser una invitación al desastre, pues obvia el hecho fundamental de que *todo el mundo trabaja mucho mejor cuando le dejan hacer trabajo de calida*d.

No tener esto en cuenta conduce a la desmoralización del equipo y, con ello, a la larga, a la ralentización del proyecto mucho más allá del **tiempo** que hubiera podido ganarse al principio con esta reducción de calidad. En cuanto al **alcance** del proyecto, es una buena idea dejar que sea esta la variable libre, de manera que, una vez fijadas las otras tres, el equipo de desarrollo determinaría el alcance mediante:

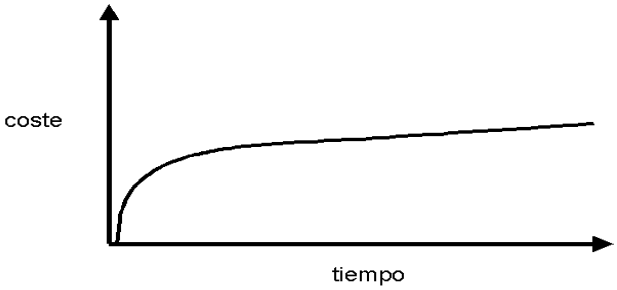
- La estimación de las tareas a realizar para satisfacer los requisitos del cliente.
- La implementación de los requisitos más importantes primero, de manera que el proyecto tenga en cada instante tanta funcionalidad como sea posible.

2.3 El coste del cambio

Es importante al menos reseñar una de las asunciones más importantes e innovadoras que hace XP frente a la mayoría de los métodos conocidos, y es la referida al coste del cambio. En efecto, siempre ha sido una verdad universal el hecho de que el coste del cambio en el desarrollo de un proyecto se incrementaba exponencialmente en el tiempo:



Lo que XP propugna es que esta curva ha perdido validez y que con una combinación de buenas prácticas de programación y tecnología es posible lograr que la curva sea la contraria. Con la metodología XP, se pretende conseguir:



Si decidimos emplear XP como proceso de desarrollo de software, deberemos adoptarlo basándonos en dicha curva.

La idea fundamental aquí es que, en vez de diseñar para el cambio, diseñaremos tan sencillo como sea posible, para hacer sólo lo que sea imprescindible en un momento dado, pues la propia simplicidad del código, y, sobretodo, los *tests* y la integración continua, hacen posible que los cambios puedan ser llevados a cabo tan a menudo como sea necesario.

2.4 Ciclo de vida

Si, como se ha demostrado, los largos ciclos de desarrollo de los métodos tradicionales son incapaces de adaptarse al cambio, tal vez lo que haya que hacer sea ciclos de desarrollo más cortos. Esta es una de las ideas centrales de XP (**gráfico 1**).

3. Fases de la metodología XP

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

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 debe ser hecho 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 –que lo es–,sino que sobre todo es un método de gestión de proyectos software.

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 los test de aceptación.

Uno o más de uno de estos test se utilizarán para verificar que las historias de usuario han sido implementadas correctamente.

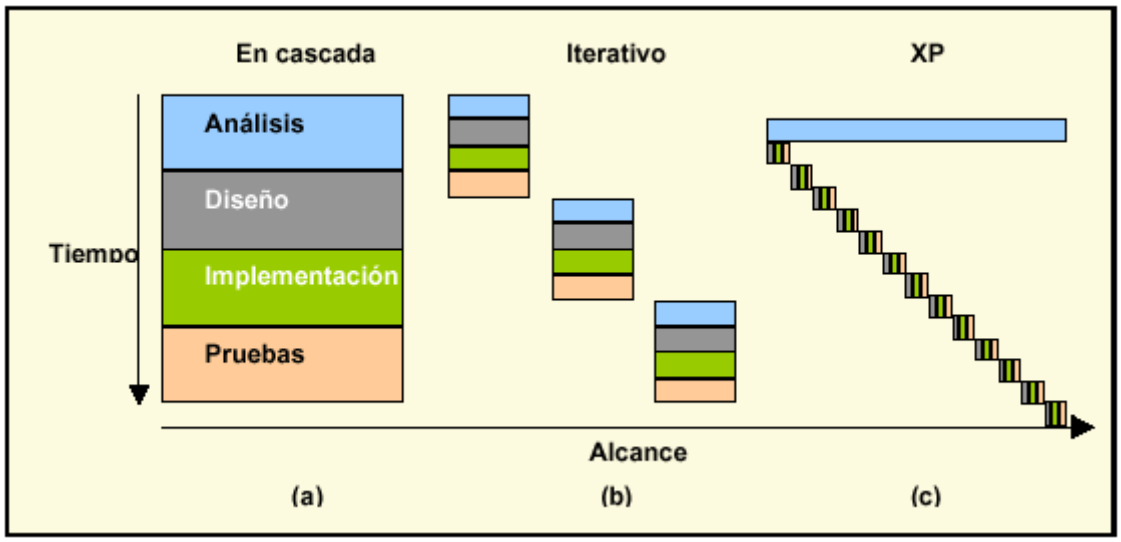


Gráfico 1. Evolución de los largos ciclos de desarrollo en cascada (a) a ciclos más cortos (b) y a la mezcla que hace XP (c)

Una de las mayores equivocaciones a cerca del uso de las historias de usuario es la diferencia que existe entre estas y la tradicional especificación de requisitos.

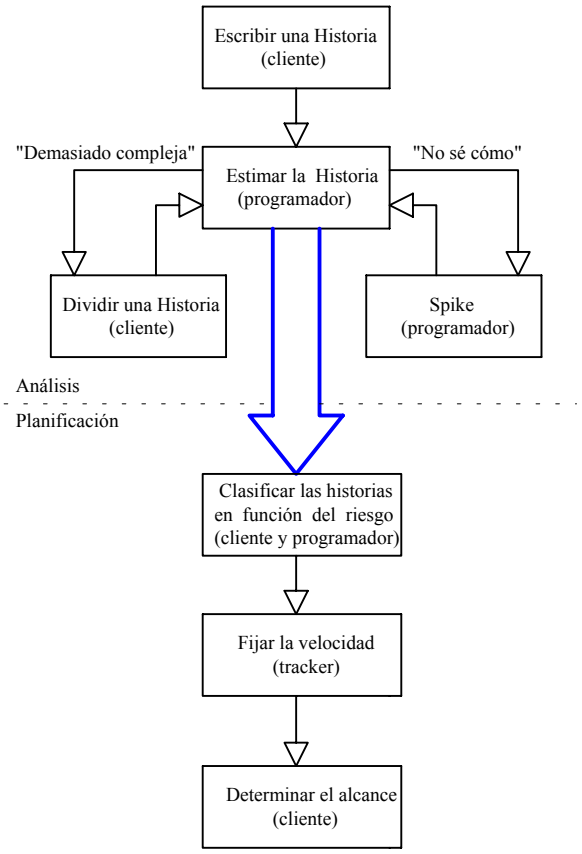
La principal diferencia es el nivel de detalle. Las historias de usuario solamente proporcionaran los detalles sobre la estimación del riesgo y cuánto tiempo conllevará la implementación de dicha historia de usuario. El nivel de detalle de las historias de usuario debe ser el mínimo posible que permita 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.

3.1.2 Se crea 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. Una semana ideal es cuánto tiempo costaría implementar dicha historia si no tenemos nada más que hacer, incluyendo la parte de test 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:



3.1.3 Se controla la velocidad del proyecto

La velocidad del proyecto es una medida de cuan rápido se está desarrollando. La velocidad de 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**). Cuando se realiza una planificación por alcance se divide el numero total de semanas entre la velocidad de proyecto para determinar cuántas iteraciones estarán disponibles.

3.1.4 Se divide el proyecto en iteraciones

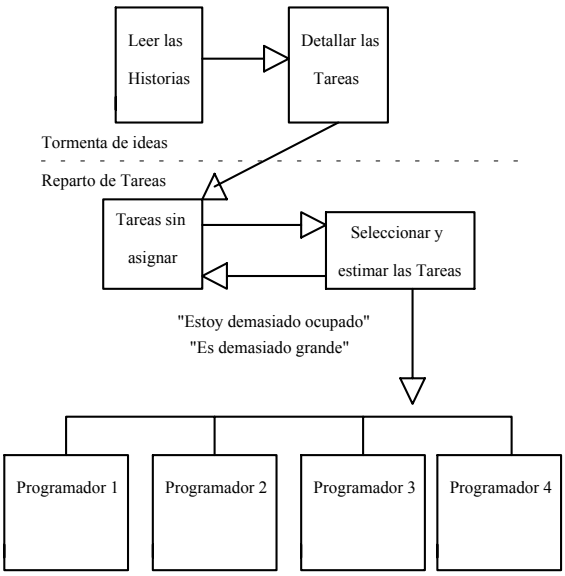
Cada iteración corresponde a un periodo de tiempo de desarrollo del proyecto de entre una y tres semanas. De esta forma, un proyecto, se divide en una docena de iteraciones, más o menos. Al principio de cada iteración se debería convocar una reunión para trazar el plan de iteración correspondiente.

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.

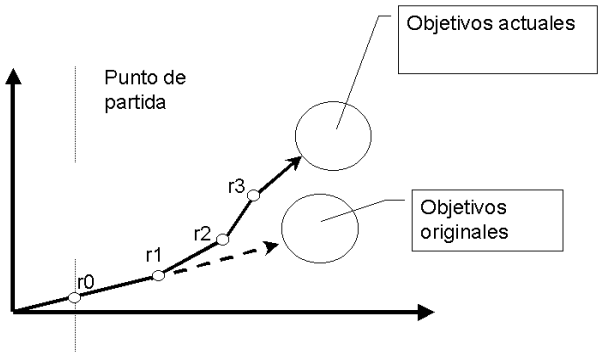
3.1.5 Al comienzo de cada iteración se traza el plan de iteración

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



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 usuario más importantes para el cliente, estaremos haciendo lo máximo posible por nuestro cliente y la dirección. En cada iteración iremos alcanzado unos metas:



3.1.6 Se rota al personal

La rotaciones evitarán que las personas se conviertan en si 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.

3.1.7 Cada día se convoca una reunión de seguimiento

La comunicación entre las diferentes partes que interviene 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 estas sean frecuentes, de poca duración y a 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.

3.1.8 Corregir la propia metodología XP cuando falla

Deberemos corregir el proceso cuando éste falle. Cuando comencemos 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 cambiemos lo que no nos guste, sino que cambiemos aquello que no funciona con nuestro problema en particular.

3.2 Diseño

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

3.2.1 Simplicidad

La simplicidad es la llave

Siempre costará menos tiempo de 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, deberías replantearla (divide y vencerás). Así, cualquier cambio y modificación será mucho más sencillo. En ocasiones, realizar un diseño sencillo puede resultar una tarea especialmente difícil

3.2.2 Elegir una metáfora para el sistema

Una metáfora para el sistema es una historia que todo el mundo puede contar a cerca de cómo el sistema funciona (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", 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, cuando se representa la segmentación en un procesador con una cadena de montaje de vehículos. Debemos encontrar una imagen de aquello que pretendemos representar en el mundo real. Así podremos tener una imagen mental más clara.

3.2.3 Usar tarjetas CRC (Cargo o clase, Responsabilidad y Colaboración) 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.

3.2.4 Crear soluciones puntuales para reducir riesgos

Un programa *Spike*, es un programa muy simple que explora una posible solución al problema. 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–.

3.2.5 No se añadirá funcionalidad en las primeras etapas

Debemos evitar caer en la tentación de ir añadiendo funcionalidades según se nos vayan ocurriendo, aun 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 sabemos si será utilizado.

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..

3.3 Desarrollo

Esta etapa debe reunir las siguientes características o cualidades:

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 los test. Estos test 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.

3.3.2 Se debe escribir código de acuerdo a los estándares

El código a 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.

3.3.3 Desarrollar la unidad de pruebas primero

Cuando los test son creados antes que el código, la implementación del código será mucho más rápida. El tiempo empleado en desarrollar un test 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 test ayuda al programador a tener una visión a cerca del cómo, en definitiva, del comportamiento del programa. Además, aún estamos a tiempo de dar marcha a tras, ya que el programador no a concluido la implementación. Esta manera de trabajar resulta especialmente beneficiosa en el diseño de complicados sistemas software.

«Cualquier característica de un programa para la que no haya un test automatizado, 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 hacemos *test*s, no estaremos haciendo XP.

Para ello, deberemos emplear algún *framework* de *testing* automático, como JUnit [Junit-www] o cualquiera de sus versiones para diferentes lenguajes. No sólo eso, sino que escribiremos los *tests* antes incluso que la propia clase a probar.

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.

3.3.5 Sólo una pareja se encargará de integrar el código

En esta etapa pueden aparecer problemas debidos a la integración de los módulos que se han desarrollado y no han sido testeados todavía. Las unidades de test se encargarán de verificar la corrección de dichos módulos. Estos test deberán ser completos, en el sentido de que, un fallo el los mismo podría derivar que determinados errores pasaran inadvertidos.

3.3.6 Integrar frecuentemente

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 en las pruebas para su test 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.

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 cambió en el mismo siempre y cuando beneficie a la arquitectura del mismo. La responsabilidad del funcionamiento recaerá sobre el equipo al completo.

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.

Haz el trabajo, hazlo bien, y entonces hazlo rápido.

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. Se trabajará un máximo de 40 horas semanales. Nadie es capaz de trabajar 60 horas a la semana y hacerlo con calidad.

3.4 Pruebas

3.4.1 Todo el código debe ir acompañado de 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 existan 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, trasladado 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.

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.

3.4.2 Todo el código debe pasar las unidades de pruebas antes de ser implantado

Las unidades de test serán incluidas junto con el código que verifican dentro del repositorio. El código no se considerará completo si este no consta de su unidad de test correspondiente.

El código será implantado cuando supere sus correspondientes unidades de test.

3.4.3 Ante un fallo, una unidad de pruebas

Cuando se detecte un fallo, crearemos 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.

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 testear 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 a cerca 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 debe ser los 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.

4. Comparativa con las Metodologías Tradicionales y conclusiones

XP ha levantado gran revuelo en la comunidad de ingeniería del software. Que hay opiniones para todos los gustos lo prueban, de manera informal, los resultados de la encuesta llevada a cabo por IBM [IBM-encuesta] (**gráfico 2**).

Especialmente duras son las críticas al *pair programming* –sobre todo desde la perspectiva de los jefes de proyecto, pero también sin duda de muchos programadores con un acusado sentimiento de posesión del código («esto lo hice yo, y además soy tan bueno programando y tengo tal domino de los *idiomas* del lenguaje que sólo yo puedo entenderlo»)–, pero también se habla del mito de las 40 horas semanales(?), de que «eso de los *tests* está muy bien si tienes tiempo de sobra, pero son un lujo imposible bajo las condiciones del mercado»...

También hay quien dice, y esta crítica lleva más razón que las anteriores, que XP sólo funciona con gente buena, es decir, gente como Kent Beck, que son capaces de hacer un buen diseño, sencillo y a la vez –y acaso precisamente por ello– fácilmente extensible, a la primera. Aunque sólo fuera por ello, merecería la pena echar al menos un vistazo a este nuevo y excitante método de desarrollo software. Extreme Programming es más una filosofía de trabajo que una metodología. Por otro lado, ninguna de las prácticas defendidas por XP son una invención del método; todas ellas ya existían, y lo que XP ha hecho ha sido ponerlas todas juntas y comprobar que funcionaban.

El Extreme Programming (XP) funciona mejor con pocos desarrolladores. Ejemplo: Las rotaciones en

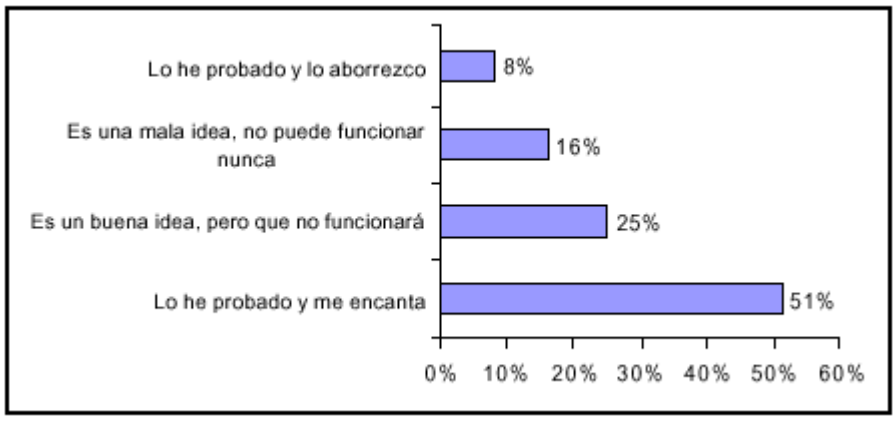


Gráfico 2. Encuesta de IBM (octubre de 2000): ¿Qué opina de eXtreme Programming?

grupos de más de 10 personas podrían ser un verdadero problema.

Extreme Programming surgió como contrapartida a las metodologías clásicas o tradicionales, añadiendo una nueva forma de hacer las cosas. Lo más importante es extraer las ideas principales.

Las metodologías tradicionales para el desarrollo del software imponen un proceso disciplinado con el objetivo de hacer el trabajo más predecible, eficiente y planificado. El único inconveniente que se le ha atribuido es de ser "orientadas a documentos", es decir, demasiado burocráticas. Las metodologías livianas o ágiles, como XP, están orientadas a las personas más que a los procesos.

Hay que tener en cuanta que en el desarrollo del software, el diseño es la actividad predominante (en XP el diseño es una actividad diaria). XP supone:

- Las personas son claves en los procesos de desarrollo de software.
- Los programadores son profesionales responsables, no requieren de supervisión.
- Los procesos se aceptan y acuerdan, no se imponen.
- Desarrolladores y gerentes comparten el liderazgo del proyecto.
- El trabajo de los desarrolladores con las personas que tienen la experiencia en el negocio es regular, no eventual.

Conviene recordar que *ninguna metodología hará el trabajo por ti, porque ninguna metodología trabaja sola*.

5. Bibliografía

[Beck 2000] Kent Beck. *Extreme Programming Explained: Embrace Chang*e. Addison Wesley Longman, 2000.
[Jeffries et al 2000] Ronald E. Jeffries et al. *Extreme Programming Installe*d. Addison-Wesley, 2000.
[Beck and Fowler] Kent Beck, Martin Fowler. *Planning Extreme Pro -gramming*. Addison-Wesley.
[Wake] William C. Wake. Extreme Programming Explored
[Fowler 1999] Martin Fowler. *Refactoring : Improving the Design of Existing Cod*e. Addison-Wesley, 1999.
[Beck 1999] Kent Beck. *Embracing Change with Extreme Programmin*g. Computer (revista de la *IEEE Computer Societ*y).Vol. 32, No. 10. Octubre1999, pp. 70-77

Y también en diferentes páginas web:

– http://www.xprogramming.com

– http://www.extremeprogramming.org

– www-106.ibm.com/developerworks/java/library /java-pollresults/xp.html. *Java poll results: What are your thoughts on Extreme Programming?* Encuesta de IBM de octubre de 2000.

– http://www.computer.org/seweb/Dynabook /Index.htm. *eXtreme Programming. Pros and Cons. What Questions remain?* El primer «dynabook» de la *IEEE Computer Society* se dedicó a XP, con una serie de artículos relacionados.