Programación Extrema

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 se produce al utilizar otras 'metodologías pesadas'.

El autor de la XP es **Kent Beck**, entre otros, que con su larga experiencia como programador eligió las mejores características de las metodologías y profundizó en las relaciones de éstas y como se reforzaban unas a otras. Por tanto, la XP no se basa en principios nuevos, sino que todas, o casi todas, sus características ya se conocen dentro de la ingeniería del software, las cuales se complementan para minimizar los tópicos problemas que pueden surgir en todo desarrollo de proyectos software.

Objetivos de la programación extrema

En la programación extrema se da por supuesto que es imposible prever todo antes de empezar a codificar. Es imposible capturar todos los requisitos del sistema, saber qué es todo lo que tiene que hacer ni hacer un diseño correcto al principio. Es bastante normal hacer un diseño, ponerse a codificar, ver que hay faltantes o errores en el diseño, empezar a codificar fuera del diseño y al final el código y el diseño, o no se parecen, o hemos echado un montón de tiempo en cambiar la documentación de diseño para que se parezca al código.
En vez de tratar de luchar contra todo eso, lo asume y busca una forma de trabajar que se adapte fácilmente a esas circunstancias. Básicamente la idea de la programación extrema consiste en trabajar estrechamente con el cliente, haciéndole mini-versiones con mucha frecuencia (cada dos semanas). En cada mini-versión se debe hacer el mínimo de código y lo más simple posible para que funcione correctamente. El diseño se hace sobre la marcha, haciendo un mini-diseño para la primera mini-versión y luego modificándolo en las siguientes mini-versiones. Además, no hay que hacer una documentación para el diseño, no hay mejor documentación que el mismo código. El código, por tanto, también se modifica continuamente de mini-versión en mini-versión, añadiéndole funcionalidad y extrayendo sus partes comunes.

El objetivo principal de la XP es la satisfacción del cliente. Se le trata de dar al cliente lo que quiere y cuando quiere. Por tanto, se debe responder rápidamente a las necesidades del cliente, aunque realice cambios en fases avanzadas del proyecto. Como metodología Ágil que es, se pueden producir modificaciones de los requisitos del proyecto a lo largo de su desarrollo, sin que esto produzca un *buen dolor de cabeza*.

Otro de los objetivos es el trabajo en grupo. Tanto los jefes del proyecto, clientes y desarrolladores forman parte del equipo y deben estar involucrados en el desarrollo.

Los pasos a seguir en un proyecto

En un proyecto usando programación extrema se siguen más o menos los siguientes pasos:
El cliente junto al equipo de desarrollo definen qué es lo que se quiere hacer. Para ello utilizan las **historias de usuario**. Una historia de usuario en un texto de una o dos frases en las que se dice algo que debe hacer el sistema. Es más extensa que un requisito (que suele ser una frase corta) y menos que un caso de uso (que puede ser de una o dos páginas). Se evalua para cada historia de usuario el tiempo que puede llevar, que debe ser corto, de aproximadamente una semana. Un programador puede estimar con cierta fiabilidad un trabajo que le lleve unos días, pero la estimación es menos fiable si es de un plazo superior a una semana. Si es más largo, hay que partir la historia en otras más pequeñas. Luego se ordenan en el orden en que se van a desarrollar y se establecen las mini-versiones, de forma que cada mini-versión implementa varias de las historias de usuario.
Toda esta planificación va a ser, por supuesto, inexacta. No se puede saber todo lo que va a ser necesario ni evaluar los tiempos correctamente. La planificación deberá revisarse y modificarse continuamente a lo largo del proyecto. Las historias de usuario se modificarán, se quitarán o se añadirán nuevas sobre la marcha. Puesto que el cliente estará presente día a día durante todo el proyecto, verá el efecto y el esfuerzo necesario para las modificaciones pedidas y sabrá evaluar si merecen o no la pena.
Para las primeras historias que se van a implementar, se define una prueba para ver si la versión cumple perfectamente con la historia. Estas pruebas deben ser automáticas, de forma que haya un programa de pruebas que ejecutemos y nos diga si la mini-versión es o no correcta.
Los programadores se ponen por parejas (dos personas en el mismo ordenador) para codificar esas historias. Primero codifican el programa de pruebas y ven que falla (¡ El código todavía no está hecho !). Luego piensan cómo van a

implementar la historia y hacen sus propios programas de prueba (también automáticos). Codifican sus programas de prueba y ven que también fallan. Luego se ponen a codificar hasta que se pasen con éxito todas las pruebas. En su código hacen de la forma más sencilla posible lo mínimo imprescindible para que se pasen las pruebas automáticas.
Las parejas de programadores se intercambian con frecuencia, de forma que todos acaban trabajando con todos. El trabajo por parejas haciendo intercambios tiene las siguientes ventajas:

- Cuatro ojos ven más que dos. Al trabajar de dos en dos, el código será de mayor calidad desde el mismo momento de crearlo y tendrá menos fallos.
- Los programadores novatos aprenderán de los expertos al emparejarse con ellos.
- Si una pareja realiza un trozo de código susceptible de ser reutilizado en el proyecto, hay dos programadores que lo saben y que lo reutilizarán cuando puedan (ya que saben cómo funciona), enseñándolo a sus nuevos compañeros. De esta manera el conocimiento del código ya hecho se propaga de forma natural entre todos los programadores del equipo.
- El estilo de programación tiende a unificarse.

Cuando una nueva pareja va a realizar un nuevo código para una historia de usuario, puede que uno de ellos recuerde haber hecho un trozo de código en otro lado que podría reutilizar. Esta pareja irá a ese trozo de código y lo reutilizará, modificándolo si es necesario. Después de modificarlo, deben asegurarse que se siguen pasando las pruebas automáticas que se hicieron en su momento, así como añadir nuevas pruebas para comprobar las modificaciones que han hecho.
Si una pareja al reutilizar código ya hecho ve que es mejorable, lo mejoran, pasando las pruebas automáticas después. Si al reutilizar el código ya hecho descubren un error que las pruebas automáticas no detectan, añaden pruebas capaces de detectar el error y lo corrigen.
El código, por tanto, no es de nadie. Cualquier pareja puede tocar el código ya hecho por otras personas si lo necesita, con la condición de que después de tocarlo las pruebas automáticas sigan pasándose correctamente y que añadan sus propias pruebas automáticas para las correcciones realizadas.
Todos los días se hace una pequeña reunión a primera hora de la mañana con todo el equipo en la que se comentan problemas, código que se está realizando, historias de usuario terminadas, etc.
Cada vez que se consigue codificar y que funcione una historia de usuario, se le da al cliente para que la vea, la pruebe y añada las posibles modificaciones para las siguientes mini-versiones. Cuando se realiza un mini-versión completa (compuesta por varias de las historias de usuario), incluso se entrega al usuario final para que empiece a trabajar con ella y reportar inicidencias o mejoras.

Este ciclo se va repitiendo una y otra vez hasta que el cliente se de por satisfecho y cierre el proyecto. Cómo se ve, no se ha hecho documentación. En algún sitio he leido que incluso la planificación inicial debe hacerse escribiendo cada historia de usuario en un tarjeta, haciendo dos montones, las que ya están hechas y las que no. Se pueden tirar las tarjetas, añadir nuevas o cambiar las que ya hay. El número de tarjetas en cada montón nos da una idea exacta de cuánto hay hecho del proyecto.

Valores de la programación extrema

Para garantizar el éxito de un proyecto, los autores de XP han considerado como fundamentales cuatro valores:

- **Comunicación**. Muy importante. La XP ayuda mediante sus prácticas a la comunicación entre los integrantes del grupo de trabajo: jefes de proyecto, clientes y desarrolladores.
- **Sencillez**. Los programas deben ser los más sencillos posibles y tener la funcionalidad necesaria que se indican en los requisitos. No hay que añadir algo que no se necesite *hoy*. Si se necesita añadir más funcionalidad *mañana* pues ya se hará entonces.
- **Retroalimentación**. Las pruebas que se le realizan al software nos mantiene informados del grado de fiabilidad del sistema.
- **Valentía**. Asumir retos, ser valientes ante los problemas y afrontarlos. El intentar mejorar algo que ya funciona. Aunque gracias a las pruebas unitarias no existe el riesgo de 'meter la pata'.

Prácticas básicas de la programación extrema

- **Equipo completo:** Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.
- **Planificación:** Se hacen las historias de usuario y se planifica en qué orden se van a hacer y las mini-versiones. La planificación se revisa continuamente.
- **Test del cliente:** El cliente, con la ayuda de los desarrolladores, propone sus propias pruebas para validar las mini-versiones.
- **Versiones pequeñas:** Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.
- **Diseño simple:** Hacer siempre lo mínimo imprescindible de la forma más

sencilla posible. Mantener siempre sencillo el código.

- **Pareja de programadores:** Los programadores trabajan por parejas (dos delante del mismo ordenador) y se intercambian las parejas con frecuencia (un cambio diario).
- **Desarrollo guiado por las pruebas automáticas:** Se deben realizar programas de prueba automática y deben ejecutarse con mucha frecuencia. Cuantas más pruebas se hagan, mejor.
- **Mejora del diseño:** Mientras se codifica, debe mejorarse el código ya hecho con el que nos crucemos y que sea susceptible de ser mejorado. Extraer funcionalidades comunes, eliminar líneas de código innecesarias, etc.
- **Integración continua:** Deben tenerse siempre un ejecutable del proyecto que funcione y en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse. Es un error mantener una versión congelada dos meses mientras se hacen mejoras y luego integrarlas todas de golpe. Cuando falle algo, no se sabe qué es lo que falla de todo lo que hemos metido.
- **El código es de todos:** Cualquiera puede y debe tocar y conocer cualquier parte del código. Para eso se hacen las pruebas automáticas.
- **Normas de codificación:** Debe haber un estilo común de codificación (no importa cual), de forma que parezca que ha sido realizado por una única persona.
- **Metáforas:** Hay que buscar unas frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que sólo con los nombres se pueda uno hacer una idea de qué es lo que hace cada parte del programa. Un ejemplo claro es el "recolector de basura" de java. Ayuda a que todos los programadores (y el cliente) sepan de qué estamos hablando y que no haya mal entendidos.
- **Ritmo sostenible:** Se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días. Al tener claro semana a semana lo que debe hacerse, hay que trabajar duro en ello para conseguir el objetivo cercano de terminar una historia de usuario o mini-versión.

"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