Reglas y Prácticas en eXtreme Programming

Ing. José Joskowicz

El presente trabajo ha sido realizado en el marco de la asignatura “Nuevas Técnicas de Desarrollo de Software en Ingeniería Telemática”, del Doctorado de Ingeniería Telemática de la Universidad de Vigo, España.

“Cualquiera puede hacer complicado algo simple.
La creatividad consiste en hacer simple lo complicado”

Charles Mingus

Indice

Abstract

"Extreme Programming" o "Programación Extrema" es una de las llamadas metodologías ágiles de desarrollo de software más exitosas y controversiales de los tiempos recientes. El presente trabajo presenta un resumen de las características más destacables de esta metodología, incluyendo las fases de su ciclo de vida, las reglas y prácticas propuestas, sus valores y su aplicabilidad. Finalmente se presentan algunas críticas, y se cita el resultado de encuestas recientes realizadas acerca del uso y éxito de las prácticas de ésta nueva metodología.

1. Introducción

Extreme Programming (XP) surge como una nueva manera de encarar proyectos de software, proponiendo una metodología basada esencialmente en la simplicidad y agilidad. Las metodologías de desarrollo de software tradicionales (ciclo de vida en cascada, evolutivo, en espiral, iterativo, etc.) aparecen, comparados con los nuevos métodos propuestos en XP, como pesados y poco eficientes. La crítica más frecuente a estas metodologías "clásicas" es que son demasiado burocráticas. Hay tanto que hacer para seguir la metodología que, a veces, el ritmo entero del desarrollo se retarda. Como respuesta a esto, se ha visto en los últimos tiempos el surgimiento de "Metodologías Ágiles". Estos nuevos métodos buscan un punto medio entre la ausencia de procesos y el abuso de los mismos, proponiendo un proceso cuyo esfuerzo valga la pena.

Los métodos ágiles cambian significativamente algunos de los énfasis de las metodologías "clásicas" [1]:

- Los métodos ágiles son adaptables en lugar de predictivos. Los métodos "clásicos" tienden a intentar planear una gran parte 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.
- Los métodos ágiles son orientados a la gente y no orientados al proceso. El objetivo de los métodos "clásicos" es definir un proceso que funcionará bien independientemente de quien lo utilice. Los métodos ágiles afirman que ningún proceso podrá nunca maquillar las habilidades del equipo de desarrollo, de modo que el papel del proceso es apoyar al equipo de desarrollo en su trabajo.

XP es una de las llamadas metodologías ágiles de desarrollo de software más exitosas de los tiempos recientes. La metodología propuesta en XP está diseñada para entregar el software que los clientes necesitan en el momento en que lo necesitan. XP alienta a los desarrolladores a responder a los requerimientos cambiantes de los clientes, aún en fases tardías del ciclo de vida del desarrollo [2].

La metodología también enfatiza el trabajo en equipo. Tanto gerentes como clientes y desarrolladores son partes del mismo equipo dedicado a entregar software de calidad.

XP fue introducida como metodología ágil de desarrollo de software sobre finales de los 1990s. Uno de los conocidos "caso de éxito" fue publicado a fines de 1998, cuando Kent Beck [3] introdujo la nueva metodología en el proyecto de desarrollo denominado C3 (Chrysler Comprehensive Compensation) para la firma Chrysler [4].

2. Agile Manifesto

En febrero de 2001, en las montañas Wasatch de Utah, se reunieron 17 desarrolladores convencidos de que era necesario un cambio en las metodologías "clásicas" de desarrollo de software. Entre ellos se encontraban los creadores de XP, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development y Pragmatic Programming. Juntos proclamaron lo que se ha dado a conocer como el "Manifesto for Agile Software Development", estableciendo en él cuatro principios [5]:

> **"Se valora a los individuos y las interacciones** sobre los procesos y las herramientas
> **Se valora a las aplicaciones que funcionan** sobre la documentación exhaustiva
> **Se valora la colaboración del cliente** sobre las negociaciones contractuales
> **Se valora la respuesta al cambio** sobre el seguimiento de un plan
>
> Esto significa que, sin desconocer el valor de los segundos items, se valoran más los primeros"

Este manifiesto se basa en los siguientes principios:

- Satisfacer al cliente a través de entregas continuas y tempranas es la mayor prioridad.
- Los cambios a los requerimientos son bienvenidos, aún en fases tardías del desarrollo.
- Entregar frecuentemente software que funciona, desde un par de semanas a un par de meses, prefiriendo los periodos más cortos.
- Desarrolladores, gerentes y clientes deben trabajar juntos diariamente, a lo largo del proyecto.
- Construir proyectos alrededor de personas motivadas, dándoles el entorno y soporte que necesitan, y confiando en que realizarán el trabajo.
- El método más eficiente y efectivo de transmitir información entre un equipo de desarrolladores es la conversación frontal (cara a cara).
- Tener software que funciona es la medida primaria del progreso.

- El proceso ágil promueve el desarrollo sostenible. Los sponsors, desarrolladores y usuarios deben ser capaces de mantener un ritmo de trabajo constante en forma permanente a lo largo del proyecto.
- La atención continua a la excelencia técnica y el buen diseño mejoran la agilidad.
- Simplicidad – el arte de maximizar el trabajo que no se debe hacer – es esencial.
- Las mejores arquitecturas, requerimientos y diseños surgen de los equipos auto-organizados.
- A intervalos regulares, el equipo debe reflexionar sobre como ser más efectivos, y ajustar su comportamiento de acuerdo a ello.

Los desarrollos de software ágil, que adoptan los principios del "Agile Manifesto", no son anti-metodológicos. Por el contrario, siguen su metodología, diferente a la de los métodos clásicos de desarrollo. Se trata de lograr un equilibrio, en el que, por ejemplo, la documentación es concreta y útil, y no burocrática y los planes existen, pero reconociendo sus limitaciones en el actual mundo en permanente cambio.

3. Ciclo de vida de Software en XP

Para apreciar los conceptos del ciclo de desarrollo de software en XP introduciremos brevemente los conceptos principales de las metodologías de desarrollo de software tradicionales [6].

3.1. Modelo en cascada

El modelo de cascada tiene sus orígenes en la década de 1970 [7], y se define como una secuencia de actividades bien planificadas y estructuradas. El proceso distingue claramente las fases de especificación de las de desarrollo y éstas, a su vez, de las de testing. Es, seguramente, la metodología más extendida y utilizada. Este modelo se basa fuertemente en que cada detalle de los requisitos se conoce de antemano, previo de comenzar la fase de codificación o desarrollo, y asume, además, que no existirán cambios significativos en los mismos a lo largo del ciclo de vida del desarrollo.

3.2. Modelo incremental

El modelo incremental consiste en un desarrollo inicial de la arquitectura completa del sistema, seguido de sucesivos incrementos funcionales. Cada incremento tiene su propio ciclo de vida y se basa en el anterior, sin cambiar su funcionalidad ni sus interfaces. Una vez entregado un incremento, no se realizan cambios sobre el mismo, sino únicamente corrección de errores. Dado que la arquitectura completa se desarrolla en la etapa inicial, es necesario, al igual que en el modelo en cascada, conocer los requerimientos completos al comienzo del desarrollo.

Respecto al modelo en cascada, el incremental tiene la ventaja de entregar una funcionalidad inicial en menor tiempo.

3.3. Modelo evolutivo

El modelo evolutivo es, en cierta forma, similar al incremental, pero admite que la especificación no esté completamente determinada al comienzo del ciclo de vida. Los requerimientos que estén suficientemente detallados al comienzo darán lugar a un entrega inicial, mientras que los siguientes incrementos serán cambios progresivos que implementen "*deltas*" de especificación de requerimientos. El modelo admite que, si la especificación no es suficientemente clara al principio, puede desarrollarse un prototipo experimental, que tiene como función validar o identificar los requisitos del sistema.

3.4. Modelo espiral

El modelo de espiral, introducido por Barry Bohem a fines de la década de 1980 [8], intenta combinar las ventajas del modelo en cascada con el modelo evolutivo. El modelo enfatiza el estudio de los riesgos del proyecto, como por ejemplo las especificaciones incompletas. Se prevé, en este modelo, varios ciclos o "vueltas de espiral", cada uno de ellos con cuatro etapas: Definición de objetivos, Evaluación y reducción del riesgo, Desarrollo y validación y Planificación del siguiente ciclo. En este modelo, una actividad comienza solo cuando se entienden los objetivos y riesgos involucrados. El desarrollo se incrementa en cada etapa, generando una solución completa. La metodología en espiral ha sido utilizado con éxito en grandes sistemas, pero su complejidad la hace desaconsejable para el desarrollo de sistemas medianos o pequeños.

3.5. Modelo XP

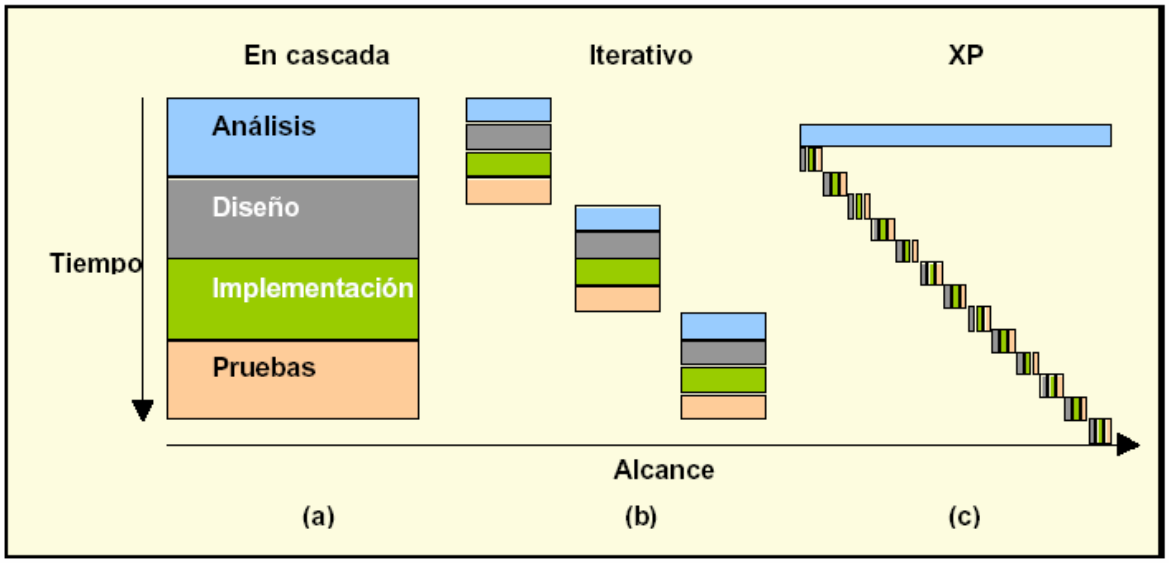
La metodología XP define cuatro variables para cualquier proyecto de software: ***costo, tiempo, calidad*** y ***alcance***. Además, se especifica que, de estas cuatro variables, sólo tres de ellas podrán ser fijadas arbitrariamente por actores externos al grupo de desarrolladores (clientes y jefes de 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. Este modelo es analizado por Kent Beck, en [9], donde propone las ventajas de un contrato con alcances opcionales.

Como se detalló en los apartados anteriores, los ciclos de vida "tradicionales" proponen una clara distinción entre las etapas del proyecto de software, y tienen un plan bien preestablecido acerca del proceso de desarrollo. Asimismo, en todos ellos se parte de especificaciones claras, si no del total del proyecto, por lo menos de una buena parte inicial.

El ciclo de vida de un proyecto XP incluye, al igual que las otras metodologías, entender lo que el cliente necesita, estimar el esfuerzo, crear la solución y entregar el producto final al cliente. Sin embargo, XP propone un ciclo de vida dinámico, donde se admite expresamente que, en muchos casos, los clientes no son capaces de especificar sus requerimientos al comienzo de un 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 (y que serán detalladas más adelante).

Típicamente un proyecto con XP lleva 10 a 15 ciclos o iteraciones. La siguiente figura [10] esquematiza los ciclos de desarrollo en cascada e iterativos tradicionales (por ejemplo, incremental o espiral), comparados con el de XP.



Si bien el ciclo de vida de un proyecto XP es muy dinámico, se puede separar en fases [11]. Varios de los detalles acerca de las tareas de éstas fases se detallan más adelante, en la sección “Reglas y Practicas”:

Fase de exploración
Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de sencillas “historias de usuarios”. Los programadores estiman los tiempos de desarrollo en base a esta información. Debe quedar claro que las estimaciones realizadas en esta fase son primarias (ya que estarán basadas en datos de muy alto nivel), y podrían variar cuando se analicen más en detalle en cada iteración.
Esta fase dura típicamente un par de semanas, y el resultado es una visión general del sistema, y un plazo total estimado.

Fase de planificación
La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas, o *"Release Plan"*, como se detallará en la sección "Reglas y Practicas".

Fase de iteraciones
Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta fase, generando al final de cada una un entregable funcional que implementa las historias de usuario asignadas a la iteración. Como las historias de usuario no tienen suficiente detalle como para permitir su análisis y desarrollo, al principio de cada iteración se realizan las tareas necesarias de análisis, recabando con el cliente todos los datos que sean necesarios. El cliente, por lo tanto, también debe participar activamente durante esta fase del ciclo.
Las iteraciones son también utilizadas para medir el progreso del proyecto. Una iteración terminada sin errores es una medida clara de avance.

Fase de puesta en producción
Si bien al final de cada iteración se entregan módulos funcionales y sin errores, puede ser deseable por parte del cliente no poner el sistema en producción hasta tanto no se tenga la funcionalidad completa.
En esta fase no se realizan más desarrollos funcionales, pero pueden ser necesarias tareas de ajuste ("fine tuning").

4. Reglas y Practicas

La metodología XP tiene un conjunto importante de reglas y prácticas. En forma genérica, se pueden agrupar en [12], [13]:

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

4.1. 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 [14]:

- 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:

Historias de usuarios

Las "Historias de usuarios" (*"User stories"*) sustituyen a los documentos de especificación funcional, y a los "casos de uso". Estas "historias" son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más importante entre estas historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuario deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Cuando llegue el momento de la implementación, los desarrolladores dialogarán directamente con el cliente para obtener todos los detalles necesarios.

Las historias de usuarios deben poder ser programadas en un tiempo entre una y tres semanas. Si la estimación es superior a tres semanas, debe ser dividida en dos o más historias. Si es menos de una semana, se debe combinar con otra historia.

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.

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 preestablecido.
Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada historia de usuario se traduce en tareas especificas de programación. Asimismo, 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.

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.

4.2. 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:

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.

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"[1]), para explorar diferentes soluciones. Estos

[1] Spike se traduce como "punta" o "clavo", o también como "frustrar"

programas son únicamente para probar o evaluar una solución, y suelen ser desechados luego de su evaluación.

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 reescrito. 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.

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. En un trabajo realizado en el School of Computer Science del Carnegie Mellon, se cuestiona la utilidad de su uso [15].

4.3. Desarrollo del código

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.

Si bien esto parece demandar del cliente recursos por un tiempo prolongado, debe tenerse en cuenta que en otras metodologías este tiempo es insumido por el cliente en realizar los documentos detallados de especificación.
Adicionalmente, al estar el cliente en todo el proceso, puede prevenir a tiempo de situaciones no deseables, o de funcionamientos que no eran los que en realidad se deseaban. En otras metodologías, estas situaciones son detectadas en forma muy tardía del ciclo de desarrollo, y su corrección puede llegar a ser muy complicada.

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.

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 refiere esta práctica, son las pruebas unitarias, realizados por los desarrolladores. La definición de estos test al comienzo, condiciona o "dirige" el desarrollo.
Un ejemplo de como realizar esto de manera práctica puede verse en [16].

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.
En un estudio realizado por Cockburn y Williams [17], se concluye que la programación en pares tiene un sobre costo aproximado de 15%, y no de un 100% como se puede pensar a priori. Este sobre costo es rápidamente pagado por la mejor calidad obtenida en el producto final.
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.

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.

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". Ward Cunningham explica en una entrevista con Bill Veners [18], que este razonamiento no es correcto cuando se trabaja con la metodología de XP. En este caso, quienes encuentran un problema, o necesitan desarrollar una nueva función, pueden resolverlo directamente, sin necesidad de "negociar" con el "dueño" o autor del módulo (ya que, de hecho, este concepto no existe en XP). Muchas veces, explica Cunningham, una solución pasa por la recodificación de varios módulos, que atraviesan de forma horizontal una determinada jerarquía vertical. Si es necesario dialogar y convencer al encargado de cada módulo, posiblemente la solución no se pueda implementar, por lo menos en tiempos razonables. En XP, se promueve la recodificación, en aras de generar códigos mas simples y adaptados a las realidades cambiantes. Cualquier pareja de programadores puede tomar la responsabilidad de este cambio. Los testeos permanentes deberían de aseguran que los cambios realizados cumplen con lo requerido, y además, no afectan al resto de las funcionalidades.

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.

4.4. Pruebas

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. Por otra parte, como se mencionó anteriormente, 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.

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.

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 éstas 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.

5. Valores en XP

XP se basa en cuatro valores, que deben estar presentes en el equipo de desarrollo para que el proyecto tenga éxito

Comunicación
Muchos de los problemas que existen en proyectos de software (así como en muchos otros ámbitos) se deben a problemas de comunicación entre las personas. La comunicación permanente es fundamental en XP. Dado que la documentación es escasa, el diálogo frontal, cara a cara, entre desarrolladores, gerentes y el cliente es el medio básico de comunicación. Una buena comunicación tiene que estar presente durante todo el proyecto.

Simplicidad
XP, como metodología ágil, apuesta a la sencillez, en su máxima expresión. Sencillez en el diseño, en el código, en los procesos, etc. La sencillez es esencial para que todos puedan entender el código, y se trata de mejorar mediante recodificaciones continuas.

Retroalimentación
La retroalimentación debe funcionar en forma permanente. El cliente debe brindar retroalimentación de las funciones desarrolladas, de manera de poder tomar sus comentarios para la próxima iteración, y para comprender, cada vez más, sus necesidades.
Los resultados de las pruebas unitarias son también una retroalimentación permanente que tienen los desarrolladores acerca de la calidad de su trabajo.

Coraje
Cuando se encuentran problemas serios en el diseño, o en cualquier otro aspecto, se debe tener el coraje suficiente como para encarar su solución, sin importar que tan difícil sea. Si es necesario cambiar completamente parte del código, hay que hacerlo, sin importar cuanto tiempo se ha invertido previamente en el mismo.

6. Aplicabilidad

Por lo general, cada metodología tiene sus escenarios de aplicabilidad. Ninguna de las metodologías de desarrollo de software son buenas para todos los proyectos. Para proyectos que requieran varias decenas de desarrolladores, y en los que las especificaciones estén claramente determinadas desde el comienzo, los métodos en cascada o espiral pueden ser los más adecuados. Por el contrario, para proyectos medianos, y en los que las especificaciones no se puedan obtener hasta luego de comenzado el proyecto, XP puede ser la metodología recomendada.

Asimismo, XP puede ser utilizado junto con otras prácticas. Un ejemplo de esto se puede encontrar en [19], donde XP se integra con métodos centrados en la arquitectura.

XP tampoco es incompatible con sistemas de calidad, como pueden ser ISO 9000 o CMMI. En varios artículos se detallan como pueden complementarse las

prácticas de XP con las prácticas de estos sistemas de gestión de calidad [20], [21] .

La metodología XP aplica a equipos relativamente pequeños. Si bien no hay un consenso en el número máximo de desarrolladores, todos parecen coincidir en números no mayores a 20. Sus propias prácticas así lo requieren, ya que, por ejemplo, mantener las *"Stand-up meeting"* con más de 20 personas parece poco razonable.
Por otro lado, el entorno físico en el que se realizan los desarrollos deben ser adecuados a la metodología. Escritorios amplios, con un ordenador y dos sillas, mesas redondas para trabajo en equipo, ambientes que permitan la permanente comunicación y colaboración, son algunos de los requerimientos de infraestructura para poder implementar esta metodología.

La participación e involucramiento del cliente en el proceso es fundamental. El cliente debe conocer y aprobar la metodología, y destinar los recursos necesarios. De otra manera, el proyecto no podrá ser llevado a cabo.

Finalmente, los participantes del equipo de desarrollo deben estar compenetrados con el proyecto y su metodología. Deben aceptar compartir sus códigos y ser responsables por el código que escribieron otros.

7. Críticas

Una de las críticas a XP es la dificultad de estimar cuánto va a costar un proyecto. Dado que el alcance del mismo no está completamente definido al comienzo, y que la metodología XP es expresamente abierta a los cambios durante todo el proceso, se torna sumamente difícil poder realizar un presupuesto previo. Para desarrollos "in house" este punto puede no ser crítico, pero sí lo es especialmente para empresas desarrolladoras de software, dónde deben presupuestar (y ganar) proyectos.

Por otro lado, muchas de las prácticas sugeridas por XP son compartibles y utilizadas, de una u otra forma, en muchas otras metodología. Por ejemplo, tener un diseño simple, utilizar estándares y mantener un ritmo sostenido de trabajo es claramente ventajoso, sea cual sea la metodología utilizada. Sin embargo, otras prácticas pueden ser más discutibles. Pensar únicamente en el diseño de lo que se debe entregar inmediatamente, sin tener en cuenta lo que deberá realizarse inmediatamente después, o dentro de poco tiempo, no siempre es la mejor decisión. Es cierto que muchas veces, por intentar generalizar o prever futuros casos, se invierte tiempo extra para entregables que quizás no lo requerirían. Sin embargo, si el análisis fue correctamente realizado, este tiempo extra inicial redundará en un menor tiempo total. Claro que, si pensamos que siempre existirán

cambios imprevistos, no conviene invertir tiempo en generalizar o prever futuras funciones. Es una práctica que puede admitir diversos puntos de vista.

Otra posible crítica a XP refiere a la baja documentación, pensando en el posterior mantenimiento del sistema. Si bien durante el proyecto el equipo tiene en mente todas sus particularidades, hay que prever que pasará luego de entregado, cuando el equipo se disuelva, y sea necesario realizar algún cambio o mejora. XP propone la recodificación permanente, para asegurar la claridad y simplicidad del código. Sin embargo, es posible que aún un código simple y claro no baste cuando otro equipo de trabajo tenga que tomar el sistema y cambiarlo. Es posible que sea necesario mantener cierta documentación, aunque es compartible que ésta debe ser la mínima necesaria.

En un artículo de Matt Stephens [22], se critica la "auto-realimentación" de XP, ya que sus prácticas requieren ser aplicadas en su totalidad para que el método sea viable: No disponer de requerimientos detallados escritos lleva a mantener un diseño simple. Un diseño simple lleva a que sea necesario recodificar constantemente. La recodificación necesita de permanentes pruebas unitarias. Las pruebas unitarias fallarán poco, gracias a la programación de a pares, los que requieren del cliente en sitio para poder conocer los detalles a implementar. Finalmente, tener al cliente en sitio hace irrelevante disponer de requerimientos escritos detallados, cerrando el círculo. Según este artículo, una falla en cualquiera de estas prácticas hará que todo el proyecto fracase.

8. Conclusiones

En una encuesta realizada sobre 45 proyectos realizados con XP en 2001 [23], se concluye que:

- Casi todos los proyectos se categorizaron como exitosos.
- El 100% de los desarrolladores encuestados afirmaron que volvería a utilizar la metodología XP en el siguiente proyecto si fuera apropiado.
- Las frecuentes ausencias del cliente fueron identificadas como el mayor riesgo en los proyectos.
- Los problemas más comunes fueron "barreras psicológicas", como por ejemplo el escepticismo de la línea gerencial, la filosofía de la empresa desarrolladora que no permitía tener al cliente en sitio, o que algunos desarrolladores se oponía al trabajo en parejas.
- Los elementos más útiles de XP fueron la propiedad colectiva del código, las pruebas y la integración continua. Las más críticas fueron la metáfora y el cliente en sitio.

Una encuesta más reciente, realizada en 2005 [24], presenta como resultados destacables que el 54.9% de los encuestados dicen tener amplia experiencia en XP, pero solo el 23.2% lo utiliza en la mayoría de los proyectos. Por otro lado, las prácticas marcadas como más útiles son el diseño simple, la programación dirigida

por los test y el uso de estándares, en ese orden. Cualquiera de estas prácticas podría aplicar perfectamente a otras metodologías.

XP es una de las nuevas metodologías ágiles. Siendo de reciente divulgación, está comenzando a ser utilizada por algunos, y criticada por otros. Es claro que no existe una metodología única para garantizar el éxito de cualquier proyecto de desarrollo de software, y esto aplica también a XP. Toda metodología requiere de cierta adaptación al proyecto, al cliente y a la idiosincrasia de la empresa desarrolladora.
Las metodologías tradicionales han intentado abarcar la mayor cantidad de situaciones, pero exigen un esfuerzo considerable en documentación y gerenciamiento de proyecto, que muchas veces genera una sobrecarga inaceptable en proyectos pequeños y medianos. XP propone una metodología ágil y concreta, aunque requiere de una nueva menara de pensar, ver y hacer las cosas, tanto por parte de los gerentes (responsables de la rentabilidad general del proyecto), como de los desarrolladores y también del cliente. La aplicabilidad de ésta metodología a cada proyecto debería ser analizada en cada caso, teniendo en cuenta el tamaño y tipo de proyecto, así como sus ventajas y desventajas.

Referencias

1 The new methodology
Martin Fowler
http://martinfowler.com/articles/newMethodology.html
13 de diciembre de 2005

2 What is Extreme Programming?
http://www.extremeprogramming.org/what.html

3 Kent Beck
http://www.threeriversinstitute.org/Kent%20Beck.htm

4 Chrysler Goes to “Extremes”
Distributed Computing, October 1998
Kent Beck et. al. (The C3 Team)

5 Manifesto for Agile Software Development
http://agilemanifesto.org/
Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

6 Ingeniería de Software orientada a objetos con UML Java e Internet
Alfredo Weitzenfeld
Thomson editores, 2005
ISBN 970-686-190-4

7 Managing the development of large software systems
Dr. Winston W Royce
Proceedings, IEEE Wescon, Pages 1-9
August 1970
www.cs.umd.edu/class/spring2003/ cmsc838p/Process/waterfall.pdf

8 A Spiral Model of Software Development and Enhancement
Barry W Boehm
Computer (IEEE)
Volume 21, Issue 5, May 1988 Pages 61 - 72

9 Optional Scope Contracts
Kent Beck, Dave Cleal, 1999

10 Extreme Programming (XP): un nuevo método de desarrollo de software
César F. Acebal, Juan M. Cueva Lovelle
Depto. de Informática, Área de Lenguajes y Sistemas Informáticos,
Universidad de Oviedo
NOVATICA, Marzo Abril 2002, pp 8-12

11 XP: A Project Manager's Primer
Steward Baird
March 22, 2002
Prentice Hall, Professional Technical Reference
http://www.phptr.com/articles/article.asp?p=26060&seqNum=6&rl=1

12 Extreme Programming: A gentle introduction
February 17, 2006
Don Wells
http://www.extremeprogramming.org/

13 The Rules and Practices of Extreme Programming
Don Wells
http://www.extremeprogramming.org/rules.html

14 Interview with Kent Beck and Martin Fowler
Addison-Wesley
Marzo 23, 2001
http://www.awprofessional.com/articles/article.asp?p=20972&redir=1&rl=1

15 How Useful Is the Metaphor Component of Agile Methods? A Preliminary Study
James Tomayko, James Herbsleb

June 2003
CMU-CS-03-152, School of Computer Science, Carnegie Mellon

16 Demystifying Extreme Programming: Test-driven programming
April 22, 2003
Roy Miller
http://www-128.ibm.com/developerworks/library/j-xp042203/index.html

17 The Costs and Benefits of Pair Programming
Humans and Technology Technical Report 2000.01, Jan '00
Alistair Cockburn, Laurie Williams
http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

18 Collective Ownership of Code and Text, A Conversation with Ward Cunningham
by Bill Venners
Artima Developer , December 1, 2003
http://www.artima.com/intv/ownership.html

19 Integrating Software-Architecture-Centric Methods into Extreme Programming
Robert L. Nord, James E. Tomayko, Rob Wojcik
September 2004
CMU/SEI-2004-TN-036, SEI

20 Extreme Programming from a CMM Perspective
Mark C. Paulk, Software Engineering Institute
IEEE SOFTWARE, November/December 2001

21 Combining Extreme Programming with ISO 9000
Jerzy R. Nawrocki, MichaÃl Jasi´nski, Bartosz Walter, and Adam Wojciechowski
Lecture Notes in Computer Science
Poznan University of Technology, Poland
www.cs.put.poznan.pl/jnawrocki/ publica/eurasia-ict02.pdf

22 The Case Against Extreme Programming
Matt Stephens
January 26, 2003
Software Reality
http://www.softwarereality.com/lifecycle/xp/case_against_xp.jsp

23 Quantitative Survey on Extreme Programming Projects
Bernhard Rumpe Astrid Schröder
Software & Systems Engineering, Munich University of Technology

24 Perceptions of Extreme Programming: A Pilot Study
Vojislav B. Misic

June 2005
Technical report TR 05/03
Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada R3T 2N2
(paper submitted to IEEE International Engineering Management conference IEMC2005)

.