novática
UPGRADE
Revista de la Asociación de Técnicos de Informática
Edición Digital
MARZO - ABRIL 2002
156

Novática, revista fundada en 1975, es el órgano oficial de expresión y formación continua de ATI (Asociación de Técnicos de Informática)

ATI es miembro de CEPIS *(Council of European Professional Informatics Societies)* y tiene un acuerdo de colaboración con ACM (Association for Computing Machinery). Tiene asimismo acuerdos de vinculación o colaboración con AdaSpain, AI^2 y ASTIC

<http://www.ati.es/novatica/>

Coordinación Editorial y Redacción Central (ATI Madrid)
Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica@ati.es*>
Composición, Edición y Redacción ATI Valencia
Palomino 14, 2ª, 46003 Valencia
Tlf./fax 963918531 <*secreval@ati.es*>
Administración, Subscripciones y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlf.934125235; fax 934127713 <*secregen@ati.es*>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel, Isla Cartuja 41092 Sevilla
Tlf./fax 954460779 <*secreand@ati.es*>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza
Tlf./fax 976235181 <*secreara@ati.es*>
Redacción ATI Asturias-Cantabria <*gp-astucant@ati.es*>
Redacción ATI Castilla-La Mancha <*gp-clmancha@ati.es*>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlf.986581413; fax 986580162 <*secregal@ati.es*>

Publicidad: Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <novatica.publicidad@ati.es>

Imprenta: 9 Impressió S.A., Juan de Austria 66, 08005 Barcelona.
Depósito Legal: B 15.154-1975
ISBN: 0211-2124; CODEN NOVAEC

Portada: Antonio Crespo Foix / © ATI 2002

SUMARIO

Monografía del próximo número: «*Recuperación de la información y la Web*»

César F. Acebal, Juan M. Cueva Lovelle
Depto. de Informática, Área de Lenguajes y Sistemas Informáticos, Universidad de Oviedo; Socios de ATI

<acebal@ieee.org>, <cueva@lsi.uniovi.es>

eXtreme Programming (XP): un nuevo método de desarrollo de software

Resumen: *¿Qué es eXtreme Programming (XP)? Eso es lo que pretende responder el presente artículo, descubriendo este nuevo método de desarrollo de software al lector desconocedor del mismo. Naturalmente, la extensión de cualquier artículo técnico no permite más que una somera introducción a cualquier nuevo método o técnica, pero trataremos de ser lo suficientemente didácticos como para que cada cual pueda formarse una idea aproximada de los principios básicos que subyacen en él, y se ofrecerán las referencias apropiadas para quien quiera profundizar en el mismo.*

Palabras clave: *eXtreme Programming, desarrollo de software, principios básicos*

1. Introducción

XP (*eXtreme Programming*), como ya se ha mencionado en el resumen, y como indica el subtítulo de este artículo, 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 un método «ligero», en contraposición a los métodos «pesados» como Métrica. Antes de continuar, aclararemos que en este artículo nos referiremos a él como «método», en contra de la tendencia oficial en Informática a denominar «metodologías» (ciencia de los métodos) a lo que no son más que métodos[1] o incluso meras notaciones gráficas.

Podríamos decir que XP nace «oficialmente» hace cinco años en un proyecto desarrollado por Kent Beck en *DaimlerChrysler*, 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. ¿Por qué suscita entonces tanta controversia, con opiniones enfrentadas entre la adoración y el desprecio? Como sugiere Kent Beck en su libro [Beck 2000], tal vez sea porque XP lleva un conjunto de técnicas y principios de sentido común a niveles extremos, entre las que podemos destacar:

- 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 unitarias**) sino que también los clientes comprobarán que el proyecto va satisfaciendo los requisitos (**pruebas funcionales**).
- Las pruebas de integración se efectuarán siempre, antes de añadir cualquier nueva clase al proyecto, o después de modificar cualquiera existente (**integración continua**), para lo que nos serviremos de *frameworks de testing*, como el *xUnit*.
- 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.

En definitiva, y para concluir con este apartado y entrar ya en materia, quedémonos con esta frase extraída del mismo libro de Beck:

> *«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.»*

2. Las cuatro variables

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

Autores

César Fernández Acebal es Ingeniero en Informática por la Universidad de Oviedo. Trabajó como profesor de Java y de programación web para estudiantes de formación profesional. Después, trabajó como director técnico de un compañía de desarrollo de sistios web. Ha combinado estos trabajos con una actividad continua como formador en Java, XML, desarrollo web, etc. Actualmente es arquitecto de sistemas en B2B 2000, una compañía de comercio electrónico. Sus intereses en cuanto a investigación incluyen programación orientada a objetos e ingeniería de software y procesos ágiles de desarrollo. Es miembro de ATI, IEEE, la Computer Society, ACM, etc.

Juan Manuel Cueva Lovelle es Ingeniero de Minas por la ETS de Ingenieros de Minas de Oviedo, 1983. Consiguió su grado de doctor en la Universidad Politécnica de Madrid en 1990. Desde 1985, es profesor del area de Lenguajes y Sistemas Informáticos de la Universidad de Oviedo. Es *voting member* de IEEE y ACM. Sus intereses en cuanto a investigación incluyen tecnología orientada a objetos, procesadores de lenguaje, interfaz hombre-máquina, bases de datos orientadas a objetos, ingeniería del software para web, etc.

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 a poco 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 calidad*. 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.

3. El coste del cambio

Aunque no es posible extendernos aquí sobre este aspecto, creemos 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, como indica la **figura 1**:

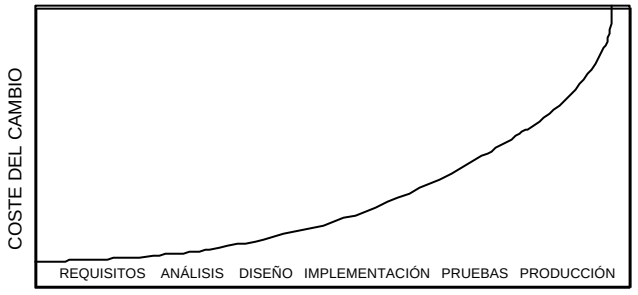

Figura 1. Coste del cambio en la ingeniería software «tradicional»

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, como se refleja en la **figura 2**.

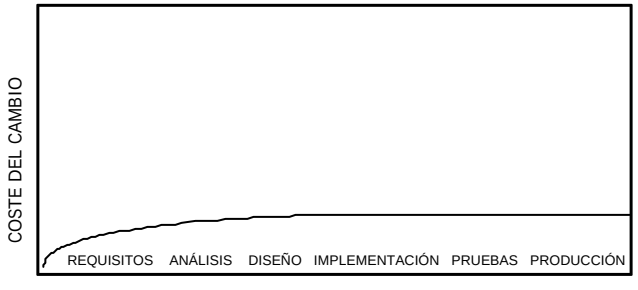

Figura 2. Coste del cambio en XP

Naturalmente, no todo el mundo está de acuerdo con está asunción (y en el sitio web de Ron Jeffries [Jeffries-www]

pueden leerse varias opiniones al respecto). Pero valga, en cualquier caso, la actitud de que, 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, junto con nuestros conocimientos de factorización [Fowler 1999] y, sobre todo, los *tests* y la integración continua, hacen posible que los cambios puedan ser llevados a cabo tan a menudo como sea necesario.

4. Las prácticas

Pero vayamos un poco más al grano. ¿En qué consiste realmente XP? ¿Cuáles son esas prácticas a las que ya se ha hecho mención y que hacen posible ese cambio de mentalidad a la hora de desarrollar software? Confiemos en que el lector sepa disculparnos por la necesaria brevedad en la exposición y pasemos a describirlas, sucintamente, una a una.

4.1. La 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.

4.2. Versiones pequeñas

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. El cliente y el equipo de desarrollo se beneficiarán de la retroalimentación que produce un sistema funcionando, y esto se reflejará en sucesivas versiones.

4.3. Diseño simple

En vez de perseguir a toda costa un diseño en el que hemos de desarrollar dotes adivinatorias para tratar de anticiparnos al futuro, lo que XP preconiza es diseñar en cada momento para las necesidades presentes.

4.4. *Testing*

«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 ... Pero aquí no hay discusión posible: si no hacemos *tests*, 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. Esto ayuda al principio de *programación por intención*, esto es, a escribir el código como si los métodos más costosos ya hubieran sido escritos y sólo tuviéramos así que enviar el correspondiente mensaje, de manera que el código refleje bien su intención y se documente a sí mismo. En el sitio web de JUnit mencionado en el párrafo anterior se pueden encontrar interesantes artículos que explican cómo deben escribirse estos *tests*.

4.5. *Refactoring*

Responde al principio de simplicidad. Y, muy escuetamente, consiste en dejar el código existente en el estado más simple posible, de manera que no pierda –ni gane– funcionalidad y que se sigan ejecutando correctamente todos los *tests*. Esto nos hará sentirnos más cómodos con el código ya escrito y, por tanto, menos reacios a modificarlo cuando haya que añadir o cambiar alguna característica. En el caso de sistemas heredados, o de proyectos que se tomen ya empezados, seguramente habrá que dedicar varias semanas sólo a factorizar el código –lo cual suele ser fuente de tensiones con el correspondiente jefe de proyecto, cuando se le diga que se va a parar éste varios días «sólo» para modificar el código existente, que funciona, sin añadirle ninguna nueva funcionalidad.

4.6. *Pair programming*

Todo el código será desarrollado en parejas –dos personas compartiendo un solo monitor y teclado. Quien codifica estará pensando en el mejor modo de implementar un determinado método, mientras que su compañero lo hará de una manera más estratégica:

- ¿Estamos siguiendo el enfoque apropiado?
- ¿Qué es lo que podría fallar aquí? ¿Qué deberíamos comprobar en los *tests*?
- ¿Hay alguna manera de simplificar el sistema?

Por supuesto, los roles son intercambiables, de manera que en cualquier momento quien observaba puede tomar el teclado para ejemplificar alguna idea o, simplemente, para turnar a su compañero. Igualmente, la composición de las parejas cambiará siempre que uno de los dos sea requerido por algún otro miembro del equipo para que le ayude con su código.

4.7. Propiedad colectiva del código

Cualquiera puede modificar cualquier porción del código, en cualquier momento. En efecto, cualquiera que vea una posibilidad de simplificar, mediante *refactoring*, cualquier clase o cualquier método, hayan sido o no escritos por él, deberá hacerlo sin más dilación. El uso de estándares de

codificación y la confianza que nos dan los *tests* de que todo va a seguir funcionando bien tras una modificación, hacen que esto no sea ningún problema en XP.

4.8. Integración continua

Cada pocas horas –o al cabo de un día de programación, como mucho– se integra el sistema completo Para ello existirá un máquina así llamada, de integración, a la que se acercará una pareja de programadores cada vez que tengan una clase que haya sido probada unitariamente. Si al añadir la nueva clase junto con sus *tests* unitarios, el sistema completo sigue funcionando correctamente –pasa todos los *tests*–, los programadores darán por finalizada esa tarea. Si no, serán los responsables de dejar el sistema de nuevo con los tests funcionando al 100%. Si después de un cierto tiempo no son capaces de descubrir qué es lo que falla, tirarán el código a la basura y volverán a comenzar de nuevo.

4.9. 40 horas semanales

Si realmente queremos ofrecer calidad, y no meramente un sistema que funcione –lo cual, en informática, ya sabemos que es trivial[2]– querremos que cada miembro de nuestro equipo se levante cada mañana descansado y se vaya a las 6 de la tarde a su casa cansado y con la satisfacción del trabajo bien hecho, y que al llegar el viernes sepa que cuenta con dos días para descansar y dedicarse a cosas que nada tengan que ver con el trabajo. Naturalmente, las 40 horas no es una regla fija –pueden ir de 35 a 45–, pero lo que es seguro es que nadie es capaz de trabajar 60 horas a la semana y hacerlo con calidad.

4.10. Cliente en el sitio

Otra controvertida regla de XP: al menos un cliente real debe estar permanentemente junto al equipo de desarrollo, para responder cualquier consulta que los programadores le planteen, para establecer prioridades ... Si el cliente arguye que su tiempo es demasiado valioso, deberemos entender que realmente el proyecto que nos ha encomendado es lo suficientemente trivial como para que no merezca su atención, y que no importa que esté construido a partir de suposiciones hechas por programadores que nada, o muy poco, saben del negocio real.

4.11. Estándares de codificación

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.

5. Planificación

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. Esto a pesar de las críticas que muchas personas le hacen, acaso tras una lectura apresurada de algún artículo como éste. Pero para cualquiera que haya leído los libros que explican el proceso, quedará claro cómo una parte fundamental de XP es precisamente la planificación. Lo que ocurre es que, aceptando que el desarrollo de software, como acaso cualquier cosa en esta vida, es un proceso caótico, no trata de buscar un determinismo inexistente y sí de poner los medios necesarios para manejar esa complejidad, aceptándola, sin tratar de doblegarla bajo los corsés de los métodos pesados o burocráticos –se recomienda vivamente la lectura de la amena introducción de Antonio Escohotado a la teoría del caos [Escohotado 1999, que creemos que tiene mucho que ver con esta idea subyacente de XP. En definitiva, los métodos ligeros –y XP es uno de ellos– son *adaptativos* más que *predictivos* [Fowler-www].

5.1. El ciclo de vida

Si, como se ha demostrado, los largos ciclos de desarrollo de los métodos tradicionales son incapaces de adaptarse al

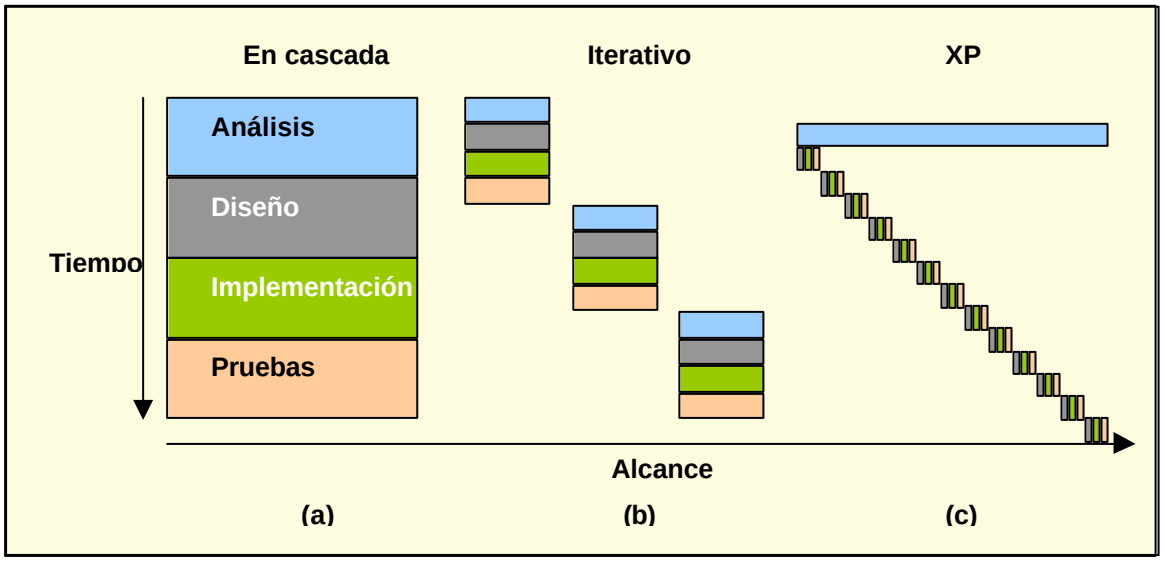

Figura 3. Evolución de los largos ciclos de desarrollo en cascada (a) a ciclos iterativos más cortos (b) y a la mezcla que hace XP (c)

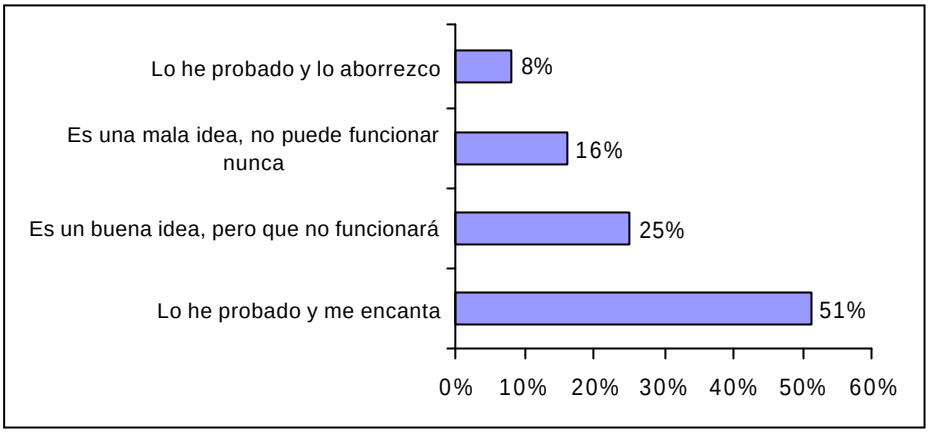

Figura 4. Encuesta de IBM (octubre de 2000): ¿Qué opina de eXtreme Programming*?*

cambio, tal vez lo que haya que hacer sea ciclos de desarrollo más cortos. Esta es una de las ideas centrales de XP. En la **figura 3** vemos una comparativa gráfica entre el modelo en cascada, el modelo en espiral y XP.

6. Conclusiones

Como decíamos al principio de esta ponencia, XP, con sólo un año desde la publicación del primer libro, 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], que aparece en la **figura 4**.

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 dominio de los *idioms* 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»... y así tantas otras encendidas críticas. 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[3].

Nosotros simplemente deseamos hacer un llamamiento para que XP no se malinterprete por la inevitable superficialidad de artículos como el presente. Quede claro también que el artífice de este método no es ningún recién llegado, sino de uno de los pioneros en el empleo de patrones software, creador de las fichas CRC, autor del *framework* para editores de dibujo *HotDraw*, del *framework* de *testing xUnit* ... Aunque sólo fuera por ello, merecería la pena echar al menos un vistazo a este nuevo y excitante método de desarrollo software.

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.

En cualquier caso, el primer libro de Beck es un soplo de aire fresco que debiera ser de obligada lectura para cualquier *ingeniero de software* –o *arquitecto de software*, si atendemos a la denominación preferida por nuestro amigo Ricardo Devis–, sea cual sea la opinión que finalmente se forme sobre XP. Por si fuera poco, asegura un rato enormemente divertido.

7. Notas

[1] Ricardo Devis Botella. *C++. STL, Plantillas, Excepciones, Roles y Objetos*. Paraninfo, 1997. ISBN 84-283-2362-3.
[2] Ricardo Devis Botella. *Curso de Experto Universitario en Integración de Aplicaciones Internet mediante Java y XML*. Universidad de Oviedo, 2000.
[3] Raúl Izquierdo Castanedo. *Comunicación privada.*

8. Referencias

[Beck 2000] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison Wesley Longman, 2000. ISBN 201-61641-6

[Jeffries et al 2000] Ronald E. Jeffries et al. *Extreme Programming Installed.* Addison-Wesley, 2000. ISBN 0201708426

[Beck and Fowler] Kent Beck, Martin Fowler. *Planning Extreme Programming*. Addison-Wesley. ISBN 0201710919

[Jeffries-www] www.xprogramming.com. Uno de los más completos portales sobre XP, por Ron Jeffries.

[Fowler 1999] Martin Fowler. *Refactoring : Improving the Design of Existing Code.* Addison-Wesley, 1999. ISBN 0201485672

[Beck 1999] Kent Beck. *Embracing Change with Extreme Programming*. Computer (revista de la *IEEE Computer Society*).Vol. 32, No. 10. Octubre 1999, pp. 70-77

[Dyna-www] 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.

[Fowler-www] Martín Fowler. *The New Methodology.* <http://www.martinfowler.com/articles/newMethodology.html>

[Junit-www] www.junit.org. JUnit, un framework de testing automático para Java, adaptado de su homónimo para Smalltalk, y existente para muchos otros lenguajes. Estos otros, bajo el nombre genérico de xUnit, se encuentran disponibles en el sitio web de Ron Jeffries [Jeffries-www], en el apartado de software.

[IBM-encuesta] 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.

[Escohotado 1999] Antonio Escohotado. *Caos y orden.* Espasa Calpe, 1999. ISBN 84-239-9751-0. Una interesante introducción a la teoría del caos, la cual describe, a nuestro juicio, la actitud necesaria para enfrentarse al desarrollo de software, desde la perspectiva de XP.