Programación Extrema: Prácticas, Aceptación y Controversia

M.C. Saúl González Campos[1] y M.C. Luis Felipe Fernández Martínez[2]

[1] saugonza@uacj.mx; [2] lfernand@uacj.mx

Universidad Autónoma de Cd. Juárez

Palabras clave: Programación Extrema, Procesos de Software, Metodologías Ágiles, Ingeniería de Software.

Resumen

La Programación Extrema es un paradigma de desarrollo de software que queda encuadrado en el grupo de metodologías ágiles. A seis años de su concepción se ha mostrado como una alternativa efectiva si se utiliza en un contexto adecuado, aunque igualmente ha sido objeto de críticas por mantener una serie de premisas que en cierta manera restringen el actuar del programador. El presente trabajo da una introducción general a esta metodología y destaca sus principales áreas de aplicación, incluyendo su uso en la enseñanza de la programación. También se resaltan los argumentos usualmente presentados tanto a favor como en contra.

1. Introducción

Las metodologías ágiles de desarrollo de software han despertado interés en los últimos años debido a que proponen simplicidad y velocidad para crear sistemas. Los programadores se concentran solamente en aquellas funciones que se necesitan inmediatamente, entregándolas al cliente lo antes posible, obteniendo retroalimentación constante y reaccionando rápidamente a los cambios en el negocio y la tecnología. Se han hecho esfuerzos por analizar y clasificar éstas metodologías [Pekka 2003], las cuales han aparecido en buen número y aparentemente no cesarán de hacerlo en un futuro cercano. Entre los ejemplos más conocidos de estas metodologías se encuentran los siguientes: "Adaptive Software Development", "Agile Modeling", "Crystal", "Dynamic Systems Development Method", "Extreme Programming", "Feature-Driven Development", "Internet-Speed Development", "Pragmatic Programming", y "Scrum".

[Aiken 2004] clasifica las metodologías ágiles de acuerdo a su evolución, según se muestra en el mapa de la figura 1.

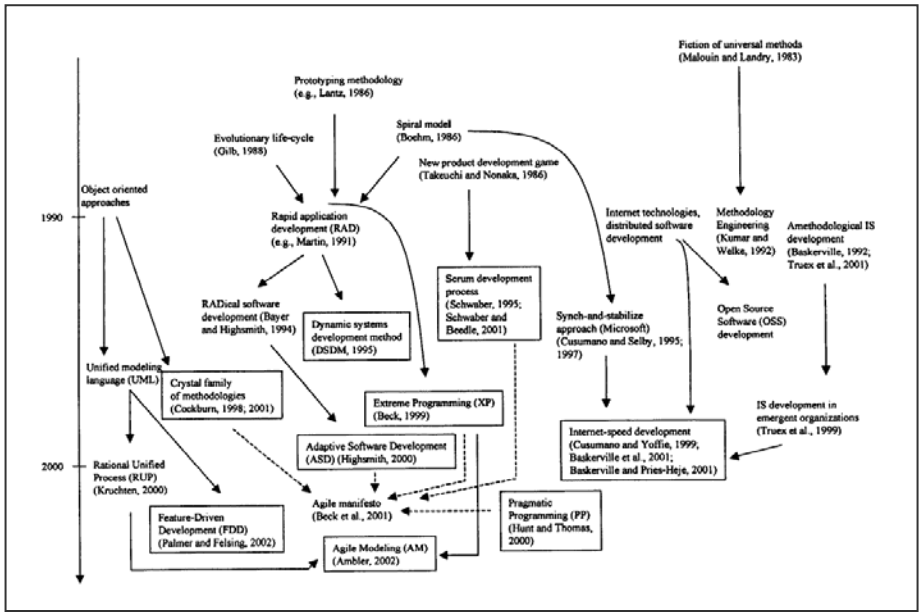


Fig. 1 Mapa evolucionario de las metodologías ágiles. Fuente: Pekka 2003

2. Antecedentes de XP

Los antecedentes de la Programación Extrema (que denominaremos simplemente como XP en el resto de este documento) se pueden encontrar en los trabajos de Ward Cunningham para proponer un desarrollo de software en el que predominara la simplicidad y la eficiencia. En 1989, Cunningham formó un equipo que usaba los principios y muchas de las prácticas que después adoptaría XP, mientras trabajaba para la compañía "Wyatt Software" [Fowler 2000]. Sin embargo, se reconoce a Kent Beck como el que articuló esta propuesta y le dio nombre propio. Beck, por su parte, reconoce a Cunningham como la persona en cuyas prácticas se inspiró para formalizar este nuevo paradigma, el cual tiene sus orígenes durante el proyecto C3 (Chrysler Comprehensive Compensation) en 1996, el cual consistió en un desarrollo a largo plazo para reescribir el sistema de nómina de Daimler-Chrysler [English 2002], y donde Beck aplicó su filosofía con éxito. Posteriormente, la consolidación de XP se logra con la publicación del libro "*Extreme Programming Explained: embrace change*" en el año 1999, donde Beck resume su propia experiencia y le da forma y nombre a la entonces nueva metodología de desarrollo de software, la cual le valió el premio: "Software Development Jolt Product Excellence". A partir de aquel año, ha crecido alrededor del mundo tanto el número de entusiastas adeptos como el de escépticos y críticos, manteniéndose aun abierto un debate acerca de su utilidad y alcance reales.

3. Características generales de XP

Una de las características distintivas de XP (que comparte con otras metodologías ágiles) es que de alguna manera representa la antítesis de lo que es el tradicional proceso de desarrollar software. XP es deliberadamente una metodología "liviana" que pasa por alto la utilización de elaborados casos de uso, la exhaustiva definición de requerimientos y la producción de una extensa documentación [English 2002]. Todo lo anterior puede parecer caótico según el enfoque tradicional de la ingeniería de software, aunque no hay que olvidar que XP tiene asociado un ciclo de vida y es considerado a su vez un proceso [Amber 2002]. La tendencia de entregar software en lapsos cada vez menores de tiempo y con exigencias de costos reducidos y altos estándares de calidad, hace que XP sea una opción a considerar. En términos generales XP parece ser una metodología adecuada para proyectos medianos y pequeños, donde los equipos de desarrollo no pasan de 10 programadores y donde la constante es que los requerimientos cambien, a veces radicalmente, durante la etapa de desarrollo.

Otra característica que distingue a XP en la práctica es que, típicamente, los que se encargan de introducirla en los ambientes de trabajo son los propios encargados del desarrollo y sus equipos de programación, a diferencia de lo que sucede con otras metodologías, las cuales normalmente se introducen a un nivel corporativo y gradualmente se van bajando hasta alcanzar a los equipos de desarrollo [Strigel 2001].

4. Justificación y fundamentos de XP

Existe un problema básico que puede justificar el empleo de una metodología ágil de desarrollo de software. Este problema es el "riesgo", visto como un factor que se hace presente de diferentes maneras en cualquier proyecto de programación, por ejemplo, el riesgo de retardos en el tiempo de entrega al cliente, de cancelación del proyecto antes de llevarlo a producción, de corta vida del proyecto por altos costos de mantenimiento, de tasa de defectos fuera de control, de inadecuación del sistema contra las expectativas del cliente, de cambios en los requerimientos originales, de exceso de funcionalidades que el cliente no requiere ni usa, de desintegración del equipo de desarrollo, etc.

En este contexto, XP se presenta como una alternativa para mantener ese riesgo en un nivel bajo durante todas las etapas del proceso de desarrollo, según se justificará mas adelante. La filosofía de XP asume que, bajo ciertas circunstancias, se puede modificar la curva de comportamiento exponencial del "costo de cambio" contra el "tiempo" [Beck 2000]. Esto es, que en las metodologías tradicionales de desarrollo (modelo de cascada por ejemplo) se piensa que introducir cambios en el proyecto tal vez tiene un impacto mínimo si se hace en la fase de análisis de requerimientos pero tendrá un gran impacto si se hace cuando el sistema ya está en producción (ver fig. 2).

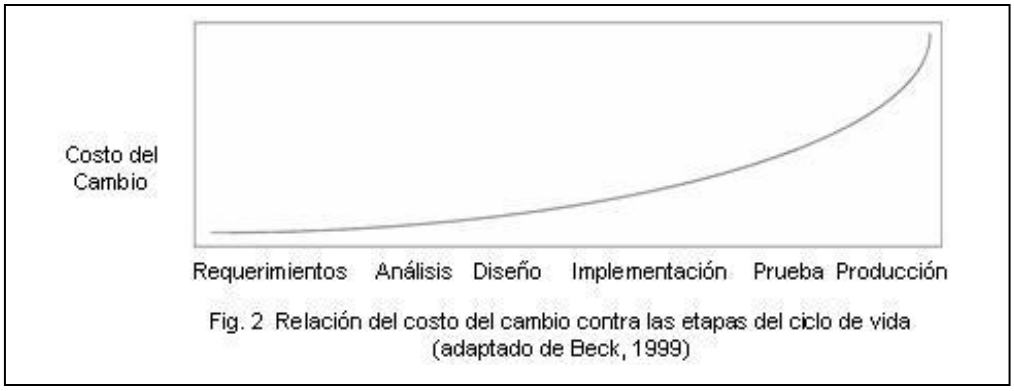


Fig. 2 Relación del costo del cambio contra las etapas del ciclo de vida (adaptado de Beck, 1999)

En XP, en cambio, se asume que mediante la aplicación de técnicas para realizar diseños simples, la automatización de las pruebas y un refinamiento del diseño en forma constante, se puede modificar la curva del costo para que tenga un comportamiento semejante al mostrado en la fig. 3

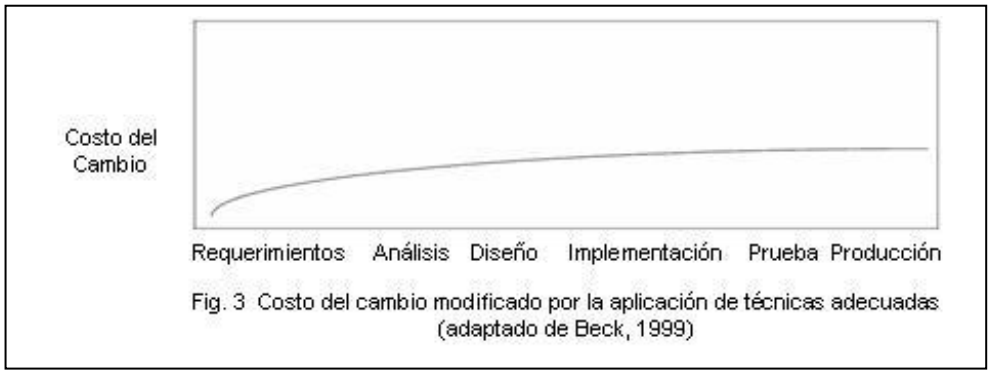


Fig. 3 Costo del cambio modificado por la aplicación de técnicas adecuadas (adaptado de Beck, 1999)

Esta visión es crítica para entender la propuesta de XP, de manera que si por cualquier razón no es posible obtener este comportamiento en el costo, adoptar la metodología XP en esas circunstancias representaría muy probablemente un camino al fracaso.

XP se fundamenta en 4 valores esenciales: comunicación, simplicidad, retroalimentación y coraje. En cuanto al primer valor, XP está definida de tal manera que diseña prácticas de programación que no pueden ser llevadas a cabo sin un nivel adecuado de comunicación (entre programadores, administradores y cliente). Respecto a la simplicidad, XP hace que los desarrolladores intenten producir el código más simple que realice la funcionalidad requerida. Ninguna función adicional debería ser incluida si no se requiere en el momento presente, esto permite mantener el diseño más simple posible en todo momento. Por otra parte, la retroalimentación se puede aplicar a diferentes escalas de tiempo, por ejemplo de minutos a días, cuando se requiere tener una información constante del estado del sistema, tanto de las pruebas del programador conforme implementa unidades de código, como del equipo de programación hacia el cliente. A una escala mayor de retroalimentación, de semanas o meses, se fomentan las revisiones del cliente hacia el avance general del proyecto, comparando las entregas parciales contra el plan general. Finalmente, respecto al coraje, este valor se refiere a la actitud del equipo hacia la programación a máxima velocidad, a estar dispuestos incluso a desechar código que no funciona, sin importar la cantidad de líneas, o a codificar diferentes alternativas para compararlas y seleccionar la mejor. También destaca el ímpetu que debe tener el programador para someter a continuas pruebas el desarrollo y a interactuar dinámica y

positivamente con todo el resto del equipo. De acuerdo a la filosofía de XP, estos 4 valores se soportan mutuamente, de manera que mantener unos y relegar otros no conducirán al resultado deseado.

5. Principios básicos de XP

Derivándose de los cuatro valores antes mencionados, XP propone una serie de principios que dirigen la metodología y los cuales se describen brevemente a continuación.

Es necesario que exista una retroalimentación rápida en todas las etapas de desarrollo y entre todos los miembros del equipo. Se debe también asumir simplicidad al momento de buscar soluciones (asumir que siempre existe una solución simple y que es necesario encontrarla). Los cambios tanto en diseño, planeación, codificación, e incluso en la adopción de XP, deben ser cambios incrementales. También se enfatiza la conveniencia de que todo el equipo de desarrollo esté convencido de abrazar el cambio, esto es, convencido de que utilizar XP dará el resultado deseado. Finalmente, XP asume que se debe realizar siempre un trabajo de calidad donde solo se permite la excelencia. Una discusión detallada de estos principios y otros denominados como complementarios puede encontrarse en la obra de Beck [Beck 2000].

6. Las 12 prácticas de XP

XP especifica ciertas prácticas concretas de programación que deben llevarse a cabo al implementar este modelo. Estas prácticas deben ser coherentes con los valores fundamentales y los principios básicos mencionados anteriormente. Estas prácticas de programación son las siguientes [Beck 2000]:

1. La planeación, en la cual la opinión del cliente y del equipo de desarrollo deben fusionarse como un todo coherente.
2. Entregas en iteraciones pequeñas, que permitan al cliente utilizar el sistema con las funcionalidades mínimas lo antes posible, e irlo complementando gradual y continuamente.
3. Manejo de metáforas, donde la metáfora ayuda a que todo el equipo de desarrollo entienda los elementos básicos del sistema y las relaciones entre ellos sin la necesidad de una arquitectura muy elaborada y detallada.
4. Diseño simple, donde siempre se intenta tener el código más simple, menos redundante y con las funcionalidades estrictamente necesarias en el presente.
5. Pruebas continuas, donde es imperativo que los programadores escriban pruebas por cada unidad de código y que el cliente participe en el diseño de pruebas funcionales.
6. Refabricación, donde este concepto se refiere a mantener una depuración y simplificación constante del sistema. Una vez que se ha añadido alguna funcionalidad, es necesario revisar y ser críticos para encontrar puntos de simplificación del código.
7. Programación en pares, donde toda la codificación debe hacerse en parejas de programadores, cada pareja compartiendo un mismo monitor y teclado. El que utiliza el teclado piensa en la mejor manera de implementar alguna funcionalidad, el otro piensa estratégicamente, cuestionando si se puede simplificar, anticipando pruebas, o preguntándose si el enfoque es el adecuado o si debe descartarse código y replantear el problema. Se alienta la rotación continua de programadores en los pares, combinando diferentes niveles de experiencia y de conocimiento del código.
8. Propiedad colectiva del código. Aquí se dice que el código puede ser modificado por cualquier elemento del equipo de programación, esto es, que no hay propiedad individual de algún programador sobre alguna sección de código. Se asume que al seguir las prácticas de XP, después de un tiempo razonable, cualquier programador conoce todo el código, principalmente por el hecho de la programación en pares.
9. Integración continua (mantener una sola revisión para todo el equipo de programación), tan pronto como pequeños "saltos" en la versión actual sean concluidos. Se recomiendan lapsos entre integraciones de pocas horas a no más de un día.
10. Semana de 40 horas. XP afirma que las condiciones de trabajo óptimas para programar "a máxima velocidad" es solo en un turno normal (8 horas). En este sentido las horas extras o fines de semana trabajados solo desgastan al equipo de programación, afectan el rendimiento y generan un ambiente propicio para cometer errores, ser displicentes en el apego a las normas y producir software de mediocre calidad.
11. El cliente debe estar disponible localmente, donde un representante del cliente debe permanecer por turnos completos en el sitio de programación para contestar cualquier pregunta y ayudar en el desarrollo de pruebas funcionales.

12. Mantener estándares de codificación entre los programadores. Esto es esencial para la programación en pares y para la propiedad colectiva del código.

El enfoque de XP respecto a estas doce prácticas es que se debe aprovechar la sinergia de todas ellas cuando se adoptan como un todo. Esto es, que si se elije implementar solamente algunas, todo el enfoque metodológico corre el riesgo de fracasar.

7. Aspectos positivos de XP

A continuación hemos querido incluir aquellos puntos que en general se reconocen como aspectos positivos de la propuesta de XP. Primeramente, las pruebas unitarias en el código es una práctica generalmente alentada y reconocida como un factor clave para obtener un software de alta calidad, si a eso se le agrega la exigencia de XP de que se hagan constantemente, durante cada etapa de codificación, quizás en el peor de los casos pudiera parecer un exceso. De manera semejante, la integración continua es aceptada y recomendada para evitar catástrofes ocasionadas por defectos no detectados a tiempo [Glass 2001]. El énfasis en la simplicidad y la refabricación es encontrado como un factor saludable en la práctica de programación, ya que normalmente se asocia el exceso de código con una lógica deficiente, un diseño innecesariamente complejo, problemas para el mantenimiento del sistema y un nicho para encontrar defectos que demeritan la calidad del producto. También se puede ver como positivo el hecho de que, filosóficamente, XP tiene un enfoque "extremadamente humano", siendo este un aspecto que el resto del campo del software debería tratar de emular. Desde el lado del programador, como ejemplos de este enfoque humano, tenemos la premisa de la semana de 40 horas, el alto valor que se le da a la comunicación y el rol protagónico que toma el programador en la etapa de planeación. Por el lado del cliente también se percibe el enfoque humano, ya que tenemos su presencia constante en las instalaciones del desarrollador, el dialogo que se fomenta entre el cliente y el resto del equipo de programación, la declaración de que "escuchar al cliente" es una de las cuatro actividades esenciales de la programación y el hecho de que se le otorga el mayor peso en la planeación.

8. Aspectos controversiales de XP

Desde los inicios de la Programación Extrema se ha afirmado que no es la metodología que va a resolver todos los problemas en Ingeniería de Software y se han resaltado sus limitaciones, señalando aquellos ambientes o proyectos en los que no se debería intentar aplicarla. Por ejemplo donde la cultura del cliente esté demasiado orientada a metodologías tradicionales (no ágiles) o a largas jornadas de trabajo. Tampoco en donde se involucren equipos de más de 20 programadores. No es aconsejable XP si no es posible disminuir la curva costo/tiempo según se analizó en la sección 4. Tampoco si la tecnología o el entorno no permiten realizar integraciones frecuentes o realizar pruebas continuamente. Finalmente, no se recomienda intentar XP si la distribución física del mobiliario impide la programación en pares o si no todos los programadores se encuentran en el mismo sitio.

Aún así, cuando se aplica a proyectos considerados como factibles para XP, el conjunto de prácticas tiene algunos aspectos que han sido considerados controversiales, por ejemplo, hay críticas hacia que XP desalienta el diseño (sobre todo una arquitectura inicial completa), que es débil en la documentación, que el modelo no aplica para proyectos donde la seguridad es crítica, que el exceso de pruebas retrasa el desarrollo, que el diseño simple solo aplica a proyectos simples, que la programación en pares consume mayor tiempo y recursos, y que la propiedad colectiva del software es causa de problemas [Aiken 2004]. También se menciona que XP asume implícitamente que siempre se utiliza el enfoque de programación orientada a objetos, lo cual no es necesariamente válido y que en general faltan datos estadísticos sólidos que comprueben que la práctica de XP mejora el desempeño ofrecido por otros modelos, ya que, según ciertos críticos, la evidencia actual recae mas en anécdotas que en abundancia de datos concretos.

Algunas prácticas, como la refabricación y la planeación, que se reconocen ampliamente como un valor positivo, son criticadas en XP por sobre-utilizarse. La refabricación, por ejemplo, se ve como sinónimo de rediseño constante y que se puede tomar como una excusa para relegar hasta el último minuto el diseño e irlo construyendo o modificando junto con el código. También la planeación, según algunos críticos, no debería hacerse "sobre la marcha" como parece recomendar XP, dada la experiencia de desastres en proyectos [Glass 2001]. Finalmente, y por ser la programación en pares quizás la práctica más polémica, merece un análisis más detallado. Se argumenta por ejemplo que no cualquier "clase" de programador desea trabajar de esta manera, dado que muchos prefieren trabajar solos cuando

están operando en forma "altamente creativa", y únicamente integrarse en equipo cuando hay piezas de información que compartir o decisiones que hacer. En XP, se exponen algunas barreras para poder implementar ésta práctica, como son: falta de comunicación, evaluaciones por desempeño (individual) en las organizaciones, idiosincrasia del programador típico (antisocial ó retraído), inadaptación al entorno físico sugerido (áreas abiertas en lugar de cubículos). Sin embargo, también se pueden encontrar beneficios en esta práctica, tales como: producir menos defectos (calidad), aumentar la productividad (menos tiempo se invierte en corregir y buscar errores), elevar la moral del equipo (el ambiente de trabajo es mejor), mejorar la confianza y el trabajo en equipo (los integrantes se conocen mucho mejor), naturalidad en la transferencia del conocimiento (todos conocen todo el código) y favorecer el aprendizaje (se aprenden estrategias y se profundiza mas en el uso de lenguajes y herramientas de desarrollo). Se ha recomendado como estrategia general para adoptar la programación en pares [Aiken 2004], que se haga lentamente en un inicio para detectar si es sostenible y compatible con la cultura de la organización y en ningún momento pretender hacerlo obligatorio sin consultarlo previamente con los elementos involucrados (aunque esto contradice en cierta forma lo indicado por la misma XP).

9. Extrapolación de las prácticas de XP

Si bien, como se mencionó anteriormente, XP se considera adecuada para proyectos de software pequeños o cuando mucho medianos, existen algunos intentos de extrapolar estas prácticas a proyectos grandes, haciendo algunas modificaciones que han funcionado en la práctica. En [Lan 2004] se describe un intento de esta naturaleza y se recomiendan las siguientes prácticas (que algunas se adoptan directamente de XP y otras se modifican):

- Diseño al inicio: Aquí se recomienda un buen diseño inicial (*up-front*) que respalde al proyecto y se combina con entregas continuas, programación en pares y refabricación constante.
- Se producen funcionalidades completas en cada iteración (entrega) durante el ciclo del software. El tiempo entre cada entrega es corto. Aquí prácticamente no se hacen modificaciones respecto a lo que se prescribe en XP.
- Se simula al cliente en las instalaciones, en lugar de ser un cliente real como dice XP, este rol lo asume alguien con experiencia en el área de aplicación del proyecto y a un nivel gerencial preferentemente.
- Programación en pares flexible. Se modifica la práctica de XP y en lugar de ser obligatoria para todo el código que se escribe, aquí se aplica solamente para las fases de análisis, diseño y prueba. La codificación, en cambio, se hace en forma individual.
- Selección y administración del equipo de desarrollo. Se buscan diferentes habilidades y experiencias en los programadores para robustecer al equipo, y se favorece la colaboración y el trabajo en grupo.
- Reutilización de software con refabricación constante. Aquí, a diferencia de XP, se fomenta la reutilización de componentes o de estructuras de programación, de manera que se refabrica el código de estas piezas de software para obtener elementos de calidad que puedan ser utilizados en el futuro.
- Organigramas horizontales con delegación cuidadosa de poder de decisión. Se fomenta que existan pocos niveles o capas organizacionales, tratando de limitar o eliminar las capas gerenciales intermedias. Al mismo tiempo se delega poder de decisión a los programadores y solo se mantienen centralizadas algunas decisiones críticas.

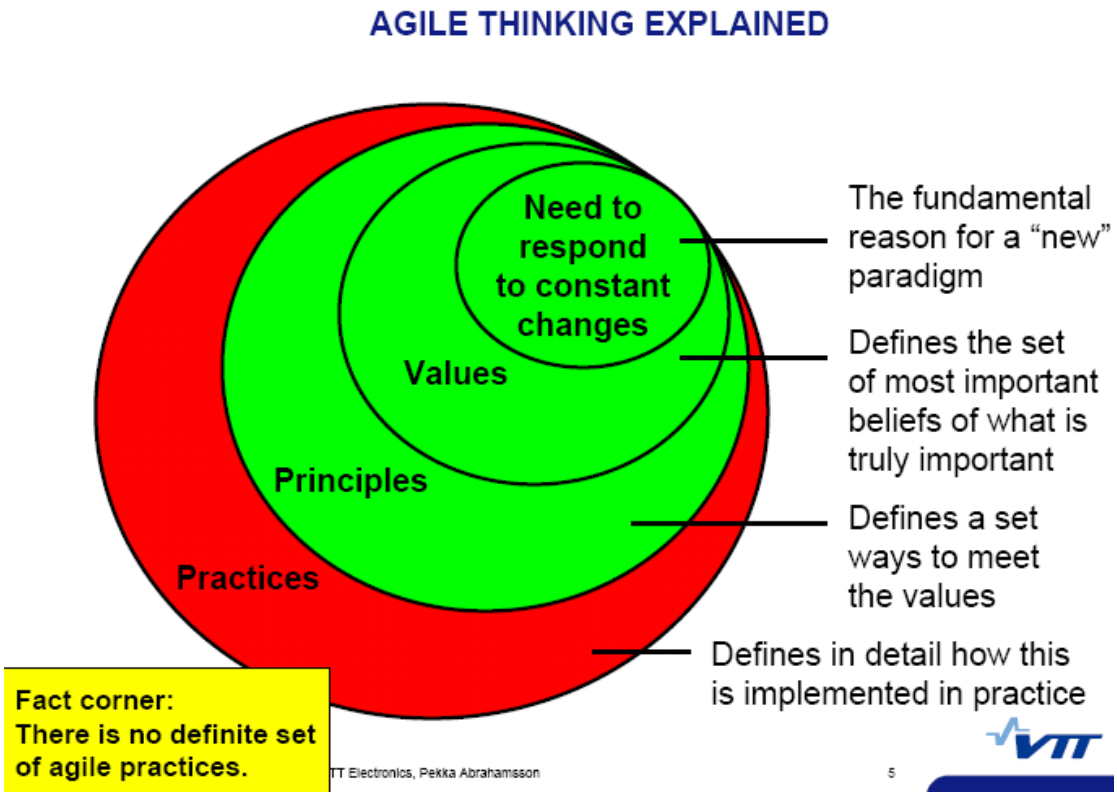
10. Aplicación de XP en la enseñanza de la programación

Un aspecto importante de la cultura de programación promovida por XP, que es la programación en pares, parece ir en contra de lo que se considera correcto en la educación, donde en las asignaciones de programación, normalmente se asume un trabajo individual y si ésta se realizará en "pares", prácticamente se consideraría un acto de plagio ó de acción ilegal. En [Smith 2001] se exponen los beneficios de adoptar prácticas de XP, tales como: programación en pares, refabricación, integración continua, diseño evolutivo y prueba, en beneficio de la enseñanza de la programación y se anima a que se realicen investigaciones que apoyen la integración de estas prácticas en la carga curricular. Similarmente, en [Noble 2004] se exponen algunas modificaciones a las prácticas estándares de XP para poder adaptarlas a la duración normal de un curso que dura un semestre. Por ejemplo, se sugiere que el cliente esté al menos 2 horas a la semana con los estudiantes (turnos completos serían impensables dadas las características de los estudios) para la práctica de la planeación, y posteriormente por

correo electrónico o por teléfono para el resto del tiempo. Por las mismas razones, la práctica de la semana de 40 horas se trasladó a una semana de 10 horas, donde 2 se dedican al avance individual y 8 al trabajo en equipo. Los ciclos de entrega se adaptaron a cada 2 semanas para poder tener al menos 6 iteraciones durante el ciclo escolar. Otras prácticas de XP se incorporaron prácticamente sin cambios, tales como: programación en pares, planeación, pruebas, integración continua, propiedad colectiva del software, estándares de programación, diseño simple y refabricación.

11. Conclusiones

A seis años de existir públicamente la Programación Extrema, se ha mostrado como una alternativa para cierto tipo de proyectos, sobretodo en aquellos donde el cambio en la tecnología ó en los requerimientos es la constante principal durante todas las etapas de desarrollo. Es precisamente en estas condiciones que otras metodologías que requieren gran inversión de tiempo en las etapas iniciales de análisis de requerimientos y diseño, muestran una inflexibilidad importante a adaptarse a cambios continuos y conllevan un riesgo de terminar en proyectos nunca implementados o insostenibles a mediano y largo plazo. Si bien, la adopción de las 12 prácticas de XP parece no ser cuestión de decisión, ya que deben implementarse en "paquete" para que ofrezcan los beneficios argumentados, se ha encontrado que en la práctica, diferentes organizaciones se sienten atraídas por esta metodología pero hacen algunas modificaciones de acuerdo a su cultura y al éxito que se va teniendo con la adopción de las diferentes prácticas. Hasta este momento no se han encontrado estudios concluyentes que cuantifiquen de alguna manera el grado de éxito obtenido al adoptar este paradigma, y poder contrastarlo con el que se obtiene al modificarlo, tal vez ligeramente, como por ejemplo al eliminar la programación en pares obligatoria (y hacerla flexible o eliminarla completamente), que al parecer es una variante frecuentemente utilizada. Por último, también encontramos especialmente interesante el enfoque de introducir en algún grado los principios de XP en la enseñanza de la programación y de las modificaciones que es necesario hacer al modelo original para que sea sostenible en el entorno educativo.



El pensamiento ágil explicado. Fuente: Pekka Abrahamsson. Agile–Finland Seminar. 2005.

Referencias

Aiken Jason, 2004. *Technical and Human Perspectives on Pair Programming*, USA, ACM SIGSOFT Software Engineering Notes, Vol. 29, No. 5.

Amber W. Scott, 2002. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*, John Wiley & Sons, Inc., NY.
Beck Kent, 2000. *Extreme Programming Explained: embrace change*, USA, Addison-Wesley.

Martin Fowler, Beck Kent, 2000. *Planning Extreme Programming Applied*, USA, Addison-Wesley.

English Arthur, 2002. *Extreme Programming: It's Worth a Look*, USA, IEEE IT Pro May/Jun, Vol. 4, No. 3, pp 48-50.

Glass Robert, 2001. *Extreme Programming: The Good, the Bad, and the Bottom Line*. IEEE Software Nov/Dic, Vol. 18, No. 6, pp. 111-112.

Lan Cao et al., 2004. *How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-Scale Projects*, USA, IEEE Proceedings of the 37th Hawaii International Conference on System Sciences. p.85-95.

Layman Lucas, 2004. *Empirical Investigation of the Impact of Extreme Programming Practices on Software Projects*, USA, OOPSLA '04: 19th ACM SIGPLAN conference on object-oriented programming systems, languages, and applications. pp 328-329.

Newkirk, James, 2002. *Introduction to Agile Processes and Extreme Programming*. Proceedings of the 24th International Conference on Software Engineering. Pp. 695-696.

Noble James et al., 2004. *Less Extreme Programming*. Proceedings of the 6th conference on Australian computing education. Vol. 30, pp. 217-226.

Pekka Abrahamsson et al., 2003. *New Directions on Agile Methods: A comparative* Analysis. Proceedings of the 25th International Conference on Software Engineering. Pp. 244-255.

Smith Suzanne et al., 2001. *What We Can Learn From Extreme Programming*. Journal of Computing Sciences in Colleges. Vol. 17, No. 2.

Strigel Wolfgang , 2001. *Reports from the Field.* , USA, IEEE Software. Nov.-Dic. 2001 pp 17-18.

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more

http://www.agilemanifesto.org/