Métodologías Ágiles en el Desarrollo de Software

José H. Canós, Patricio Letelier y Mª Carmen Penadés

DSIC -Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia
{ jhcanos | letelier | mpenades }@dsic.upv.es

RESUMEN

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales. En este trabajo se presenta resumidamente el contexto en el que surgen las metodologías ágiles, sus valores, principios y comparación con las metodologías tradicionales. Además se describen brevemente las principales propuestas, especialmente Programación Extrema (eXtreme Programming, XP) la metodología ágil más popular en la actualidad.

PALABRAS CLAVE. Procesos de Software, Metodologías Ágiles, Programación Extrema (XP)

1. INTRODUCCIÓN

En las dos últimas décadas las notaciones de modelado y posteriormente las herramientas pretendieron ser las "balas de plata" para el éxito en el desarrollo de software, sin embargo, las expectativas no fueron satisfechas. Esto se debe en gran parte a que otro importante elemento, la metodología de desarrollo, había sido postergado. De nada sirven buenas notaciones y herramientas si no se proveen directivas para su aplicación. Así, esta década ha comenzado con un creciente interés en metodologías de desarrollo. Hasta hace poco el proceso de desarrollo llevaba asociada un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del "buen hacer" de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico. Por estar especialmente orientadas para proyectos pequeños, las metodologías ágiles constituyen una solución a medida para ese entorno, aportando una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto.

Las metodologías ágiles son sin duda uno de los temas recientes en ingeniería de software que están acaparando gran interés. Prueba de ello es que se están haciendo un espacio destacado en

la mayoría de conferencias y workshops celebrados en los últimos años. Es tal su impacto que actualmente existen 4 conferencias internacionales de alto nivel y específicas sobre el tema[1]. Además ya es un área con cabida en prestigiosas revistas internacionales. En la comunidad de la ingeniería del software, se está viviendo con intensidad un debate abierto entre los partidarios de las metodologías tradicionales (referidas peyorativamente como "metodologías pesadas") y aquellos que apoyan las ideas emanadas del "Manifiesto Ágil"[2]. La curiosidad que siente la mayor parte de ingenieros de software, profesores, e incluso alumnos, sobre las metodologías ágiles hace prever una fuerte proyección industrial. Por un lado, para muchos equipos de desarrollo el uso de metodologías tradicionales les resulta muy lejano a su forma de trabajo actual considerando las dificultades de su introducción e inversión asociada en formación y herramientas. Por otro, las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles, y/o basados en nuevas tecnologías.

El artículo está organizado como sigue. En la sección 2 se introducen las principales características de las metodologías ágiles, recogidas en el Manifiesto y se hace una comparación con las tradicionales. La sección 3 se centra en *eXtreme Programming* (XP), presentando sus características particulares, el proceso que se sigue y las prácticas que propone. En la sección 4 se citan otros métodos ágiles, enumerándose sus principales características. Finalmente aparecen las conclusiones.

2. METODOLOGÍAS ÁGILES

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término "ágil" aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas.

Tras esta reunión se creó *The Agile Alliance*[3], una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos. El punto de partida es fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".

2.1. El Manifiesto Ágil.

Según el Manifiesto se valora:

- **Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.** La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- **Desarrollar software que funciona más que conseguir una buena documentación**. La regla a seguir es "no producir documentos a menos que sean necesarios de forma inmediata para tomar un decisión importante". Estos documentos deben ser cortos y centrarse en lo fundamental.

[1] XP Agile Universe: `www.agileuniverse.com`. Conference on eXtreme Programming and Agile Processes in Software Engineering: `www.xp2004.org`. Agile Development Conference (EEUU): `www.agiledevelopmentconference.com`. Agile Development Conference (Australia): `www.softed.com/adc2003/`

[2] `agilemanifesto.org`

[3] `www.agilealliance.com`

- **La colaboración con el cliente más que la negociación de un contrato**. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.
- **Responder a los cambios más que seguir estrictamente un plan**. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

Los valores anteriores inspiran los doce principios del manifiesto. Son características que diferencian un proceso ágil de uno tradicional. Los dos primeros principios son generales y resumen gran parte del espíritu ágil. El resto tienen que ver con el proceso a seguir y con el equipo de desarrollo, en cuanto metas a seguir y organización del mismo. Los principios son:

I. *La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.*
II. *Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.*
III. *Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.*
IV. *La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.*
V. *Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.*
VI. *El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.*
VII. *El software que funciona es la medida principal de progreso.*
VIII. *Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.*
IX. *La atención continua a la calidad técnica y al buen diseño mejora la agilidad.*
X. *La simplicidad es esencial.*
XI. *Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.*
XII. *En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.*

2.2. Comparación

La Tabla 1 recoge esquemáticamente las principales diferencias de las metodologías ágiles con respecto a las tradicionales ("no ágiles"). Estas diferencias que afectan no sólo al proceso en sí, sino también al contexto del equipo así como a su organización.

3. PROGRAMACIÓN EXTREMA (*EXTREME PROGRAMMING*, XP)

XP[4] [2] es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

[4] www.extremeprogramming.org, www.xprogramming.com, c2.com/cgi/wiki?ExtremeProgramming

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 1. Diferencias entre metodologías ágiles y no ágiles

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Kent Beck, el padre de XP, describe la filosofía de XP en [2] sin cubrir los detalles técnicos y de implantación de las prácticas. Posteriormente, otras publicaciones de experiencias se han encargado de dicha tarea. A continuación presentaremos las características esenciales de XP organizadas en los tres apartados siguientes: historias de usuario, roles, proceso y prácticas.

3.1. Las Historias de Usuario

Son la técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas [12].

Beck en su libro [2] presenta un ejemplo de ficha (*customer story and task card*) en la cual pueden reconocerse los siguientes contenidos: fecha, tipo de actividad (nueva, corrección, mejora), prueba funcional, número de historia, prioridad técnica y del cliente, referencia a otra historia previa, riesgo, estimación técnica, descripción, notas y una lista de seguimiento con la fecha, estado cosas por terminar y comentarios. A efectos de planificación, las historias pueden ser de una a tres semanas de tiempo de programación (para no superar el tamaño de una iteración). Las historias de usuario son descompuestas en tareas de programación (task card) y asignadas a los programadores para ser implementadas durante una iteración.

3.2. Roles XP

Los roles de acuerdo con la propuesta original de Beck son:

- **Programador.** El programador escribe las pruebas unitarias y produce el código del sistema.

- **Cliente.** Escribe las historias de usuario y las pruebas funcionales para validar su implementación. Además, asigna la prioridad a las historias de usuario y decide cuáles se implementan en cada iteración centrándose en aportar mayor valor al negocio.

- **Encargado de pruebas (*Tester*)**. Ayuda al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

- Encargado de seguimiento (*Tracker*). Proporciona realimentación al equipo. Verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado, para mejorar futuras estimaciones. Realiza el seguimiento del progreso de cada iteración.

- Entrenador (*Coach*). Es responsable del proceso global. Debe proveer guías al equipo de forma que se apliquen las prácticas XP y se siga el proceso correctamente.

- Consultor. Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan surgir problemas.

- Gestor (*Big boss*). Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente creando las condiciones adecuadas. Su labor esencial es de coordinación.

3.3. Proceso XP

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos [12]:

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración.

El ciclo de vida ideal de XP consiste de seis fases [2]: Exploración, Planificación de la Entrega (*Release*), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto.

3.4. Prácticas XP

La principal suposición que se realiza en XP es la posibilidad de disminuir la mítica curva exponencial del costo del cambio a lo largo del proyecto, lo suficiente para que el diseño evolutivo funcione. Esto se consigue gracias a las tecnologías disponibles para ayudar en el desarrollo de software y a la aplicación disciplinada de las siguientes prácticas.

- **El juego de la planificación**. Hay una comunicación frecuente el cliente y los programadores. El equipo técnico realiza una estimación del esfuerzo requerido para la implementación de las historias de usuario y los clientes deciden sobre el ámbito y tiempo de las entregas y de cada iteración.
- **Entregas pequeñas**. Producir rápidamente versiones del sistema que sean operativas, aunque no cuenten con toda la funcionalidad del sistema. Esta versión ya constituye un resultado de valor para el negocio. Una entrega no debería tardar más 3 meses.
- **Metáfora**. El sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
- **Diseño simple**. Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- **Pruebas**. La producción de código está dirigida por las pruebas unitarias. Éstas son son establecidas por el cliente antes de escribirse el código y son ejecutadas constantemente ante cada modificación del sistema.

- **Refactorización (*Refactoring*)**. Es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo [8].
- **Programación en parejas**. Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva ventajas implícitas (menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, …).
- **Propiedad colectiva del código**. Cualquier programador puede cambiar cualquier parte del código en cualquier momento.
- **Integración continua**. Cada pieza de código es integrada en el sistema una vez que esté lista. Así, el sistema puede llegar a ser integrado y construido varias veces en un mismo día.
- **40 horas por semana**. Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse. El trabajo extra desmotiva al equipo.
- **Cliente in-situ**. El cliente tiene que estar presente y disponible todo el tiempo para el equipo. Éste es uno de los principales factores de éxito del proyecto XP. El cliente conduce constantemente el trabajo hacia lo que aportará mayor valor de negocio y los programadores pueden resolver de manera inmediata cualquier duda asociada. La comunicación oral es más efectiva que la escrita.
- **Estándares de programación**. XP enfatiza que la comunicación de los programadores es a través del código, con lo cual es indispensable que se sigan ciertos estándares de programación para mantener el código legible.

El mayor beneficio de las prácticas se consigue con su aplicación conjunta y equilibrada puesto que se apoyan unas en otras. Esto se ilustra en la Figura 1 (obtenida de [2]), donde una línea entre dos prácticas significa que las dos prácticas se refuerzan entre sí. La mayoría de las prácticas propuestas por XP no son novedosas sino que en alguna forma ya habían sido propuestas en ingeniería del software e incluso demostrado su valor en la práctica (ver [1] para un análisis histórico de ideas y prácticas que sirven como antecedentes a las utilizadas por las metodologías ágiles). El mérito de XP es integrarlas de una forma efectiva y complementarlas con otras ideas desde la perspectiva del negocio, los valores humanos y el trabajo en equipo.

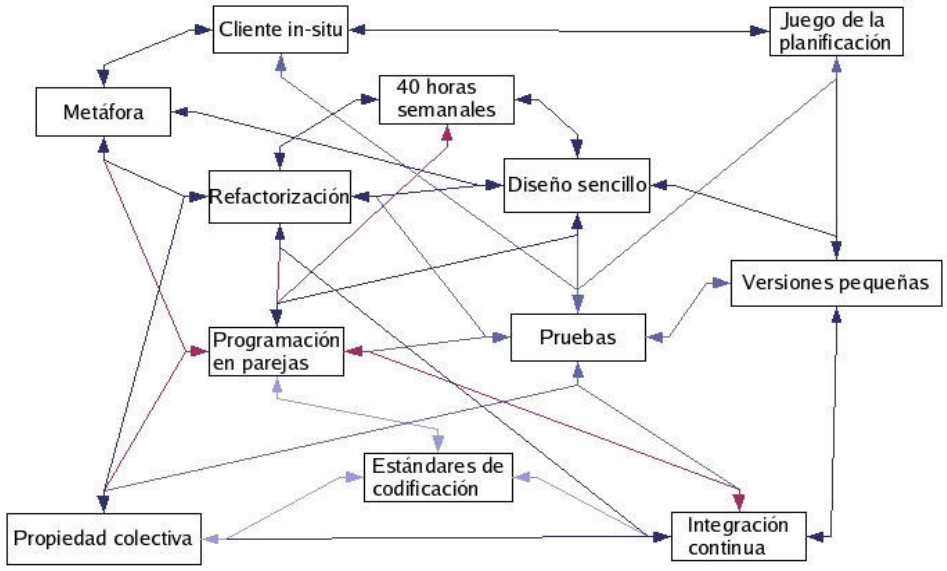

Figura 1. Las prácticas se refuerzan entre sí

3. OTRAS METODOLOGÍAS ÁGILES

Aunque los creadores e impulsores de las metodologías ágiles más populares han suscrito el manifiesto ágil y coinciden con los principios enunciados anteriormente, cada metodología tiene características propias y hace hincapié en algunos aspectos más específicos. A continuación se resumen otras metodologías ágiles. La mayoría de ellas ya estaban siendo utilizadas con éxito en proyectos reales pero les faltaba una mayor difusión y reconocimiento.

- **SCRUM**[5] [16]. Desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración.

- **Crystal Methodologies**[6] [5]. Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros).

- **Dynamic Systems Development Method**[7] (**DSDM**) [17]. Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: estudio viabilidad, estudio del negocio, modelado funcional, diseño y construcción, y finalmente implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases.

- **Adaptive Software Development**[8] (**ASD**) [9]. Su impulsor es Jim Highsmith. Sus principales características son: iterativo, orientado a los componentes software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: especulación, colaboración y aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

- **Feature-Driven Development**[9] (**FDD**) [3]. Define un proceso iterativo que consta de 5 pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Sus impulsores son Jeff De Luca y Peter Coad.

- **Lean Development**[10] (**LD**) [15]. Definida por Bob Charette's a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. En LD, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la

[5] www.controlchaos.com
[6] www.crystalmethodologies.org
[7] www.dsdm.org
[8] www.adaptivesd.com
[9] www.featuredrivendevelopment.com
[10] www.poppendieck.com

productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios.

4. CONCLUSIONES

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc. Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes. Las metodologías ágiles ofrecen una solución casi a medida para una gran cantidad de proyectos que tienen estas características. Una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo. Esto ha llevado hacia un interés creciente en las metodologías ágiles. Sin embargo, hay que tener presente una serie de inconvenientes y restricciones para su aplicación, tales como: están dirigidas a equipos pequeños o medianos (Beck sugiere que el tamaño de los equipos se limite de 3 a 20 como máximo, otros dicen no más de 10 participantes), el entorno físico debe ser un ambiente que permita la comunicación y colaboración entre todos los miembros del equipo durante todo el tiempo, cualquier resistencia del cliente o del equipo de desarrollo hacia las prácticas y principios puede llevar al proceso al fracaso (el clima de trabajo, la colaboración y la relación contractual son claves), el uso de tecnologías que no tengan un ciclo rápido de realimentación o que no soporten fácilmente el cambio, etc.

Falta aún un cuerpo de conocimiento consensuado respecto de los aspectos teóricos y prácticos de la utilización de metodologías ágiles, así como una mayor consolidación de los resultados de aplicación. La actividad de investigación está orientada hacia líneas tales como: métricas y evaluación del proceso, herramientas específicas para apoyar prácticas ágiles, aspectos humanos y de trabajo en equipo. Entre estos esfuerzos destacan proyectos como NAME[11] (*Network for Agile Methodologies Experience*) en el cual hemos participado como nodo en España.

Aunque en la actualidad ya existen libros asociados a cada una de las metodologías ágiles existentes y también abundante información en internet, es XP la metodología que resalta por contar con la mayor cantidad de información disponible y es con diferencia la más popular.

BIBLIOGRAFIA

[1] Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. "Agile software development methods Review and analysis". VTT Publications. 2002.

[2] Beck, K.. "Extreme Programming Explained. Embrace Change", Pearson Education, 1999. Traducido al español como: "Una explicación de la programación extrema. Aceptar el cambio", Addison Wesley, 2000.

[3] Coad P., Lefebvre E., De Luca J. "Java Modeling In Color With UML: Enterprise Components and Process". Prentice Hall. 1999.

[4] Cockbun, A. "Agile Software Development". Addison-Wesley. 2001.

[5] Fowler, M., Beck, K., Brant, J. "Refactoring: Improving the Design of Existing Code". Addison-Wesley. 1999

[6] Highsmith J., Orr K. "Adaptive Software Development: A Collaborative Approach to Managing Complex Systems". Dorset House. 2000.

[7] Jeffries, R., Anderson, A., Hendrickson, C. "Extreme Programming Installed". Addison-Wesley. 2001

[8] Poppendieck M., Poppendieck T. "Lean Software Development: An Agile Toolkit for Software Development Managers". Addison Wesley. 2003.

[9] Schwaber K., Beedle M., Martin R.C. "Agile Software Development with SCRUM". Prentice Hall. 2001.

[10] Stapleton J. "Dsdm Dynamic Systems Development Method: The Method in Practice". Addison-Wesley. 1997.

[11] name.case.unibz.it/