Revisión de metodologías ágiles para el desarrollo de software

A review of agile methodologies for software development

Andrés Navarro Cadavid[1], Juan Daniel Fernández Martínez[2], Jonathan Morales Vélez[3]

[1]Doctor Ingeniero en Telecomunicaciones, Grupo de investigación i2T, Universidad Icesi
Email: anavarro@icesi.edu.co
[2]Ingeniero de Sistemas e Ingeniero Telemático, Grupo de investigación i2T, Universidad Icesi
[3]Ingeniero de Sistemas, Grupo de investigación i2T, Universidad Icesi

Recibido 04/06/13, Aceptado 20/09/2013

RESUMEN

En los años noventa surgieron metodologías de desarrollo de software ligeras –luego llamadas ágiles– dirigidas a reducir la probabilidad de fracaso por subestimación de costos, tiempos y funcionalidades en los proyectos de desarrollo de software. Se gestaron como alternativa a las metodologías tradicionales, específicamente para reducir la carga burocrática propia ellas, en proyectos de pequeña y mediana escala. A diferencia de las tradicionales, las metodologías ágiles son adaptativas –no predictivas–, y están orientadas a las personas –no a los procesos–. Este documento hace una revisión de publicaciones sobre las metodologías ágiles, sus principios y fundamentos; establece criterios para definir la relevancia de las metodologías ágiles; define y explica con detalle las más relevantes (i.e., Scrum y XP); y presenta las características de otras cuatro destacadas (i.e., DSDM, Crystal, ASD y FDD).

Palabras clave: Scrum, XP, Método de desarrollo de sistemas dinámicos, Crystal, Desarrollo adaptativo de software, Desarrollo orientado a funcionalidades, Metodologías *á*giles.

ABSTRACT

In the nineties appeared lightweight software development methodologies – then called agile - that aimed to reduce the probability of failure due to underestimation of cost, time and functionality in software development projects. Agile methodologies were developed as an alternative to traditional methodologies specifically to reduce the bureaucratic burden in projects of small and medium scale. Unlike traditional methodologies, agile methodologies are adaptive and oriented to people. This document describes the agile methodologies, its principles and fundamentals, establishes a criteria for defining the relevance of the agile methodologies, defines and explains in detail the most relevant (i.e., Scrum and XP) and presents other prominent ones(i.e., DSDM, Crystal, ASD and FDD).

Keywords: Scrum, XP, Dynamic system development method, Crystal, Adaptative software development, Feature-Driven Development, Agile methodologies.

1. INTRODUCCIÓN

En la década de los noventa surgieron metodologías de desarrollo de software ligeras, más adelante nombradas como metodologías ágiles, que buscaban reducir la probabilidad de fracaso por subestimación de costos, tiempos y funcionalidades en los proyectos de desarrollo de software. Estas metodologías nacieron como reacción a las metodologías existentes con el propósito de disminuir la burocracia que implica la aplicación de las metodologías tradicionales en los proyectos de pequeña y mediana escala.

Las metodologías tradicionales buscan imponer disciplina al proceso de desarrollo de software y de esa forma volverlo predecible y eficiente. Para conseguirlo se soportan en un proceso detallado con énfasis en planeación [1] propio de otras ingenierías. El principal problema de este enfoque es que hay muchas actividades que hacer para seguir la metodología y esto retrasa la etapa de desarrollo. Las metodologías ágiles tienen dos diferencias fundamentales con las metodologías tradicionales; la primera es que los métodos ágiles son adaptativos –no predictivos-. La segunda diferencia es que las metodologías ágiles son orientadas a las personas –no orientadas a los procesos- [1].

Las metodologías ágiles son adaptativas. Este hecho es de gran importancia ya que contrasta con la predictibilidad buscada por las metodologías tradicionales. Con el enfoque de las metodologías ágiles los cambios son eventos esperados que generan valor para el cliente [2].

Este artículo está motivado por la necesidad de encontrar una metodología que se adapte al proceso de desarrollo de sistemas de radio software (SDR) empleando plataformas (hardware) y software abierto, lo que llevó a una revisión de la literatura sobre metodologías ágiles y su respectiva comparación para decidir la más adecuada a este tipo de proyectos.

El resto del documento está organizado de la siguiente forma: en la sección 2 se presenta una descripción de las metodologías ágiles, cuáles son sus principios y fundamentos. En la sección 3 se describen una serie de criterios para definir la relevancia de las metodologías ágiles y se muestran las más relevantes. La sección 4 explica en detalle las metodologías seleccionadas como más relevantes y hace una breve descripción de otras metodologías ágiles destacadas. En la sección 5 se muestran las conclusiones.

2. METODOLOGÍAS ÁGILES

Hablar de metodologías agiles implica hacer referencia a las metodologías de desarrollo de software tradicionales ya que las primeras surgieron como una reacción a las segundas; sus características principales son antagónicas y su uso ideal aplica en contextos diferentes.

2.1 Metodologías tradicionales

Las metodologías tradicionales de desarrollo de software son orientadas por planeación. Inician el desarrollo de un proyecto con un riguroso proceso de elicitación de requerimientos, previo a etapas de análisis y diseño. Con esto tratan de asegurar resultados con alta calidad circunscritos a un calendario.

En las metodologías tradicionales se concibe un solo proyecto, de grandes dimensiones y estructura definida; se sigue un proceso secuencial en una sola dirección y sin marcha atrás; el proceso es rígido y no cambia; los requerimientos son acordados de una vez y para todo el proyecto, demandando grandes plazos de planeación previa y poca comunicación con el cliente una vez ha terminado ésta [3].

2.2 Metodologías Ágiles

Las metodologías ágiles son flexibles, pueden ser modificadas para que se ajusten a la realidad de cada equipo y proyecto.

Los proyectos ágiles se subdividen en proyectos más pequeños mediante una lista ordenada de características. Cada proyecto es tratado de manera independiente y desarrolla un subconjunto de características durante un periodo de tiempo corto, de entre dos y seis semanas. La comunicación con el cliente es constante al punto de requerir un representante de él durante el desarrollo. Los proyectos son altamente colaborativos y se adaptan mejor a los cambios; de hecho, el cambio en los requerimientos es una característica esperada y deseada, al igual que las entregas constantes al cliente y la retroalimentación por parte de él. Tanto el producto como el proceso son mejorados frecuentemente [4].

2.3 Comparación entre metodologías

La tabla 1 muestra aspectos relevantes de las metodologías de desarrollo tradicional contrastándolas con los aspectos relevantes de las metodologías de desarrollo ágil.

Tabla 1. Metodologías tradiciones vs metodologías Ágiles
Table 1. Traditional methodologies vs Agile methodologies

Metodologías tradicionales	Metodologías ágiles
Predictivos	Adaptativos
Orientados a procesos	Orientados a personas
Proceso rígido	Proceso flexible
Se concibe como un proyecto	Un proyecto es subdividido en varios proyectos más pequeños
Poca comunicación con el cliente	Comunicación constante con el cliente
Entrega de software al finalizar el desarrollo	Entregas constantes de software
Documentación extensa	Poca documentación

2.4 Manifiesto por el desarrollo Ágil

Las metodologías ágiles son flexibles, sus proyectos son subdivididos en proyectos más pequeños, incluyen comunicación constante con el cliente, son altamente colaborativos y se adaptan mejor a los cambios. De hecho, el cambio en los requerimientos es una característica esperada al igual que las entregas constantes al cliente y la retroalimentación por parte de él. Tanto el producto como el proceso son mejorados frecuentemente [4].

En 2001 se crea el *Manifiesto por el desarrollo ágil de software*, documento en el que se acuerdan cuatro principios básicos para el desarrollo de software, que establece prioridades y marca diferencias de fondo frente a los sistemas tradicionales: individuos e interacciones, por encima de procesos y herramientas; software funcionando, por encima

de documentación extensiva; colaboración con el cliente, por encima de negociación contractual; y respuesta ante el cambio, por encima de seguir un plan [5].

Los principios que dan origen al manifiesto implican la satisfacción del cliente mediante entregas tempranas y continuas de software que funcione; requerimientos cambiantes en cualquier etapa del proyecto; participación activa del cliente; simplicidad; equipos de desarrollo motivados y auto-organizados; comunicación efectiva; auto inspecciones; y adaptación [5].

El manifiesto por el desarrollo ágil de software es el resultado del trabajo colaborativo de un grupo formado por diecisiete personas, entre desarrolladores de software, escritores y consultores, quienes lo construyeron y suscribieron en 2001. La firma y publicación del Manifiesto en ese año no implica que esa sea la fecha de origen de las metodologías ágiles o que antes de ese año no existieran, sino el reconocimiento de la necesidad −y la expresión− de un lineamiento común capaz de hacer posible algún tipo de agrupación entre ellas [6].

Las metodologías ágiles se caracterizan por el desarrollo iterativo e incremental; la simplicidad de la implementación; las entregas frecuentes; la priorización de los requerimientos o características a desarrollar a cargo del cliente; y la cooperación entre desarrolladores y clientes. Las metodologías ágiles dan como un hecho que los requerimientos van a cambiar durante el proceso de desarrollo [7].

3. SELECCIÓN DE METODOLOGÍAS

En la revisión de las metodologías ágiles para el propósito de seleccionar aquellas en las que se enfocará este artículo se sigue una metodología simple, propuesta por los autores, y que se describe a continuación.

Una primera selección surge del manifiesto: *Scrum*, *Extreme Programming* [XP], *Dynamic System Development Method* [DSDM], *Crystal*, *Adaptative Software Development* [ASD] y *Feature-Driven Development* [FDD], están representadas en él, a través de al menos una de las personas que lo suscribieron [5].

Otra aproximación para definir las metodologías ágiles más relevantes consiste en revisar las que han sido citadas y explicadas en libros de ingeniería de software. Al respecto, Sommerville [6] indica que las metodologías ágiles más conocidas son *XP*, Scrum, Crystal, ASD, DSDM y FDD, mientras que Pressman [8] centra su trabajo en la explicación del funcionamiento de XP, DSDM, ASD, Scrum, FDD y Agile Modelling [AM].

Una revisión de las investigaciones, revisiones e implementaciones referenciadas por asociaciones como IEEE y ACM, arroja un tercer parámetro [9]. Las metodologías más populares en los documentos científicos entre el 2003 y el 2007 fueron XP y Scrum. Sin embargo, cabe aclarar que la mayoría de lo que se publicó no está relacionado con una metodología en particular, sino con el concepto de metodologías ágiles en general; de hecho, el eje común a los temas tratados con mayor frecuencia es la presentación de experiencias exitosas de implantación [10].

Otro criterio de selección es el reconocimiento de la alta adopción en la industria de desarrollo de software en los últimos años. Según la séptima versión de la encuesta anual del estado del desarrollo de software usando metodologías ágiles, el tamaño promedio de las organizaciones es de 100 empleados. De quienes contestaron la encuesta, el 84% dijo que en sus compañías se practican las metodologías ágiles, lo que en algunos casos se ha venido haciendo por más de 5 años.

La tendencia de definir a XP y a Scrum como los métodos de desarrollo ágil más relevantes es notoria, así se evidencia en los trabajos de Jiang y Eberlein [11], Thorstein, Hannay, Pfahl, Benestad, y Langtangen [12], Silva, Selbach, Maurer, y Hellmann [13], Maurer y Hellmann [14], Vijayasarathy y Turk [15], Hoda, Kruchten, Noble, y Marshal [16] y Szöke [17], así como en encuestas hechas por importantes proveedores de software para la gestión de este tipo de metodologías [18, 19]. Thorstein et ál incluso listan *todas* las prácticas ágiles existentes, tomando en cuenta solamente XP y Scrum [12].

La mayoría de equipos ágiles exitosos han adaptado prácticas ágiles de distintas metodologías para generar un proceso de desarrollo propio que se ajusta a sus necesidades [20]. Estas adaptaciones parecen estar centradas en mezclas de Scrum con XP [21]. XP se enfoca en prácticas de desarrollo mientras que Scrum apunta a la administración de proyectos [16].

Con base en lo anterior, en el resto del artículo se centra en la presentación de estas dos metodologías y una referencia a las demás destacadas en la literatura.

4. METODOLOGÍAS ÁGILES REPRESENTATIVAS

4.1 Scrum

Su nombre no corresponde a una sigla, sino a un concepto deportivo, propio del rugby, relacionado con la formación requerida para la recuperación rápida del juego ante una infracción menor [22]. Su primera referencia en el contexto de desarrollo data de 1986, cuando Takeuchi y Nonaka utilizan el *Rugby Approach* para definir un nuevo enfoque en el desarrollo de productos, dirigido a incrementar su flexibilidad y rapidez, a partir de la integración de un

equipo interdisciplinario y múltiples fases que se traslapan entre sí [23].

La metodología Scrum para el desarrollo ágil de software es un marco de trabajo diseñado para lograr la colaboración eficaz de equipos en proyectos, que emplea un conjunto de reglas y artefactos y define roles que generan la estructura necesaria para su correcto funcionamiento.

Scrum utiliza un enfoque incremental que tiene como fundamento la teoría de control empírico de procesos. Esta teoría se fundamenta en transparencia, inspección y adaptación; la transparencia, que garantiza la visibilidad en el proceso de las cosas que pueden afectar el resultado; la inspección, que ayuda a detectar variaciones indeseables en el proceso; y la adaptación, que realiza los ajustes pertinentes para minimizar el impacto de las mismas [24].

Los llamados Equipos Scrum son auto-gestionados, multifuncionales y trabajan en iteraciones. La autogestión les permite elegir la mejor forma de hacer el trabajo, en vez de tener que seguir lineamientos de personas que no pertenecen al equipo y carecen de contexto. Los integrantes del equipo tienen todos los conocimientos necesarios (por ser multifuncionales) para llevar a cabo el trabajo. La entrega del producto se hace en iteraciones; cada iteración crea nuevas funcionalidades o modifica las que el dueño del producto requiera.

Scrum define tres roles: el *Scrum master*, el dueño del producto y el equipo de desarrollo [25]. El Scrum master tiene como función asegurar que el equipo está adoptando la metodología, sus prácticas, valores y normas; es el líder del equipo pero no gestiona el desarrollo [25]. El dueño del producto es una sola persona y representa a los interesados, es el responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo; tiene entre sus funciones gestionar la lista ordenada de funcionalidades requeridas o *Product Backlog*. El equipo de desarrollo, por su parte, tiene como responsabilidad convertir lo que el cliente quiere, el Product Backlog, en iteraciones funcionales del producto; el equipo de desarrollo no tiene jerarquías, todos sus miembros tienen el mismo nivel y cargo: desarrollador. El tamaño óptimo del equipo está entre tres y nueve personas.

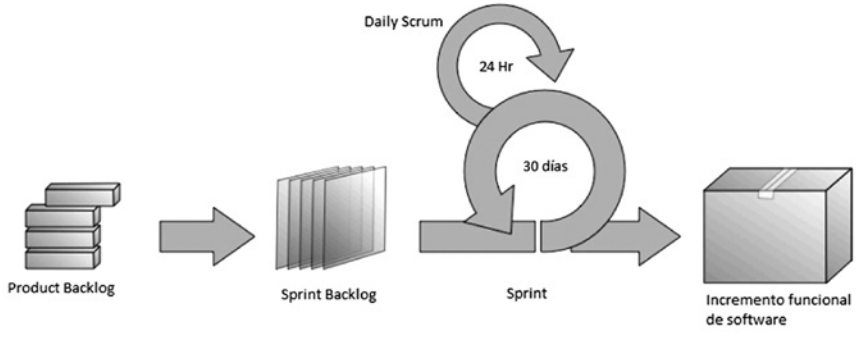
Scrum define un evento principal o *Sprint* (figura 1) que corresponde a una ventana de tiempo donde se crea una versión utilizable del producto (incremento). Cada Sprint, como en el rugby, es considerado como un proyecto independiente. Su duración máxima es de un mes. Un Sprint se compone de los siguientes elementos: reunión de planeación del Sprint, *Daily Scrum*, trabajo de desarrollo, revisión del Sprint y retrospectiva del Sprint.

En la reunión de *Planeación del Sprint* se define su plan de trabajo: qué se va a entregar y cómo se logrará. Es decir, el diseño del sistema y la estimación de cantidad de trabajo. Esta actividad dura ocho horas para un Sprint de un mes. Si el Sprint tiene una duración menor, se asigna el tiempo de manera proporcional.

El *Daily Scrum* es un evento del equipo de desarrollo de quince minutos, que se realiza cada día con el fin de explicar lo que se ha alcanzado desde la última reunión; lo que se hará antes de la siguiente; y los obstáculos que se han presentado. Este evento se desarrolla mediante una reunión que normalmente es sostenida de pie con los participantes reunidos formando un círculo, esto, para evitar que la discusión se extienda [26].

La *Revisión del Sprint* ocurre al final del Sprint y su duración es de cuatro horas para un proyecto de un mes (o una proporción de ese tiempo si la duración es menor). En esta etapa: el dueño del proyecto revisa lo que se hizo, identifica lo que no se hizo y discute acerca del Product Backlog; el equipo de desarrollo cuenta los problemas que encontró y la manera en que fueron resueltos, y muestra el producto y su funcionamiento. Esta reunión es de gran importancia para los siguientes Sprints.

Figura 1. Metodología Scrum: Fases de un Sprint [27]
Figure 1. Scrum Methodology: Sprint phases [27]



La *Retrospectiva del Sprint* es una reunión de tres horas del equipo Scrum en la que se analiza cómo fue la comunicación, el proceso y las herramientas; qué estuvo bien, qué no, y se crea un plan de mejoras para el siguiente Sprint. El tiempo, tal como en los casos anteriores, se debe ajustar proporcionalmente en el caso de proyectos de duración menor a un mes.

Existen también *Artefactos de Scrum*. Estos son subproductos de las actividades del marco de trabajo que le brindan dirección y transparencia al equipo [28]. Los artefactos de Scrum son: Product Backlog, Sprint Backlog, Monitoreo de Progreso e Incremento.

El *Product Backlog* es una lista −ordenada por valor, riesgo, prioridad y necesidad− de los requerimientos que el dueño del producto define, actualiza y ordena. La lista tiene como característica particular que nunca está terminada, pues evoluciona durante el desarrollo del proyecto.

El Sprint Backlog es un subconjunto de ítems del Product Backlog y el plan para realizar en el *Incremento* del producto. Debido a que el Product backlog está organizado por prioridad, el Sprint backlog es construido con los requerimientos más prioritarios del Product backlog y con aquellos que quedaron por resolver en el Sprint anterior. Una vez construido, el Sprint backlog debe ser aceptado por el equipo de desarrollo, pertenece a éste y solo puede ser modificado por él. Requerimientos adicionales deben ser incluidos en el Product backlog y desarrollados en el siguiente Sprint, si su prioridad así lo indica.

El *Monitoreo de Progreso* consiste en la suma del trabajo que falta por realizar en el Sprint. Tiene como característica que se puede dar en cualquier momento, lo que le permite al dueño del producto evaluar el progreso del desarrollo. Para que esto sea posible, los integrantes del equipo actualizan constantemente el estado de los requerimientos que tienen asignados indicando cuánto consideran que les falta por terminar.

El *Incremento* es la suma de todos los ítems terminados en el Sprint backlog. Si hay ítems incompletos deben ser devueltos al Product backlog con una prioridad alta para que sean incluidos en el siguiente Sprint. Se considera que un ítem está terminado si es funcional. La suma de ítems terminados es el producto a entregar.

El ciclo de vida de este marco de trabajo está compuesto de cuatro fases: planeación, puesta en escena, desarrollo y entrega [29]. En la planeación se establece la visión, se fijan las expectativas y se asegura el financiamiento. En la puesta en escena se identifican más requerimientos y se priorizan para la primera iteración. En la implementación se desarrolla el sistema, y en la entrega se hace el despliegue operativo.

4.2 Extreme Programming [XP]

XP es la metodología ágil más conocida [30, 31]. Fue desarrollada por Kent Beck buscando guiar equipos de desarrollo de software pequeños o medianos, entre dos y diez desarrolladores, en ambientes de requerimientos imprecisos o cambiantes [32]. XP tiene como base cinco valores: Simplicidad, Comunicación, Retroalimentación, Respeto y Coraje [33].

Estos valores, a su vez, son la base para la definición de sus principios. De ellos, los fundamentales son: la retroalimentación rápida, asumir simplicidad, el cambio incremental, la aceptación del cambio y el trabajo de calidad [32].

Las prácticas de esta metodología se derivan de sus valores y principios y están enfocadas en darle solución a las actividades básicas de un proceso de desarrollo, esto es: escribir código, realizar pruebas, escuchar (planear) y diseñar [32, 8].

Las prácticas de XP incluyen: *planning game*, pequeñas entregas, diseño simple, programación en pareja, pruebas, *refactoring*, integración continua, propiedad común del código, paso sostenible, cliente en sitio, metáfora y estándares de código.

Planning Game define el alcance y la fecha de cumplimiento de una entrega funcional completa −es decir, la fecha de entrega de un *release* que pueda ser puesto en funcionamiento [32]− y divide las responsabilidades entre el cliente y los desarrolladores. El cliente define, utilizando *Historias de Usuario*, una versión simplificada de los tradicionales *Casos de Uso*, los requerimientos de manera general, y precisa su importancia; Con base en ellas, los desarrolladores estiman el costo de implementarlas y se definen las características de una entrega y el número de iteraciones que se necesitarán para terminarla. Para cada iteración el cliente define cuáles de las historias de usuario que componen la entrega funcional desea que se desarrollen. Se pueden crear o modificar historias de usuario en cualquier momento excepto cuando forman parte de una iteración en curso [21].

Entregas Pequeñas se refiere al uso de ciclos cortos de desarrollo (iteraciones) que le muestran software terminado al cliente y obtienen retroalimentación de él. La definición de terminado está relacionada con la pruebas de aceptación.

Diseño Simple indica que el sistema debe ser tan simple como sea posible, en un momento determinado, lo cual implica que los desarrolladores deben preocuparse únicamente por las historias de usuario planeadas para la iteración actual, sin importar cuanto pueden cambiar por funciones futuras [21, 32].

Programación en pareja indica que todo el código debe ser desarrollado por dos programadores. Las parejas deben cambiar con cierta frecuencia para que el conocimiento de todo el sistema quede en todo el equipo, una práctica que fortalece los principios de diseño simple, calidad y propiedad colectiva del código [34].

Las *Pruebas* son la guía del desarrollo del producto ya que los detalles de las historias de usuario (i.e., requerimientos específicos) se obtienen en el desarrollo de la prueba de aceptación. Las pruebas son lo primero que se desarrolla; con base en ellas, se desarrolla el código que las satisfaga. A este concepto se le denomina *Desarrollo orientado a las pruebas* [21].

Refactoring consiste en realizar cambios que mejoren la estructura del sistema sin afectar su funcionamiento. Para garantizar la no afectación, después de cada cambio se corre la prueba unitaria, con el fin de corroborar sus beneficios. Esta práctica ayuda a mantener el código simple.

Integración continua, establece que cada tarea que se completa se integra al sistema, un proceso que puede darse, incluso, varias veces al día [21]. Con cada tarea que se integra al sistema se corren las pruebas unitarias, las cuales validan si lo que ha sido agregado no perjudica las funcionalidades existentes. En ocasiones las pruebas unitarias fallan y es responsabilidad del desarrollador –o de la pareja que integró el código– arreglarlo. Es común encontrar una regla en los proyectos XP que evita que un desarrollador o pareja se enfoque en otra cosa distinta a arreglar los errores que surgieron de la integración de su código con el del sistema. Esta regla va acompañada de otra que evita que otras partes de código sean integradas en el sistema mientras no hayan superado exitosamente las pruebas. Esto pone un poco más de presión en aquellos desarrolladores cuyo código fue incapaz de superar las pruebas.

*Propiedad común del c*ódigo implica que no hay una persona propietaria del código que se está desarrollando o de una porción del mismo, por lo que cualquier desarrollador que esté participando en una iteración puede hacer cambios en el código, siempre que le agregue valor al sistema.

Paso sostenible hace referencia al ritmo de trabajo del equipo. Indica que debe ser adecuado para que sea factible terminar la iteración o la entrega sin trabajar horas extras. La metodología establece además que nunca se debe trabajar horas extras dos semanas consecutivas [32].

Cliente en sitio se refiere a la necesidad de incluir un representante del cliente trabajando a tiempo completo con el equipo de desarrollo. Su función es resolver oportunamente dudas en la implementación de las historias de usuario.

Metáfora establece que todos los implicados en el proyecto deben tener la misma visión del sistema. Para lograrlo deben hacer abstracciones que establezcan un lenguaje común entre el cliente y los desarrolladores. La metáfora es la guía global del desarrollo [32].

Estándares de código son las reglas que definen como se va a escribir la aplicación. La metodología establece que estos estándares deben estar definidos de tal forma que el código en si mismo sirva como un documento [35]. Este tema es de altísima relevancia porque tanto la programación en pareja, como la propiedad común del código, son imposibles sin estos estándares [32].

El ciclo de vida de esta metodología, por su parte, está compuesto por Exploración, Planeación, Iteraciones hacia la primera entrega, *Productionizing* y Mantenimiento [29, 32].

En la fase de *Exploración* se hace una estimación con base en las historias de usuario requeridas para la primera entrega; en la de *Planeación*, el cliente y los programadores definen las historias de usuario que se van a implementar y sus fechas; *Iteraciones hacia la Primera Entrega*, por su parte, se transforma en el calendario acordado con el cliente, expresado en iteraciones, donde cada una de ellas representa historias de usuario implementadas y probadas; en *Productionizing* se afina el funcionamiento del programa y se despliega; y en *Mantenimiento* se continúan realizando mejoras y arreglos, e implementando nuevas funcionalidades.

En 2004 se publicó una versión revisada de XP [36] en la que se modifican los principios y las prácticas. Los nuevos principios hacen más directa la asociación de los valores con las prácticas. Las nuevas prácticas se subdividen en dos categorías: primarias, las de mayor impacto y por lo tanto las primeras que deberían ser implementadas; y coralarias, las más difíciles de implementar y por ello las que deberían adoptarse después de haber adoptado las primarias. Esta versión, sin embargo, no ha tenido gran acogida y la mayoría de los autores siguen tomando como referencia la versión original.

4.3 Otras metodologías

Como se mencionó al inicio de la sección II, las metodologías ágiles no están limitadas a Scrum y XP, sino que incluyen varias más. A continuación se presenta una breve descripción de las más relevantes.

4.3.1 Crystal

La familia de metodologías Crystal se basa en los conceptos de *Rational Unified Process* [RUP] y está compuesta por

Crystal Clear, Crystal Yellow, Crystal Orange y Crystal Red [37]; el nivel de opacidad del color en el nombre indica un mayor número de personas implicadas en el desarrollo, un mayor tamaño del proyecto y, por lo tanto, la necesidad de mayor control en el proceso [7, 38].

La filosofía de Crystal define el desarrollo como un juego cooperativo de invención y comunicación cuya meta principal es entregar software útil, que funcione, y su objetivo secundario, preparar el próximo juego [38].

Los valores compartidos por los miembros de la familia Crystal están centrados en las personas y en la comunicación. Sus principios indican que: el equipo puede reducir trabajo intermedio en la medida que produce código con mayor frecuencia y utiliza mejores canales de comunicación entre las personas; los proyectos evolucionan distinto con el tiempo por lo que las convenciones que el equipo adopta tienen que adecuarse y evolucionar; los cambios en el cuello de botella del sistema determinan el uso de trabajo repetido; y el afinamiento se realiza sobre la marcha.

Existen dos reglas que aplican para toda la familia Crystal. La primera indica que los ciclos donde se crean los incrementos no deben exceder cuatro meses; la segunda, que es necesario realizar un taller de reflexión después de cada entrega para afinar la metodología.

4.3.2 Método de desarrollo de sistemas dinámicos (*Dynamic Systems Development Method* - DSDM)

DSDM es un marco de trabajo creado para entregar la solución correcta en el momento correcto. Utiliza un ciclo de vida iterativo, fragmenta el proyecto en periodos cortos de tiempo y define entregables para cada uno de estos periodos. Tiene roles claramente definidos y especifica su trabajo dentro de periodos de tiempo [39].

El Consorcio DSDM es el responsable del mantenimiento de esta metodología. Su versión actual es DSDM Atern. Los principios de DSDM son: la necesidad del negocio como eje central; las entregas a tiempo; la colaboración; nunca comprometer la calidad; construir de modo incremental sobre una base sólida; el desarrollo iterativo; la comunicación clara y continua; y la demostración de control [39].

4.3.3 Desarrollo adaptativo de software (*Adaptative Software Development* - ASD)

ASD tiene como fundamento la teoría de sistemas adaptativos complejos. Por ello, interpreta los proyectos de software como sistemas adaptativos complejos compuestos por agentes −los interesados−, entornos −organizacional, tecnológico− y salidas −el producto desarrollado−.

El ciclo de vida de ASD está orientado al cambio y se compone de las fases: especulación, colaboración y aprendizaje [40, 41]. Estas fases se caracterizan por estar enfocadas en la misión, estar basadas en características, ser iterativas, tener marcos de tiempo especificados, ser orientadas por los riesgos y ser tolerantes a los cambios.

4.3.4 Desarrollo orientado a funcionalidades (*Feature-Driven Development* - FDD)

FDD tiene como rasgo característico la planeación y el diseño por adelantado. En consecuencia, el modelo de objetos, la lista de características y la planeación se hacen al inicio del proyecto. Las iteraciones son incrementos con características identificadas [42].

Las prácticas que FDD pregona son: el modelado de objetos de dominio (*domain object modeling*), el desarrollo por características, *Class (code) ownership*, los equipos de características o *Feature Teams*, las inspecciones, la construcción regular de planificación (*Regular Build Schedule*), la gestión de configuración y los reportes y visibilidad de los resultados [43].

Su ciclo de vida está compuesto por cinco etapas: el desarrollo de un modelo general, la construcción de la lista de características, la planeación por característica, el diseño por característica y la construcción por característica [44].

FDD se enfoca en las fases de diseño –diseño por característica– y desarrollo –construcción por característica– siendo estas las etapas del proceso que se iteran [45].

5. CONCLUSIONES

Las metodologías ágiles funcionan bien dentro de un contexto específico caracterizado por equipos pequeños de desarrollo, ubicados en el mismo sitio, con clientes que pueden tomar decisiones acerca de los requerimientos y su evolución, con requerimientos que cambian con frecuencia (semanal, mensual), con alcance del proyecto o presupuesto variable, con pocas restricciones legales y con pocas restricciones en el proceso de desarrollo [16]. Por fuera de este ambiente, es común que se presenten inconvenientes derivados de la falta de participación del cliente, los contratos con precio fijo, los proyectos con arquitectura o diseño intensivo o con documentación intensiva, la tasa de cambios lenta y los equipos distribuidos [16].

El papel del cliente es más notorio en el proceso de desarrollo de las metodologías ágiles; uno de los principios del Manifiesto dice que los responsables de negocio y los desarrolladores trabajan juntos de manera cotidiana durante todo el proyecto [6]. Esta condición no es fácil de satisfacer. Los clientes pueden estar separados geográficamente o puede ser muy costoso para ellos mantener un represen-

tante con la capacidad de responder por la totalidad de los requerimientos del sistema que se está desarrollando, de manera permanente.

Para resolver estas limitantes es posible utilizar varias estrategias. La primera, tener múltiples dueños de historias de usuario. Con esto se busca no inhabilitar a un cliente durante todo el desarrollo. Al tener cada historia de usuario *su* dueño, una vez que esta termina, la siguiente se trata como una historia de usuario con un dueño distinto. La segunda estrategia consiste en nombrar un intermediario del cliente, lo que involucra a un desarrollador del equipo que coordina de manera conjunta los requerimientos.

Otra dificultad, independiente del contexto, tiene que ver con el enfoque de la documentación en la evolución y mantenimiento del producto. Se deriva del concepto emitido por los expertos en agilidad, quienes indican que la documentación debe ser breve, precisa y limitada a las funciones fundamentales del sistema, porque tener el producto funcionando, es lo más importante [46].

La visión de que la calidad del sistema se alcanza mediante el desarrollo iterativo y no mediante la documentación trae problemas a largo plazo ya que la documentación es el medio que asiste la transferencia del conocimiento y el mantenimiento y evolución del software; una documentación incompleta genera una rápida degradación y envejecimiento del software [47].

Si se tiene en cuenta que 90% del costo total del software está relacionado con tareas de mantenimiento, es decir con modificaciones que se realizan después de que el producto ha sido entregado [48], parece prudente utilizar algún método de documentación que detalle el proceso de desarrollo de software, más allá de la comunicación cara a cara [49].

El desconocimiento del sistema por parte del equipo de desarrollo, debido a la complejidad en el código, puede generar deterioro en su arquitectura e incertidumbre en los cambios que se han realizado. La implementación de cambios en el sistema es difícil como consecuencia de la dificultad que implica hacer cambios en código que ha sido desarrollado por otra persona o incluso recordar los cambios que se han hecho.

Otras limitaciones tiene relación con: el entrenamiento de nuevos miembros del equipo, que implica ocupar a otros miembros; el monitoreo y control del proceso, ya que no es común que se documente; y el seguimiento del progreso.

El énfasis de Scrum es la administración de proyectos, el de XP, la definición de prácticas técnicas más específicas; por ello pueden ser combinadas en un mismo proyecto [50, 13]. El planning game de XP es comúnmente utilizado para definir el esfuerzo que conlleva desarrollar determinado requerimiento. La suma del esfuerzo de los requerimientos incluidos en un Sprint backlog ayuda al equipo a decidir si acepta o no desarrollar ese Sprint. El historial de las estimaciones de los ítem desarrollados en cada Sprint ayuda al equipo a conocer su velocidad, lo que a su vez es útil para hacer más precisos los tiempos de entrega y, en últimas, incrementa la tasa de éxito en los proyectos.

La propiedad común del código, la metáfora, el paso sostenible y el cliente en sitio, todas características de XP van muy de la mano con las propuestas de Scrum, su igualdad de los miembros del equipo, la ausencia de jerarquía, la constante comunicación y retroalimentación por parte del cliente, la gestión de expectativas y los criterios que definen si un proyecto es exitoso o no.

Respecto a XP cabe recordar que aunque existe una versión publicada en 2004 [36] que revisa y amplía sus principios y métodos, la versión de referencia para la comunidad de desarrolladores sigue siendo la primera, la de 1999 [32]. Esto se aprecia en los trabajos de Sommerville [7], Sato et ál. [30] y Sampaio et al [31], todos ellos publicados después de 2004, que al referirse a XP hacen referencia a las prácticas definidas por Beck en 1999.

Aunque hay diversas metodologías ágiles, hay dos que se han popularizado y a las que se suele hacer mayor referencia. La literatura muestra que es posible combinar varias metodologías de forma exitosa. En el caso que motiva este trabajo, se encuentra que XP es tal vez la metodología más cercana, pudiéndose utilizar en combinación con SCRUM.

Agradecimientos

Este trabajo es un resultado intermedio del proyecto: *Sistema de Monitoreo de Espectro y Benchmarking de Sistemas Móviles, Usando Radio Software de Dominio Público-Simones*, financiado por Colciencias. Proyecto 211750227325, contrato RC- 743-2011.

REFERENCIAS

[1] Fowler, M. (2005). The new methodology [Internet] Disponible desde http://martinfowler.com/articles/newMethodology.html [Acceso Junio 1, 2013].

[2] Patel, A., Seyfi, A., Taghavi, M., Wills, C., Liu, N., Latih, R., & Misra, S. A comparative study of agile, component-based, aspect-oriented and mashup software development methods. Technical Gazette, 19(1), 175-189, 2012.

[3] Khurana, H. & Sohal, J.S. Agile: The necessitate of contemporary software developers. International Journal of Engineering Science & Technology, 3(2), 1031-1039, 2011.

[4] Ghosh, S. (2012). Systemic comparison of the application of EVM in traditional and agile software project [Internet]. Disponible desde http://pm.umd.edu/files/public/documents/student-papers/2011/EVM%20in%20Waterfall%20and%20Agile%20Software%20Project%20by%20Sam%20Ghosh.pdf [Acceso Junio 1, 2013].

[5] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M.,… Thomas, D. (2001). Manifesto for Software Agile Development [Internet], Disponible desde http://agilemanifesto.org/ [Acceso Junio 1, 2013].

[6] Sommerville, I. Software engineering [9ª ed.]. Addison Wesley, Boston, 2010

[7] Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. New Directions on Agile Methods: A Comparative Analysis. *Proceedings 25th International Conference on Software Engineering.* Portland, IEEE, 2003.

[8] Pressman, R. S. Ingeniería del software: un enfoque práctico. McGraw-Hill, México, 2005.

[9] IEEE Standards Association IEEE Standard 830-1998. IEEE recommended practice for software requirements specifications. IEEE Computer Society, Washington DC, 1998.

[10] Hasnain, E. An Overview of Published Agile Studies: A Systematic Literature Review. *Proceedings of the 2010 National Software Engineering Conference.* New York, ACM, 2010.

[11] Jiang, L., & Eberlein, A. Towards A Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies. In *Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral (APOS '08*). New York, ACM, 2008.

[12] Thorstein, M., Hannay, J., Pfahl, D., Benestad, H., & Langtangen, H. A literature review of agile practices and their effects in scientific software development. *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering* (SECSE '11). New York, ACM, 2011

[13] Silva, T., Selbach, S., Maurer, F., & Hellmann, T. User experience design and agile development: from theory to practice. Journal of Software Engineering and Applications, 5(10), 743-751, 2012.

[14] Maurer, F. & Hellmann, T. (2013). People-Centered Software Development: An Overview of Agile Methodologies. En Lecture Notes in Computer Science, 7171, Berlín, Springer, p.185-215.

[15] Vijayasarathy, L. & Turk, D. Agile software development: a survey of early adopters. Journal of Information Technology Management Volume, 19(2), 1-8, 2008.

[16] Hoda, R., Kruchten, P., Noble, J., & Marshal, S. Agility in Context. *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications* (OOPSLA '10). New York, ACM, 2010.

[17] Szöke, Á. (2011). A feature partitioning method for distributed agile release planning [Internet], Disponible desde http://www.fi.upm.es/catedra-ibmrational/sites/www.fi.upm.es.catedra-ibmrational/files/770027.pdf [Acceso Junio 1, 2013].

[18] VersionOne. (2011). State of Agile development survey results [Internet], Disponible desde http://www.versionone.com/state_of_agile_development_survey/11/ [Acceso Junio 1, 2013].

[19] VersionOne (2013). 7th annual state of agile development survey. Atlanta, GA: VersionOne [Internet], Disponible desde http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf [Acceso Junio 1, 2013].

[20] Paige, R., Chivers, H., McDermid, J., & Stephenson, Z. High-integrity extreme programming. En L. Liebrock [Ed.]. *Proceedings of the 2005 ACM symposium on Applied computing (SAC '05)*, New York, ACM, 2005.

[21] Martin, R. & Martin, M. Agile principles, patterns, and practices in c#. Prentice Hall, Westford, 2006.

[22] International Rugby Board (2012, mayo 15). Ley 20, Scrum. En Leyes del juego de Rugby [Internet], Disponible desde http://www.irblaws.com [Acceso Junio 1, 2013].

[23] Takeuchi, H. & Nonaka, I. The new product development game. Harvard Business Review, Ene-Feb, 137-146, 1986.

[24] Scrum Alliance (2012, marzo 28). Scrum: the basics [Internet], Disponible desde http://www.scrumalliance.org/pages/what_is_scrum [Acceso Junio 1, 2013].

[25] Schwaber, K. & Sutherland, J. (2011). The Scrum guide [Internet], Disponible desde http://www.scrumguides.org/ [Acceso Junio 1, 2013].

[26] Hasnain, E., & Hall, T. Introduction to stand-up meetings in agile methods. IAENG Transactions on Engineering Technologies [Special edition of the World Congress on Engineering and Computer Science. AIP Conference Proceedings]. 1127, 110-120, 2009.

[27] Cohn, M. User stories applied: for Agile software development. Addison Wesley, Boston, 2004.

[28] Blankenship, J., Bussa, M., & Millett, S. Pro Agile .NET Development with Scrum. Apress, New York, 2011.

[29] Larman, C. Agile & iterative development: a manager´s guide. Addison-Wesley, Boston, 2003.

[30] Sato, D., Bassi, D., Bravo, M., Goldman, A., & Kon, F. Experiences tracking agile projects: an empirical study. Journal of the Brazilian Computer Society, 12(3), 45-64, 2006.

[31] Sampaio, A., Vasconcelos, A., & Falcone, P. Assessing agile methods: an empirical study. Journal of the Brazilian Computer Society, 10(2), 22-41, 2004.

[32] Beck, K. Extreme Programming Explained: Embrace Change [1ª ed.]. Addison Wesley, Stoughton, 1999.

[33] Ronald, J. (2012). What is extreme programming [Internet], Disponible desde http://xprogramming.com/what-is-extreme-programming/ [Acceso Junio 1, 2013].

[34] Wells, D. (1999). XP, Lessons learned: pair programming [Internet], Disponible desde http://www.extremeprogramming.org/rules/pair.html [Acceso Junio 1, 2013].

[35] Kuppuswami, S., Vivekanandan, K., Ramaswamy, P., & Rodrigues, P. Perceptions of extreme programming: an exploratory study. ACM SIGSOFT Software Engineering Notes, 28(6), 6-6, 2003.

[36] Beck, K., & Andres, C. Extreme Programming Explained: Embrace Change [2ª ed.]. Addison Wesley, Stoughton, 2004.

[37] Kruchten, P. The Rational Unified Process: An Introduction (2nd ed.). Addison-Wesley, Indianapolis, 2000.

[38] Cockburn, A. Agile software development. Addison Wesley, Reading, 2001.

[39] DSDM Consortium. DSDM Atern Handbook V2/2. Whitehorse Press, Ashford, 2008.

[40] Highsmith, J. Agile software development ecosystems. Addison Wesley, Boston, 2002.

[41] Adaptive Software Development [Internet], Bogotá, Universidad de los Andes. Disponible desde http://sistemas.uniandes.edu.co/~isis3425/dokuwiki/doku.php?id=ciclos:asd

[42] Koch, A.S. Agile software development. Evaluating the methods for your organization. Artech House, Londres, 2005.

[43] Palmer, S. R., & Felising, J. M. A practical guide to feature-driven devolpment. Pearson, Indianapolis, 2002.

[44] Ambler, S. Agile modeling: effective practices for extreme programming and the unified process. John Wiley & Sons, New York, 2002.

[45] De Luca, J. (2005). Feature driven development overview [Internet], Disponible desde http://www.nebulon.com/articles/fdd/download/fddoverview.pdf [Acceso Junio 1, 2013].

[46] Kajko-Mattsson, M. Problems in agile trenches. *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, ACM, 2008.

[47] Sousa, M. A Survey on the software maintenance process. En *Proceedings of the International Conference on Software Maintenance ICSM '98*. Washington DC: IEEE Computer Society, 1998.

[48] Cozzetti S. Anquetil, N., & de Oliveira, K. A study of the documentation essential to software maintenance. *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*, New York, ACM, 2005.

[49] Aguiar, A. Tutorial on agile documentation with Wikis. En *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, (art.41). ACM, New York, 2009.

[50] Janus, A. Towards a common agile software development model (ASDM). ACM SIGSOFT Software Engineering Notes, 37(4), 1-8, 2012.