Delfdroid: metodología ágil de desarrollo de software para dispositivos móviles.

Ernesto Avila-Domenech*, Abel Meneses Abad, Viana De la Cruz Leyva

Facultad de Ciencias Informáticas, Universidad de Granma, Cuba.

Resumen.-

Debido a las diferencias notables entre el desarrollo de software tradicional y el software para dispositivos móviles, es necesario que la metodología utilizada para guiar proyectos de desarrollo de aplicaciones para móviles tenga características no tradicionales. En el presente trabajo se propone una metodología ágil de desarrollo de software específica para dispositivos móviles. Para su descripción se ha tomado la propuesta de Alistair Cockburn, en la que se indica desglosar la metodología en diez elementos como mínimo: roles, destrezas, artefactos, actividades, valores, equipos, asignación de tareas, técnicas, herramientas y estándares. Además se realiza una evaluación comparativa con las metodologías ágiles XP y Scrum mediante el método 4-Dimensional Analytical Tool propuesto por Asif Qumer y Brian Henderson-Sellers. Los resultados resultan ser alentadores para el Centro de Desarrollo de la Facultad Regional Granma perteneciente a la Universidad de las Ciencias Informáticas y el desarrollo de aplicaciones para dispositivos móviles.

Palabras clave: Dispositivo móvil, Ingeniería de software, Métodos ágiles.

Deldroid: agile development methodology for mobile device.

Abstract.-

Because of the significant differences between traditional software development and software for mobile devices, it is necessary for mobile application projects to use a development methodology with non-traditional characteristics. This paper proposes an agile software development methodology for mobile devices. For description is taken Alistair Cockburn's proposal, which break down the methodology described in at least ten elements: roles, skills, artifacts, activities, values, teams, assignments, techniques, tools and standards. It also shows a benchmarking between XP and Scrum using 4-Dimensional Analytical Tool proposed by Asif Qumer and Brian Henderson-Sellers. The results appear to be encouraging of the Granma Development's Center of Informatics Sciences and the development of mobile's applications.

Keywords: Agile methods, Mobile device, Software engineering.

Recibido: Diciembre 2012
Aceptado: Diciembre 2013

1. Introducción.

El desarrollo de aplicaciones móviles difiere del desarrollo de software tradicional en muchos aspectos, lo que provoca que las metodologías usadas para estos entornos también difieran de las del software tradicional. Esto es porque el software móvil tiene que satisfacer una serie de requerimientos y condicionantes especiales que lo hace más complejo. Según [1] entre ellos se pueden mencionar:

1. Canal radio. Consideraciones tales como la disponibilidad, las desconexiones, la variabilidad del ancho de banda, la heterogeneidad

*Autor para correspondencia
Correo-e: eadomenech@grm.uci.cu (Ernesto Avila-Domenech)

de redes o los riesgos de seguridad han de tenerse especialmente en cuenta en este entorno de comunicaciones móviles.

2. Movilidad. Aquí influyen consideraciones como la migración de direcciones, alta latencia debido a cambio de estación base o la gestión de la información dependiente de localización. Sobre esta última, de hecho, se pueden implementar un sinfín de aplicaciones, pero la información de contexto asociada resulta muchas veces incompleta y varía frecuentemente.
3. Portabilidad. La característica portabilidad de los dispositivos terminales implica una serie de limitaciones físicas directamente relacionadas con el factor de forma de los mismos, como el tamaño de las pantallas (algo que ha variado sustancialmente con la popularización de las pantallas táctiles), o del teclado, limitando también el número de teclas y su disposición.
4. Fragmentación de la industria. La existencia de una considerable variedad de estándares, protocolos y tecnologías de red diferentes añaden complejidad al escenario del desarrollo móvil.
5. Capacidades limitadas de los terminales. Aquí se incluyen factores como la baja potencia de cálculo o gráfica, los riesgos en la integridad de datos, las interfaces de usuario poco funcionales en muchos aspectos, la baja capacidad de almacenamiento, la duración de las baterías o la dificultad para el uso de periféricos en movilidad. Factores todos que, por otro lado, están evolucionando en la dirección de la convergencia de los ultraportátiles (netbooks) con los dispositivos inteligentes (smartphones) constituyendo cada vez menos un elemento diferencial.
6. Diseño. Desde el punto de vista del desarrollo, el diseño multitarea y la interrupción de tareas es clave para el éxito de las aplicaciones de escritorio; pero la oportunidad y frecuencia de éstas es mucho mayor que en el software tradicional, debido al entorno móvil que manejan, complicándose todavía más debido a la limitación de estos dispositivos.
7. Usabilidad. Las necesidades específicas de amplios y variados grupos de usuarios, combinados con la diversidad de plataformas tecnológicas y dispositivos, hacen que el diseño para todos se convierta en un requisito que genera una complejidad creciente difícil de acotar.
8. Time-to-market. En un sector con un dinamismo propio, dentro de una industria en pleno cambio, los requisitos que se imponen en términos de tiempo de lanzamiento son muy estrictos y añaden no poca dificultad en la gestión de los procesos de desarrollo.

En la literatura se pueden encontrar algunas propuestas de metodologías para el desarrollo de aplicaciones para dispositivos móviles, entre ellas se encuentran Dynamic Channels [2], Mobile-D [3] y un modelo híbrido [4]. Si bien no son populares, tienen algunos elementos interesantes en comparación con las populares Extreme Programming (XP) [5], Scrum [6] y Feature-Driven Development (FDD) [7].

Teniendo en cuenta que entre los dispositivos móviles, el teléfono móvil es el más utilizado; y que además las aplicaciones que estos utilizan están soportadas bajo distintas plataformas, se dispuso diseñar una metodología ágil tomando algunas características de las conocidas XP y Scrum de forma tal que pudiese guiar, de forma más ajustada y ágil, proyectos donde se desarrollen aplicaciones para dispositivos móviles sin tener en cuenta la plataforma de desarrollo, ya sea iOS, Android, J2ME, etc.

2. Metodología.

Cada metodología de desarrollo de software, ya sea tradicional o ágil, posee ventajas y desventajas. Se comparte el criterio de que las metodologías tradicionales como RUP no son viables en proyectos donde la satisfacción del cliente a corto tiempo es importante; y en la que dicha satisfacción se logra entregándole una versión ejecutable en un tiempo sumamente corto aunque no posea todas las funcionalidades solicitadas.

2.1. Metodología Ágil.

Las metodologías ágiles surgen con la creación por parte de Kent Beck, tomando ideas recopiladas junto a Ward Cunningham, de XP. En 1996, Beck es llamado por Chrysler como asesor del proyecto Chrysler Comprehensive Compensation (C3) payroll system. Dada la poca calidad del sistema que se estaba desarrollando, Beck decide tirar todo el código y empezar de cero utilizando las prácticas que él había ido definiendo a lo largo del tiempo. El sistema, que administra la liquidación de aproximadamente 10.000 empleados, y contiene unas 2.000 clases y 30.000 métodos, es puesto en operación en mayo de 1997 casi respetando el calendario propuesto. Como consecuencia del éxito de dicho proyecto, Kent Beck dio origen a XP iniciando el movimiento de metodologías ágiles al que se anexarían otras metodologías surgidas mucho antes que el propio Beck fuera convocado por Chrysler [8].

Es así como las metodologías de este tipo fueron inicialmente llamadas "metodologías livianas", sin embargo, aún no contaban con una aprobación pues se le consideraba por muchos desarrolladores como meramente intuitiva. Luego, con el pasar de los años, en febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace formalmente el término "ágil"aplicado al desarrollo de software. En esta misma reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software con el objetivo de 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 [8].

Tras esta reunión se creó The Agile Alliance, 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 fue el Manifiesto Ágil, un documento que resume la filosofía "ágil".

En el Manifiesto Ágil [9] se valora:

1. Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.
2. Desarrollar software que funciona más que conseguir una buena documentación.
3. La colaboración con el cliente más que la negociación de un contrato.
4. Responder a los cambios más que seguir estrictamente un plan.

Las valoraciones anteriores han sido fundamentales para que el desarrollo ágil de software haya tenido un enorme impacto en cómo se desarrolla el software en todo el mundo.

2.2. Diseño.

Para el diseño de la metodología que se propone, a la que se ha nombrado como Delfdroid, se ha seguido la propuesta de Alistair Cockburn. Ella desglosa una metodología en 10 elementos como mínimo [10].

2.3. Roles.

Equipo del Proyecto (Team). Es el equipo del proyecto que tiene la autoridad para decidir cómo organizarse para cumplir con los objetivos de cada iteración; dichas iteraciones son nombradas Sprint al igual que en Scrum.

Jefe de Proyecto (Project Manager). Es un rol de administración que debe asegurar que el proyecto se está llevando a cabo de acuerdo con la metodología de desarrollo empleada y que todo funciona según lo planeado.

Jefe de Producto (Product Management). Es el responsable de tomar las decisiones finales, acerca de estándares y convenciones a seguir durante el desarrollo de un producto determinado. Participa en la selección de objetivos y requerimientos.

Visionario (Visionary). Es la persona que luego de expresar una idea y solicitar que sea desarrollada es aprobada por el Jefe de Proyecto junto con otros especialistas. Tendrá las ideas iniciales del producto que se desea desarrollar.

ADD (Analyst-Designer-Developer). Encargado de realizar el análisis, diseño e implementación correspondiente al producto que se desarrolla.

Arquitecto (Architect). Se vincula directamente con los ADD debido a que su trabajo tiene que ver con la estructura y el diseño en general de las aplicaciones.

Encargado de Pruebas (Tester). Es el encargado de ayudar al Visionario 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.

Consultor (Consultant). Es un miembro externo del equipo con un conocimiento específico en algún tema necesario para el proyecto, en el que puedan sugerir problemas, además aporta ideas y experiencias para el beneficio del sistema en desarrollo. Esta es una especialización menos activa, quien la ejecuta funciona por un corto período de tiempo. Está más orientada a roles de desarrollo y sus ejecutores pueden trabajar en otros roles (en otro equipo) durante el desarrollo del software.

2.4. Destrezas.

Jefe de Proyecto. Debe poseer aptitudes en comunicación para poder capturar y entender la visión generada por cualquier tipo de persona. Debe saber escuchar y ser optimista en todo momento.

Jefe de Producto. Debe poseer aptitudes en comunicación y ser exigente con el equipo y con sigo mismo.

Arquitecto. Debe tener buena formación técnica, contar con experiencia en las herramientas y técnicas utilizadas. Además debe tener aptitudes comunicativas para que la arquitectura llegue a todos los miembros del equipo. También debe ser perseverante en conseguir los hitos técnicos planteados para asegurar el progreso de la construcción.

Visionario. Debe poseer aptitudes en comunicación para poder explicar la visión generada. Además debe ser una persona decidida.

ADD. Debe tener amplio conocimiento de técnicas de levantamiento de funcionalidades. Además debe tener amplio conocimiento de las herramientas de desarrollo, del lenguaje de programación utilizado y de los aspectos técnicos involucrados.

Encargado de Pruebas. Debe tener amplio conocimiento de técnicas de pruebas, debe conocer a fondo la aplicación que probará. También debe tener conocimientos de programación para trabajar con las pruebas automatizadas.

Consultor. Debe tener amplio conocimiento y una elevada preparación en los temas que abordará con el equipo de desarrollo.

2.4.1. Artefactos.

Ficha técnica. Documento donde se escribirá claramente lo que se desea con el producto, así como su posible aceptación o impacto en los usuarios finales. Tendrá un análisis de factibilidad, además contendrá las funcionalidades más importantes del sistema a ser construido y los requerimientos suplementarios que se detecten de forma temprana.

Lista reserva de producto (LRP). Documento que posee una lista priorizada que define el trabajo que se va a realizar en el proyecto. Esta lista puede crecer y modificarse a medida que se obtiene más conocimiento acerca del producto (con la restricción de que solo puede cambiarse entre Sprint).

Plan de capacitación. Recoge la planificación del entrenamiento y la preparación necesaria del equipo de desarrollo para lograr los objetivos del proyecto.

Historias de Usuario (HU). Documento donde se especifican las funciones del producto. Servirán para guiar los demás artefactos y para la construcción de los componentes del producto. Son escritas en lenguaje natural, no excediendo su tamaño de unas pocas líneas de texto.

Tarjetas CRC. Las clases pueden ser descubiertas usando tarjetas de Colaboración – Responsabilidad – Clases (CRC por sus siglas en inglés). Fueron introducidas por Ward Cunningham y Kent

Beck en 1989. Una tarjeta CRC es una tarjeta que muestra: el nombre de la clase y su descripción, la responsabilidad de la clase, conocimiento interno de la clase, servicios brindados por la clase y los colaboradores para las responsabilidades. Un colaborador es una clase cuyos servicios son necesarios para una responsabilidad.

Plan de release o Plan de liberación. Mediante este documento se realiza una planificación del proyecto. Especifica qué Historias de Usuarios deben ser implementadas para cada uno de los release (liberación) del sistema. Cada release es conformado por un número de iteraciones (Sprints).

Arquitectura de software. Documento donde se especifican los aspectos técnicos de la solución propuesta por el equipo de desarrollo. Sirve como medio de comunicación entre el arquitecto y el equipo en relación a cómo deberán ser implementadas las HU.

Informe de investigación. Documentos donde se describen las investigaciones realizadas, así como resultados obtenidos.

Glosario de términos. Documento que recoge los términos que se relacionan con el producto y la metodología utilizada que pueden causar dudas al equipo. Es necesaria su actualización continua para un mejor entendimiento de todo el proceso de desarrollo de software.

Minuta de reunión. Documento que recoge los acuerdos tomados así como la asistencia a la reunión.

Plan de pruebas. Contiene una lista de pruebas a realizar con su fecha de realización correspondiente así como las personas involucradas en las mismas. Divide las pruebas en pruebas de aceptación (pruebas que avala el Visionario) y pruebas de mercado (pruebas realizadas por usuarios finales).

Plan de aseguramiento de la calidad. Describe cómo se asegurará la calidad de los productos de la institución. Se detallan de forma general los artefactos, herramientas y procesos a través de los cuales se planifica lograr dicho propósito.

Casos de prueba. Documento que contiene las especificaciones de las pruebas a realizar para la comprobación y validación de las funcionalidades del producto y de esta forma saber si está apto para ser liberado.

Gestión de cambios. Documento donde se guardan los cambios realizados en el producto una vez terminado el producto y se pasa a la fase donde se mantiene, dándole soporte a los usuarios. Recoge los datos para identificar el nombre de la HU donde se deriva, quién realiza el cambio y el lugar donde ocurre la actualización en el código.

Plan mitigación de riesgos. Documento donde se recoge cada una de las acciones a ejecutar para erradicar o mitigar uno los riesgos del proyecto.

Relación de herramientas y componentes. documento que recoge todas las herramientas y componentes utilizadas en el proyecto.

Manual de usuario. Documento que sirve de ayuda a los usuarios para la interacción con la aplicación.

Plantilla para registro. Documento legal para el registro del producto.

2.4.2. Actividades.

Delfdroid posee tres fases llamadas Inicio, Elaboración-Construcción y Transición. Las mismas indican el énfasis propuesto por Barry Boehm [11] en relación a la división en dos perspectivas de desarrollo. La primer perspectiva ocurre en las iteraciones iniciales del proyecto, cuando se analiza la factibilidad de la ejecución del mismo terminando con la obtención de la arquitectura y los riesgos más críticos mitigados. La segunda perspectiva ocurre en fases iterativas de construcción en donde el énfasis está en el código fuente generado.

Durante la fase de Inicio el Visionario deberá plasmar los objetivos y elementos que persigue la aplicación, comienza a realizar un estudio de aplicaciones similares existentes, así como de los posibles usuarios. Además se realiza un diseño del escenario (diagrama o descripción que refleja la manera que se venderá o divulgará la aplicación). Una vez terminadas las actividades anteriores se realiza un estudio de factibilidad económica de la aplicación propuesta a desarrollar teniendo en cuenta otros beneficios no necesariamente económicos.

En la siguiente fase Elaboración-Construcción se describen las funcionalidades a desarrollar mediante HU. Luego se diseñan, haciendo uso de las tarjetas CRC y por último son implementadas mediantes Sprints empleando la práctica Test Driven Developmen (TDD) descrita posteriormente.

Durante la fase Transición ya deberán existir varias versiones Betas en el mercado con sus respectivos beneficios; no obstante, la experiencia dice que existirán inconvenientes o funcionalidades con resultados no esperados en el accionar de los usuarios, por lo que habrá que realizar algunos cambios. En caso de llegar a una versión estable y con altos niveles de aceptación se procede a registrarlo y se firma el acta de aceptación si procede.

Además de las fases se proponen cuatro flujos de trabajo (Concepción, Planificación-Definición, Implementación y Comercialización), a continuación se describen:

Concepción. Con el surgimiento de una idea por parte de cualquier persona y su evidente deseo de que sea desarrollada, dicha persona toma el rol de Visionario. Se hará un primer bosquejo de las funcionalidades de la aplicación, y la definición de restricciones y cuestiones no funcionales. Todas las tareas realizadas deberán quedar plasmada en un documento llamado "Ficha Técnica", perteneciente al expediente del proyecto.

Planificación-Definición. Se fijan las expectativas y se realiza el aseguramiento del proyecto. Se seleccionan las personas necesarias así como los materiales a utilizar. Luego se realizaría la primera reunión del proyecto donde estaría madurada la idea global de la aplicación a construir y el negocio en que está embebida.

Tomando ideas de FDD una de las actividades a realizar consiste en desarrollar un modelo global, que sugiere un cierto paralelismo con la construcción de la arquitectura del software. Es por ello que surge el artefacto "Arquitectura de Software".

Otros documentos del expediente del proyecto que comienzan a ser utilizados son Plan de capacitación, Relación de herramientas y componentes, Lista de reserva del producto, HU, Plan mitigación de riesgos y Plan de release. Pudiendo los dos últimos ser gestionados desde la plataforma de gestión de proyectos empleada.

En este flujo se definen las HU a realizar en cada uno de los Sprints mediante una planificación. Tomando idea de Scrum al comienzo de cada Sprint se realiza una reunión llamada "Reunión de Planificación del Sprint"dividida en dos partes. En la primera, denominada "Planificación del Sprint Parte Uno"hay una comunicación entre el Visionario y el Equipo de Proyecto con la facilitación del Jefe de Producto para realizar una estimación del esfuerzo requerido en la implementación de las HU. Esta parte se centra en entender qué quiere el Visionario. Según las reglas de Scrum, al final de esta parte el Visionario se puede marchar aunque debe estar disponible (por ejemplo, por teléfono) durante la siguiente reunión. Sin embargo, es bienvenido a estar en la misma.

La segunda y última parte llamada "Planificación del Sprint Parte Dos"se centra en la planificación detallada de tareas para saber cómo implementar los elementos que el equipo decide hacer. El Equipo de Proyecto selecciona los elementos de la Lista de reserva del producto a las que se comprometen que estará al final del Sprint, comenzando por las de mayor prioridad según el Visionario. Esto es una práctica clave en Scrum y por supuesto en la metodología que se propone: el equipo decide a cuanto trabajo se compromete en vez de serles asignado por el Visionario o Dueño de Producto (en Scrum). Esto hace que el compromiso sea más fiable porque el equipo lo está haciendo basado en su propio análisis y

planificación en vez de que venga "hecho"por otros. Aunque el Visionario no tiene control sobre la cantidad de trabajo que el equipo se compromete a hacer, sabe que los elementos que el equipo elija son de los de mayor prioridad.

Implementación. Cuenta con tres tareas fundamentales: generación de código, prueba y optimización del software.

Luego de la "Reunión de Planificación del Sprint"son diseñadas las Historias de Usuario mediante tarjetas CRC para luego ser implementadas mediante un desarrollo orientado a pruebas (TDD) dando como resultado productos ALFAs. Cuando al finalizar un Sprint se considera que el producto cumple con determinadas funcionalidades que hacen de él un producto funcional deja de ser ALFA y pasa a ser BETA, por lo que estará listo para salir a un mercado de pruebas con el objetivo de que el equipo de desarrollo pueda retroalimentarse rápidamente.

En el caso que un producto BETA alcance niveles altos de aceptación y tenga una aceptable calidad se convertirá en un producto estable y por tanto comenzará el proceso de registro.

Comercialización. El producto sale al mercado y comienza una interacción constante con los usuarios apareciendo criterios tanto negativos como positivos. Estos elementos a los que se les pueden añadir nuevas ideas salidas de los usuarios, proporcionan funcionalidades mejoradas y en otros casos funcionalidades nuevas. Además obligan a un cambio que va asociado a la corrección de errores y a las adaptaciones requeridas a medida que va evolucionando el entorno del software. Los tipos de cambios pueden ser: Corrección, Adaptación, Mejora y Prevención.

2.4.3. Valores.

Como metodología ágil posee los valores de la Alianza Ágil:

1. Individuos e interacciones, más que procesos y herramientas.
2. Software operante más que documentaciones completas.
3. Respuesta al cambio más que apegarse a una rigurosa planificación.

Otros valores de la metodología:

1. Creatividad: el elemento más importante que persigue Delfdroid es lograr un alto grado de creatividad en los integrantes del equipo para que así puedan surgir novedosas aplicaciones.
2. Comunicación: desarrolla la comunicación entre sus integrantes partiendo de un respeto mutuo y una alta modestia.
3. Humildad: todas las ideas son importantes, todos opinan y proponen.

2.4.4. Equipos.

Los equipos son formados con un pequeño número de personas, entre 2 y 8. Se distribuyen equitativamente, dentro de lo posible, a los que más experiencia tengan en el lenguaje de programación utilizado en los distintos equipos.

Estos equipos se dividen en parejas trabajando cada una de ellas en un ordenador implementando las clases diseñadas mediante las tarjetas CRC ayudándose uno con el otro e intercambiando ideas.

En ocasiones, cuando se está madurando una idea, método, algoritmo, etc., no es necesario estar frente de un ordenador, por lo que se recomienda reunirse cómodamente en un mesa disponible y debatir allí sobre el tema en cuestión.

2.4.5. Asignación de Tareas.

La asignación de tareas se realiza a través del Redmine, herramienta para la administración del proyecto. Cuando el proyecto es aprobado oficialmente una de las acciones a realizar de inmediato es la creación en el Redmine del proyecto aprobado. Una vez creado el proyecto he identificado el Visionario, Jefe de Producto y Jefe de Proyecto se asignan roles al equipo de desarrollo. Cada rol en el Redmine posee privilegios específicos.

2.5. Técnicas.

Sprint de 15 días.

Los Sprints cortos están bien. Permiten a la compañía ser "ágil", es decir, cambiar de dirección frecuentemente [6].

En el desarrollo de aplicaciones para dispositivos móviles uno de los aspectos más importante es el time-to-maker por lo que se propone realizar Sprints de solo 15 días, logrando así una rápida salida al mercado de una versión funcional aunque no cuente con la totalidad de las funcionalidades concebidas.

"Descanso" entre Sprints.

Se intenta introducir algo de descanso antes de empezar un nuevo Sprint (más específicamente, en el periodo después de terminar la retrospectiva del Sprint y antes de la siguiente reunión de planificación de Sprint). No siempre se consigue [6].

Como mínimo, se intenta que la retrospectiva y la subsiguiente reunión de planificación de Sprint no ocurran el mismo día. Todo el mundo debería tener al menos una buena noche de sueño sin Sprint antes de comenzar el siguiente Sprint [6].

Una forma de hacer esto son los "días de laboratorio"(o como se quiera llamar). Es decir, días en los que a los desarrolladores se les permite hacer esencialmente cualquier cosa que deseen (inspirado en Google). Por ejemplo, leer sobre las últimas herramientas y API's, estudiar para una certificación, discutir asuntos técnicos con colegas, programar un proyecto personal como hobby, etc. [6].

Diseño mediante tarjetas CRC.

Las tarjetas CRC (Clase - Responsabilidad-Colaborador) son una herramienta con la cual a veces se asignan las responsabilidades y se indica una colaboración con otros objetos. Fueron inventadas por Kent Beck y Ward Cunningham, para que los diseñadores de objetos piensen en términos más abstractos sobre la asignación de responsabilidades y de las colaboraciones. Las tarjetas CRC son tarjetas de índices, una por cada clase, sobre las cuales se abrevian las responsabilidades de la clase y se anota una lista de los objetos con los que colaboran para desempeñarlas. Suelen elaborarse en una sesión de grupos pequeños donde los participantes representan papeles de las diversas clases. Cada grupo tiene las tarjetas CRC correspondientes a las clases cuyo papel está desempeñando. La representación de papeles hace divertido aprender a pensar en función de objetos y de responsabilidades [12].

Desarrollo orientado a pruebas (TDD, Test Driven Development).

Test Driven Development es una técnica para aumentar la agilidad en el desarrollo del proyecto. La literatura empírica existente sobre TDD ha demostrado una mayor productividad y un código más robusto, entre otros beneficios importantes [13].

El objetivo de TDD es ofrecer beneficios de agilidad a través de un conjunto de pruebas de unidad automatizadas.

Primeramente es implementada una prueba, luego se le añade el código para pasar esta prueba que se ha escrito. Cuando la prueba pasa, el código es refactorizado. La refactorización según Fowler [14] es el proceso de realizar cambios al código sin cambiar su comportamiento externo, es decir, el código se ve alterado por los efectos de comentar, hacerlo más sencillo, o mejorando algún aspecto de calidad. Este ciclo se repite hasta que toda la funcionalidad esté implementada.

Importancia al factor Dedicación.

Sería un error no considerar el tiempo que se deberá dedicar a tareas de otra índole y que se escapan del proyecto

$$\mathrm{DH}_D F_D = \mathrm{V}_E$$

donde:
DH_D: Días-Hombre disponibles,
F_D : factor de dedicación y
V_E: velocidad estimada.

El factor de dedicación es una estimación de cómo de centrado va a estar el equipo. Un factor de dedicación bajo puede significar que el equipo espera encontrar muchas distracciones e impedimentos o que considera que sus propias estimaciones son optimistas [6].

Es por ello que se propone incluir en el artefacto Minuta de Reunión, correspondiente a la segunda reunión de cada Sprint, una tabla que contenga una celda identificando al Sprint a que pertenece, los nombres de cada miembro del equipo de desarrollo de software así como los días disponibles, horas

Número de Sprint		# del Sprint	
Días laborables durante el Sprint		# de días laborables durante el Sprint	
Miembro del equipo	Días disponibles durante el Sprint	Horas disponibles por día	Total de horas disponibles
Nombre 1	#	#	#
Nombre 2	#	#	#
⋮	#	#	#
Nombre n	#	#	#

Figura 1: Control del factor dedicación.

disponibles en cada día y total de horas disponibles correspondientes a cada uno de los integrantes como se muestra en la Figura 1.

2.5.1. Herramientas.

Bazaar.

Bazaar, es una herramienta para el control de versiones y tiene la característica de soportar distintas formas de trabajo, en vez de obligar a que el equipo de desarrollo se adapte a una única forma posible. Las formas de trabajo se pueden cambiar, mezclar o ajustar como se necesite. Por ejemplo, un equipo puede decidir usar una forma tradicional "centralizada", pero complementarla con el modo en "pareja", es decir, dos desarrolladores intercambian directamente sus cambios cuando trabajan juntos.

Las formas de trabajo son muy independientes de la forma de almacenamiento. Si se quiere asegurar que todas las ramas se guarden en un repositorio central, hay formas de hacerlo, por ejemplo, llevando los cambios a un servidor central del que se hacen backups. En muchos equipos que utilizan herramientas de control de versiones, los desarrolladores suben el código al repositorio simplemente como backup o para etiquetarlo. Los sistemas de control de versiones distribuidos permiten hacer lo mismo, pero sin subir los cambios a la rama principal antes de lo debido, este es el caso de Bazaar, un software libre patrocinado por Canonical.

Redmine.

Redmine es una aplicación web para la administración flexible de proyectos. Está escrito en Ruby. Además de que es multiplataforma soporta disímiles bases de datos. Es de código abierto siendo GNU Genral Public License v2 (GPL) su licencia.

Se confirma que posee características como:

1. Soporta múltiple proyectos.
2. Control de acceso flexible basado en roles.
3. Posee notificaciones por correo electrónico.
4. Se integra con distintos controladores de versiones tales como SVN, CVS, Git, Mercurial, Bazaar y Darcs.
5. Soporta múltiples autentificaciones LDAP (en español Protocolo Ligero de Acceso a Directorios, el cual permite el acceso a un servicio de directorio ordenado y distribuido para buscar información en un entorno de red).
6. Soporta una gran variedad de lenguajes.
7. Soporta múltiples bases de datos como MySQL, PostgreSQL o SQLite.

2.5.2. Estándares.

La mayoría de los programadores tienen su propio y distintivo estilo de programación. Pequeños detalles como la forma en la que tratan las excepciones, como comentan el código, cuando devuelven un valor null, etc. En algunos casos esta diferencia no importa, pero en otros puede conducir a una severa inconsistencia del diseño del sistema y a un código difícil de leer. Un estándar de código ayuda a que esto no ocurra, siempre que se concentre en lo que realmente importa [6].

2.6. Prácticas.

Las prácticas que intervienen en la metodología de desarrollo propuesta son las siguientes:

1. Desarrollo iterativo e incremental mediante Sprint cortos: pequeñas mejoras, unas tras otras. Se integra y se prueba todo el sistema varias veces en el día.
2. Programación en pares: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se le otorga mayor importancia a la calidad del código escrito de esta manera el código es revisado y discutido mientras se escribe- que a la posible pérdida de productividad.

3. Pruebas: los desarrolladores escriben pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión, el código escrito debe ejecutarse sin fallo para que el desarrollo continúe. El Visionario junto con el Encargado de Pruebas deben escribir pruebas que demuestren que las funcionalidades han sido desarrolladas correctamente.
4. 40 horas semanales: Trabajar en el proyecto 40 horas semanales y solo 40 horas, a razón de 8 horas diarias. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente está ocurriendo un problema que debe corregirse.
5. Utilización de metáforas: una metáfora es una historia compartida que describe cómo debería funcionar el sistema (conjunto de nombres que actúen o como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema).
6. Rápida retroalimentación (feedback): Sacar versiones al mercado interno (versiones BETAs) para retroalimentarse rápidamente de los usuarios.
7. Diseño simple: realizar diseños simples para evolucionar constantemente adicionando necesarias flexibilidades y eliminando innecesarias complejidades.
8. Refactorización: los desarrolladores reestructuran el sistema eliminando código duplicado, simplificándolo o adicionándole flexibilidad.
9. Participación activa de todos los miembros del proyecto: todos los miembros del proyecto pueden sugerir funcionalidades nuevas.
10. Creación de varios modelos en paralelo: puede existir un punto donde se debe escoger un camino a tomar, sin embargo el equipo se puede dividir y tomar ambos.
11. Reutilización constante: no re-inventar la rueda. Dejar a un lado nuestro orgullo y reutilizar lo ya desarrollado. Vale realizar mejoras.
12. Estándar de codificación: utilización por parte del equipo de desarrollo de un estándar de programación.
13. Reunión para controlar el Sprint terminado y planificar el siguiente: para el siguiente Sprint se realiza un juego de planeación teniendo en cuenta la estimación del esfuerzo.
14. Reportando progreso: el progreso se reporta sobre la base de trabajo completado a todos los niveles organizativos necesarios.
15. Cliente en la puerta: los clientes no están fuera ni dentro del proyecto. Si bien no se cuenta con un cliente específico en el equipo de desarrollo, se poseen varios usuarios bien cerca; los cuales pueden ser invitados y tomar el papel de cliente.

2.7. Evaluación comparativa con XP y Scrum de la caracterización de la agilidad mediante el método 4-Dimensional Analytical Tool ($4 - DAT$).

Un método de desarrollo de software se dice que es un método ágil de desarrollo de software cuando se centra en las personas, es orientado a la comunicación, flexible (listo para su adaptación a la espera de un cambio inesperado en cualquier momento), rápida (estimula el rápido e iterativo desarrollo del producto en versiones pequeñas), delgado (se centra en acortar los plazos y costes y en la mejora de la calidad), sensible (reacciona adecuadamente a los cambios esperados e inesperados), y aprende (centrado en la mejora durante y después del desarrollo del producto) [15].

El método 4-DAT facilita el examen de los métodos ágiles desde cuatro perspectivas o dimensiones: alcance de la metodología, caracterización de la agilidad, valores ágiles (Manifiesto Ágil) y caracterización del proceso de software. Aunque en la actualidad hay cuatro dimensiones evaluadas en el enfoque de 4-DAT, es ampliable en el hecho de que se pueden agregar o quitar dimensiones o elementos de las dimensiones, si se considera necesario en el futuro [16].

La herramienta está diseñada para comparar y analizar los métodos ágiles. Un informe que se genera con la ayuda de 4-DAT se puede utilizar para la toma de decisiones con respecto a la adopción de un método ágil apropiado [17].

El grado de agilidad (DA) depende de los términos flexibilidad (FY), velocidad (SD), eficiencia

(LS), aprendizaje (LG) y adaptabilidad (RS).

Tabla 1: Grado de Agilidad de Delfdroid.

	Características de agilidad					
Delfdroid	FY	SD	LS	LG	RS	Total
(i) Fases						
Inicio	1	1	0	0	1	3
Elaboración-Construcción	1	1	0	1	1	4
Transición	1	1	0	1	1	4
Total	3	3	0	2	3	11
Grado de Agilidad	3//3	3//3	0//3	2//3	3//3	11//(3*5)
(ii) Prácticas						
Desarrollo iterativo e incremental mediante Sprint cortos	1	1	1	1	1	5
Programación en pares	1	0	0	1	1	3
Pruebas	1	1	0	1	1	4
40 horas semanales	0	0	0	1	0	1
Utilización de Metáforas	0	1	1	0	0	2
Rápida retroalimentación	1	1	1	1	1	5
Diseño simple	1	1	1	1	1	5
Refactorización	1	1	1	1	1	5
Participación activa de todos los miembros del proyecto	1	1	1	1	1	5
Creación de varios modelos en paralelo	1	1	0	1	1	4
Reutilización constante	0	1	1	0	1	3
Estándar de codificación	1	1	1	1	1	5
Reunión para controlar el Sprint terminado y planificar	1	1	0	1	1	4
Reportando progreso	1	1	1	1	1	5
Cliente en la puerta	1	0	1	1	1	4
Total	12	12	10	13	13	60
Grado de Agilidad (DA)	12//15	12//15	10//15	13//15	13//15	60//(15*5)

La operación a//b se lee a de b posibles

Tabla 2: Grado de agilidad de XP, Scrum y Delfdroid.

	XP	Scrum	Delfdroid
Fases	0.70	0.60	0.73
Prácticas	0.73	0.80	0.80
Promedio (Fases y Prácticas)	0.72	0.70	0.77

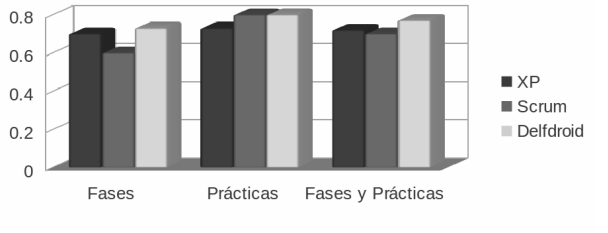

Figura 2: Gráfica representativa del grado de agilidad de XP, Scrum y Delfdroid.

Delfdroid posee varias prácticas que poseen XP o Scrum, pues de cierta manera es una mezcla de ambas; por tal motivo las prácticas que coinciden con algunas de las prácticas de XP o Scrum han sido evaluadas de igual manera que fueron evaluadas por Qumer y Henderson-Sellers en [18] (Ver Tabla 1, Tabla 2 y Figura 2).

3. ANÁLISIS DE LOS RESULTADOS

Delfdroid al ser un híbrido de las metodologías ágiles XP y Scrum, posee prácticas que estas dos últimas también poseen, o al menos una de ellas. Las mismas han sido incluidas de tal manera que la metodología se adapte de una mejor manera al desarrollo de aplicaciones para dispositivos móviles. Además no se ha descuidado en lo absoluto la agilidad correspondiente a cada una de ellas.

Si se analizan las fases se observa que Delfdroid posee solo tres, las mismas poseen tal agilidad que en su conjunto se obtienen resultados ligeramente mejores que XP y en comparación con Scrum la diferencia es notable, para bien.

En cuanto a las prácticas, si bien la mayoría se han escogido XP y Scrum, brindan junto con las nativas de Delfdroid resultados semejantes a las de Scrum. Si se comparan con las de XP se nota una diferencia significante.

4. Conclusiones.

Luego de describir Delfdroid se puede concluir que se ha propuesto una metodología con tres fases y cuatro flujos de trabajo principales. Por sus características se considera ágil y adaptable a entornos de desarrollo de aplicaciones para dispositivos móviles. Además, agrupa muchas de las llamadas “buenas prácticas de desarrollo de software”empleadas en las más importantes y utilizadas metodologías ágiles encontradas en la literatura; y adiciona otras como el cliente en la puerta. Delfdroid está enfocada a los usuarios; la práctica cliente en la puerta es una metáfora que indica que no existe cliente dentro ni fuera del equipo del proyecto. Como se ha explicado, se toman en cortos periodos de tiempo, a usuarios finales y se les solicita que comenten, critiquen y propongan cambios o nuevas funcionalidades.

Referencias

[1] Blanco P, Camarero J, Fumero A, Werterski A and Rodríguez P. (2009): "Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone". Universidad Politécnica de Madrid.

[2] Alfonso A, Regateiro S and Silva M. (1998) "Dynamic Channels: a New Development Methodology for Mobile Computing Applications". Universidad de Lisboa, pp. 1–22.

[3] Abrahamsson P, Hanhineva A, Hulkko H, Ihme H, Jäälinoja J, Korkala M, Koskela J, Kyllönen P and Salo O. (2004). "Mobile-D: an agile approach for mobile application development". pp. 174–175.

[4] Rahimian V and Ramsin R. (2008) "Designing an agile methodology for mobile software development: A hybrid method engineering approach". Second International Conference on Research Challenges in Information Science. pp. 337–342.

[5] Beck K. (1999). "Extreme Programming Explained". p. 224.

[6] Kniberg H. (2007): "Scrum y XP desde las trincheras".

[7] Anderson D. (2004). "Feature-Driven Development".

[8] Calderón A and Dámaris S. (2007): "Metodologías Ágiles".

[9] Principios del Manifiesto Ágil. In: [online]. [Consultado el 13 de febrero de 2012]. Disponible en: `http://www.agilemanifesto.org/iso/es/principles.html`.

[10] Cockburn A. (1998): "Surviving object oriented projects". Addison-Wesley Object Technology Series.

[11] Boehm B and Turner R. (2003). "Balancing Agility and Discipline: A Guide for the Perplexed". Addison Wesley, p. 304.

[12] Larman C. (1999): "UML y Patrones: Introducción al análisis y diseño orientado a objetos". Prentice Hall. ISBN 0-13-748880-7.

[13] Abrahamsson P, Hanhineva A and Jaalinoja J. (2005): "Improving Business Agility Through Technical Solutions: A Case Study on Test-Driven Development in Mobile Software Development".

[14] Fowler, M. (1999). "Refactoring: Improving the design of existing code". Boston: Addison Wesley.

[15] Qumer A and Henderson-Sellers B. (2008): "An evaluation of the degree of agility in six agile methods and its applicability for method engineering". Elsevier Inc.

[16] Qumer A and Henderson-Sellers B. (2008): "A framework to support the evaluation, adoption and improvement of agile methods in practice", Journal of Systems and software, 18 (11), 1988–1919. [DOI 10.1016/j.jss.2007.12.806].

[17] Qumer A and Henderson-Sellers B. (2006): "Measuring Agility and Adoptability of Agile Methods: A 4-Dimensional Analytical Tool". IADIS International Conference Applied Computing.

[18] Qumer A and Henderson-Sellers B. (2006): "Comparative Evaluation of XP and SCRUM using the 4D Analytical Tool (4-DAT)". European and Mediterranean Conference on Information Systems (EMCIS). Alicante, Spain.