Ingeniare. Revista Chilena de Ingeniería
ISSN: 0718-3291
facing@uta.cl
Universidad de Tarapacá
Chile

Leiva Mundaca, Ignacio; Villalobos Abarca, Marco
Método ágil híbrido para desarrollar software en dispositivos móviles
Ingeniare. Revista Chilena de Ingeniería, vol. 23, núm. 3, 2015, pp. 473-488
Universidad de Tarapacá
Arica, Chile

Método ágil híbrido para desarrollar *software* en dispositivos móviles

Hybrid method for agile software develop mobile devices

Ignacio Leiva Mundaca[1] Marco Villalobos Abarca[1]

Recibido 29 de julio de 2014, aceptado 20 de abril de 2015
Received: July 29, 2014 Accepted: April 20, 2015

RESUMEN

Este trabajo presenta un nuevo método ágil híbrido para desarrollar *software* en dispositivos móviles, la evaluación de cuán ágil es, si es o no adecuada para el desarrollo de aplicaciones en dispositivos móviles y su aplicación en un caso real para evaluar sus bondades. El nuevo método se derivó mediante un proceso de cinco etapas, que consistieron en un análisis del *software* para dispositivos móviles y los métodos ágiles existentes, el diseño de un enfoque evaluativo, la aplicación del enfoque evaluativo, el desarrollo del método híbrido ágil para *software* en dispositivos móviles y la aplicación del método a un caso real.

Se describen como parte del nuevo método los recursos, esquemas y artefactos que emplea, debido a que pueden ser de gran utilidad tanto para los desarrolladores como para los jefes de proyecto de este tipo de *software*. Las evaluaciones teóricas y prácticas del nuevo método indican que posee propiedades que son exigibles a los métodos ágiles. Incorpora las fortalezas y mejoras descubiertas de métodos existentes. Así mismo, el desarrollo de un *software* para dispositivo móvil real demostró su aplicabilidad y buen rendimiento.

Palabras clave: Ingeniería de *software*, métodos ágiles, método híbrido.

ABSTRACT

This research work presents a novel hybrid agile method for software development on mobile devices. The evaluation of several aspects is presented to show the advantages of this method, including level of agility the method, if the method is suitable or not to develop applications on mobile devices and its applicability on a real example. This new method was created after a process of five steps, which consisted on the analysis of current software for mobile devices and existent agile development methods, the design of an evaluation methodology for agile methods, the application of the evaluation method, the development of a hybrid agile development method and the use of the method in a real example.

Components of the novel method, such as resources, schemas and artefacts used are described, since those elements can be useful for developers and project managers of mobile devices' software. The theoretical and practical evaluation of this new method indicates it possess the necessary characteristics to be consider an agile methodology. The method incorporates the strengths and improvements discovered in current agile development methods. Also, the application on the real example of development for mobile devices showed its applicability and good performance.

Keywords: Software engineering, agile methods, hybrid method.

[1] Área de Ingeniería en Computación e Informática, Escuela Universitaria de Ingeniería Industrial, Informática y de Sistemas. Universidad de Tarapacá Arica, Chile. E-mail: mvillalo@uta.cl; nacholeiva1@gmail.com

INTRODUCCIÓN

Debido a la rapidez con que la industria del desarrollo de *software* para dispositivos móviles genera productos, resulta bastante complicado y costoso seguir utilizando los métodos de desarrollo convencionales. Así, una parte no menos importante de esta industria ha optado por un patrón de desarrollo "no convencional", dirigiendo la mayoría de los esfuerzos del equipo a la entrega constante de prototipos funcionales, dando prioridad a la buena comunicación entre los miembros del mismo más que a la documentación, a la misma vez que a la interacción constante con el cliente. El desarrollo de la aplicación se agiliza considerablemente, por cuanto se reducen todos los procesos que no estén directamente relacionados con la entrega de un producto o prototipo funcional. Además, conlleva un alto grado de satisfacción del cliente, el que es un miembro activo del equipo de desarrollo. Métodos de naturaleza no convencionales son conocidos como Métodos Ágiles.

Con el advenimiento de los teléfonos inteligentes el mercado de las SmartPhone Apps ha tenido un crecimiento sostenido desde el 2005 a la fecha. En el caso de Chile, el segundo semestre de 2013 la venta de Smartphones superó a la de los teléfonos celulares tradicionales, alcanzando el 52% del mercado, en un país donde hay más celulares que personas [1]. Aplicaciones de todo tipo, desde sencillos blocks de notas hasta complejos videojuegos, han tenido que entrar a la espiral del mercado, en donde el factor tiempo es fundamental y puede resultar poco eficiente para los desarrolladores el usar un método tradicional que invierte más recursos en la administración de procesos más que en la entrega de resultados.

Tomando en cuenta lo anterior, se implementó un trabajo de investigación aplicada, tendiente a explorar el funcionamiento de diferentes métodos ágiles existentes, evaluando cada una de ellas en función de las necesidades que la industria de desarrollo de *software* para dispositivos móviles requiere. A partir de esta evaluación se propuso un método híbrido tomando en cuenta los factores mejor evaluados de cada uno de los métodos en cuestión. Finalmente se puso en práctica el nuevo método mediante el desarrollo de un *software* para dispositivos móviles. El uso demostró su aplicabilidad y buen rendimiento.

El presente trabajo se encuentra estructurado en la presente introducción, posteriormente se discute el marco teórico, desarrollando la diferencia entre métodos tradicionales y ágiles, el *software* para dispositivos móviles y el trabajo relacionado. Continúa con la descripción del diseño conceptual del proceso para construir el método. Luego se describe el método propiamente tal y su posterior evaluación. Finalmente se entregan las conclusiones relevantes y las referencias bibliográficas.

MARCO TEÓRICO

Métodos para desarrollar *software*

Los métodos para desarrollar *software* ya tienen una larga data. Desde los años 60 en adelante aparecen varias propuestas que hoy se clasifican en tradicionales, ágiles e híbridos.

Métodos tradicionales

Los métodos tradicionales nacen a principios de la década de los 60 en contraposición a lo que representaba un desarrollo *ad hoc* o artesanal. 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. Demostraron ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en muchos otros. Una posible mejora sería 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.

Métodos ágiles

Los métodos ágiles nacen a principios de la década de los 90 en contraposición a lo que representaban los métodos tradicionales.

En la Tabla 1 se muestran algunas de las metodologías nacidas en las décadas de 1990 y 2000. Esta explosión de metodologías llevó a que, en febrero del 2001, tras una reunión celebrada en Utah, USA, se acuñara formalmente el término "ágil" aplicado al desarrollo de *software*. En esta misma reunión participó 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 pudieran surgir a lo largo del proyecto.

Tabla 1. Métodos ágiles, décadas de 1990-2000.

Método	Acrónimo	Autor (es)
Adaptive software development	ASD	Highsmith 2000
Agile modeling	AM	Ambler 2002
Cristal methods	CM	Cockbum 1998
Agile RUP	dX	Booch, Martin, Newkirk 1998
Dynamic solutions delivery model	DSDM	Stapleton 1997
eXtreme Programming	XP	Beck 1999
Feature-driven development	FDD	Charette 2001, Mary y Tom Poppendieck
Rapid development	RAD	McConnell 1996
Microsoft solutions framework	MSF	Microsoft 1994
Scrum	Scrum	Sutherland 1994

El 2001, en Utah, USA, se crea el Manifiesto ágil [2], documento firmado por los principales exponentes de la corriente de desarrollo ágil a nivel mundial, marcando un hito en la historia de este tipo de métodos.

Estos métodos se centran en otras dimensiones, distintas que en los métodos tradicionales, como por ejemplo el factor humano o el producto *software*. Esta es la filosofía de los métodos ágiles, las que 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.

Métodos híbridos

Los métodos híbridos constituyen una mezcla de prácticas y artefactos que no necesariamente provienen de una misma metodología, ni son una variación de una metodología ágil o tradicional. Los métodos híbridos basan su existencia en las debilidades de los métodos anteriormente nombrados, con la finalidad de crear un método robusto pero al mismo tiempo flexible, que combine las bondades de dos o más metodologías ágiles.

Los métodos híbridos pretenden retomar las ventajas de los métodos mencionados anteriormente, de tal forma que son una combinación de las mejores prácticas de cada uno de ellos.

La nueva tendencia en ingeniería de software es diseñar metodologías híbridas. Esta propuesta es atribuida a Ivar Jacobson, uno de los tres creadores de UML (Unified Modeling Language, Lenguaje Unificado de Modelado, Object Management group, 2011); creador de UP (Unified Process, Proceso Unificado), y ahora creador de EssUP (Essential Unified Process). EssUP es una metodología híbrida que combina RUP con Scrum.

Desarrollo de *Software* para Dispositivos Móviles

Software para dispositivos móviles:

Con el advenimiento de los teléfonos inteligentes, el mercado del desarrollo de *software* para dispositivos móviles (SDM) experimentó un crecimiento constante desde el 2005 a la fecha, generando oportunidades para que los Métodos Ágiles encontraran un lugar para posicionarse y evolucionar.

Una de las principales tecnologías que requieren hoy de estas metodologías es el desarrollo, cada vez más masivo, de SDM (teléfonos inteligentes o tabletas electrónicas), dispositivos que ya forman parte del uso diario de millones de personas en el mundo. Estos dispositivos mantienen una base lógica arquitectónica similar a la de cualquier computador, corriendo un sistema operativo como base operacional, sobre la cual pueden instalarse múltiples aplicaciones para comodidad del usuario. Estas aplicaciones son, efectivamente, mucho más pequeñas que el *software* que se construía hace décadas, esto principalmente por dos razones. Primero, la tecnología de desarrollo de *software* ha evolucionado a tal punto que no se requiere conocer en profundidad la comunicación entre la máquina y el sistema operativo para construir *software*, sino que simplemente se construye a base de las herramientas provistas por el mismo lenguaje de programación. Segundo, pero no menos importante, es la amplia cantidad de desarrolladores y de metodologías que han apostado por comprender y optimizar al máximo los procesos de desarrollo en pos de disminuir la

longitud del *software*, que para estos dispositivos, debe ser lo más pequeño posible para no sobrecargar el uso del *hardware*.

El SDM tiene que satisfacer varios requerimientos y condicionantes especiales ([3], [4] y [5]) que lo hace relativamente más complejo. Algunos de ellos se detallan en la Tabla 1.

Desarrollo de SDM:
No todas las metodologías están bien adaptadas para el desarrollo de SDM, pero sin duda, el principio de agilidad que radica en ellas son la base fundamental para comprender el real funcionamiento que debe tener una metodología ágil enfocada en el desarrollo de SDM.

Considerando lo anterior, en la Tabla 2 se presentan algunas de las características principales que, según Rahimian y Ramsin [6], debería satisfacer todo método ágil orientado al desarrollo de SDM.

Tabla 1. Requerimientos y condicionantes especiales del software móvil

Requerimiento	Descripción
Canal de comunicación	La disponibilidad, las desconexiones, la variabilidad del ancho de banda, la heterogeneidad de redes o los riesgos de seguridad.
Movilidad	La migración de direcciones, alta latencia debido a cambio de estación base (*Roaming*) o la gestión de la información dependiente de localización.
Portabilidad	Implica varias limitaciones físicas directamente relacionadas con el factor de forma de los mismos, como el tamaño de las pantallas o del teclado, limitando también el número de teclas y su disposición.
Capacidades limitadas de los terminales	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.
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 estas es mucho mayor que en el *software* tradicional, debido al entorno móvil que manejan, complicándose todavía más debido a la limitación de estos dispositivos.
Usabilidad	Las necesidades específicas de amplios y variados grupos de usuarios, combinados con la diversidad de plataformas tecnológicas y dispositivos.
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.

Fuente: Elaboración propia basada en [3], [4] y [5].

Tabla 2. Características de un método ágil orientado al desarrollo de SDM

Característica	Descripción
Agilidad	Las metodologías ágiles mejoran la flexibilidad del desarrollo y la productividad, proveyendo métodos que se adaptan a los cambios y que aprenden de la experiencia. Características específicas de los entornos móviles deben ser: procesos iterativos e incrementales, desarrollo conducido por pruebas, procesos adaptativos, hablar con el cliente continuamente, desarrolladores altamente cualificados, asegurar la calidad, revisiones continuas del proceso, priorización de los requerimientos y bajo *time-to-market*.
Conciencia de mercado	El mercado actual está orientado hacia los productos *software,* por lo que un proceso de desarrollo de SDM debería enfocarse al desarrollo del producto y no del proyecto. Consecuentemente, los procesos deben enfocarse al nicho del negocio, usando las prácticas de NPD (*New Product Development)*, Ulrich y Eppinger [7]) haciendo un análisis del mercado. Toda esta información mitigará las dudas y los riesgos. Procesos orientados al mercado seguirán una planificación bastante estricta en lo que se refiere a requerimientos de *time-to-market*. Antes los procesos estaban orientados a las actividades técnicas, ahora tienen que hacer un balance entre las actividades orientadas a mercado y las técnicas.
Soporte a toda la línea de producción de *software*	Se refiere al "conjunto de sistemas intensivos de *software* compartiendo un conjunto de características comunes que satisfacen las necesidades de un segmento particular del mercado y que son desarrolladas con una serie de valores centrales en una forma predeterminada" [8]. El ciclo de vida es más corto, pudiendo desarrollar una familia de productos SDM en un solo intento, reduciendo costos. Debe tener especial importancia la línea de producción para la mejora de la calidad del *software*.

Característica	Descripción
Desarrollo basado en arquitectura	La eficiencia de la línea de producción de *software* depende del desarrollo de una plataforma común, por lo que la necesidad de una arquitectura genérica para una clase de productos es esencial, pudiendo reconfigurarse de forma específica para cada componente o producto determinado.
Soporte para reusabilidad	El desarrollo basado en componentes y el basado en capas reducen los costos de desarrollo, agiliza la entrega del producto y hace el *software* menos propenso a errores, ya que los componentes no deben ser hechos desde cero cada vez.
Inclusión de sesiones de revisión y de aprendizaje	La metodología debería incorporar sesiones de revisión en todo el proceso para asegurar el análisis del producto y sesiones de aprendizaje después de la entrega de cada producto para que la experiencia sea analizada y registrada, y así la abstracción del conocimiento obtenido realimente a todo el equipo.
Especificación temprana de la arquitectura física:	La arquitectura física debe ser elaborada en las etapas tempranas del desarrollo del *software* gracias a que un alto número de riesgos técnicos son inherentes a las limitaciones de los dispositivos móviles y las diferencias en la implementación pueden ser obtenidas de características básicas. El uso de un prototipo mitigaría dichos riesgos técnicos.

Fuente: Elaboración basada en [6].

Trabajo relacionado

En la literatura existen numerosos trabajos que proponen la creación de herramientas para evaluar metodologías ágiles y metodologías ágiles de desarrollo de *software*. Sin embargo, existe poca literatura relacionada con el desarrollo ágil de SDM, pero se debe destacar el trabajo realizado por Blanco, Camarero, Fumero, Werterski y Rodríguez [9] que exponen el funcionamiento de una metodología híbrida de desarrollo.

El planteamiento del ciclo de vida de la metodología expuesta por dichos autores se basa en el desarrollo adaptativo de *software* (ASD) y en el diseño de nuevos productos (NPD), en conjunto con el diseño híbrido de metodologías y el modelo en cascada tradicional. El principal valor de este trabajo radica en exponer en términos sencillos las directrices principales que afectan al diseño de una metodología ágil enfocada en el desarrollo de SDM.

Es importante señalar también el trabajo realizado por Visconti [10], el que rescata un conjunto de valores a partir del Manifiesto Ágil y propone el concepto de "Ideal Ágil". Desde este concepto, elabora una comparación entre dos de las más conocidas metodologías ágiles, exponiendo los resultados e inferencias logradas. El trabajo de Visconti constituirá una base intelectual importante al momento de elaborar la herramienta de evaluación.

Otro de los autores clave en el desarrollo de metodologías ágiles y en especial en el desarrollo de metodologías de desarrollo de SDM es Abrahamsson, Hanhineva, Hulkko, Ihme, Jäälinoja, Korkala, Koskela, Kyllönen y Salo [11] quienes exponen el desarrollo, diseño y funcionamiento de Mobile-D, una de las pocas metodologías con certificación CMMI orientadas al desarrollo de SDM. En su trabajo, se analizan diversas metodologías, además de variadas prácticas y sus efectos en la productividad. Abrahamsson, Hanhineva, Hulkko, Ihme, Jäälinoja, Korkala, Koskela, Kyllönen y Salo [12] también analizan en profundidad diversas metodologías ágiles en búsqueda de encontrar o definir el "Pensamiento ágil" y cómo el desarrollo de *software* se ve afectado por lo que él denomina *Peopleware*, el factor humano.

DISEÑO CONCEPTUAL DEL PROCESO PARA CONSTRUIR EL MÉTODO

El diseño del método ágil híbrido para desarrollar SDM se basó en la ejecución de 5 etapas [13]:

- Análisis de SDM y los métodos ágiles existentes. Se analizan en profundidad tanto las propiedades de SDM como los métodos ágiles existentes. Como salida se obtienen, respectivamente, los requerimientos y condicionantes especiales de SDM y las características de un método ágil orientado al desarrollo de SDM.
- Diseño de un enfoque evaluativo. Se establece un enfoque para evaluar los métodos estudiados con la etapa anterior, mediante tres pasos:
  - Prueba de identificación basado en los 4 valores fundamentales de la agilidad.
  - Prueba de agilidad sobre la base de los 12 principios de la agilidad.
  - Prueba de agilidad en el desarrollo de SDM en función de las características de un método ágil orientado al desarrollo de SDM.

  Como resultado se establece el enfoque y el procedimiento evaluativo.

- Aplicación del enfoque evaluativo. Se evalúan los métodos más referenciados por los distintos autores. Como salida se obtiene la Tabla de un índice de agilidad de cada método y fortalezas, debilidades de los métodos y propuestas de mejoras, que se usarán para diseñar el método propuesto.
- Desarrollo del método híbrido ágil para el desarrollo de SDM. Se diseña el método propuesto considerando las salidas anteriores y tres aspectos adicionales:
  - Requerimiento para orientarse a SDM.
  - Requerimiento para orientarse a empresas con grupos pequeños
  - Requerimiento para crear un prototipo mínimo viable en 30 días

  El modelo derivado es evaluado con el enfoque desarrollado en la etapa anterior.
- Aplicación del método a un caso real. Se aplica el método a un caso real, registrando los resultados.

Las etapas, sus entradas, salidas y sus componentes, se muestran en la Figura 1.

MÉTODO PROPUESTO

A continuación se describe un perfil del nuevo método, su filosofía, ciclo de vida, roles, prácticas y artefactos, estos estarán orientados a mejorar la experiencia de desarrollo de *software*, así también los niveles de velocidad, comunicación y planificación, con tal de alcanzar los más altos niveles de productividad y cumplir con el objetivo principal de tener un prototipo mínimo viable (PMV) en 30 días, todo lo anterior dentro del marco establecido del Manifiesto Ágil [2]. La aplicación y resultados de las etapas 1 al 4 del diseño del método ágil híbrido pueden ser vistos en [13].

Es importante destacar que el método propuesto es independiente del estilo de desarrollo que se desee usar.

Perfil del método

Nombre: "Método ágil híbrido para desarrollar *software* en dispositivos móviles".

Principios/ Conceptos clave:

Equipos pequeños y altamente cohesionados (*grupos pequeños*), comunicación, documentación ágil, construcción basada en características, ciclos cortos de trabajo, entregas constantes, permanente participación del cliente, sesiones de retroalimentación, colaboración entre los miembros del equipo (*ganking*).

Descripción general:

Se basa principalmente en dividir el proceso de desarrollo en varios macrociclos de 1 mes cada

Tabla 3. Resumen de las actividades en un macrociclo.

	DÍA 1		DÍA 2		DÍA 3		DÍA 4		DÍA 5		DÍA 6		DÍA 7	
S0	SEMANA DE PLANIFICACIÓN DEL PROYECTO Y DEL PRIMER MACROCICLO													
S1	AR	DP	P	P	P	P	P	T	T	E	FB	C	OBS	TRANS
S2	AR	DP	P	P	P	P	P	T	T	E	BS	C	OBS	TRANS
S3	AR	DP	P	P	P	P	P	T	T	E	FB	C	OBS	TRANS
S4	AR	DP	P	P	P	P	P	T	T	E	BS	C	OBS	TRANS
	SEMANA DE *FEEDBACK*, PLANIFICACIÓN DEL SIGUIENTE MACROCICLO													

Donde:
S0 a S4: Semana 0 a semana 4.
AR: Análisis de requerimientos.
DP: Diseño del producto (entrega semanal: diseño del microciclo).
P: Programación/Producción.
T: *Testing*.
E: Entrega del producto o característica semanal.
FB: Sesión de *feedback*.
BS: Jornada de *brainstorming*.
C: Sesión de comunicación y distensión.
I: Investigación.
OBS: Sesión de revisión del avance, detección de obstáculos y dificultades.
TRANS: Sesión de transición. Preparación de los detalles generales del siguiente microciclo.

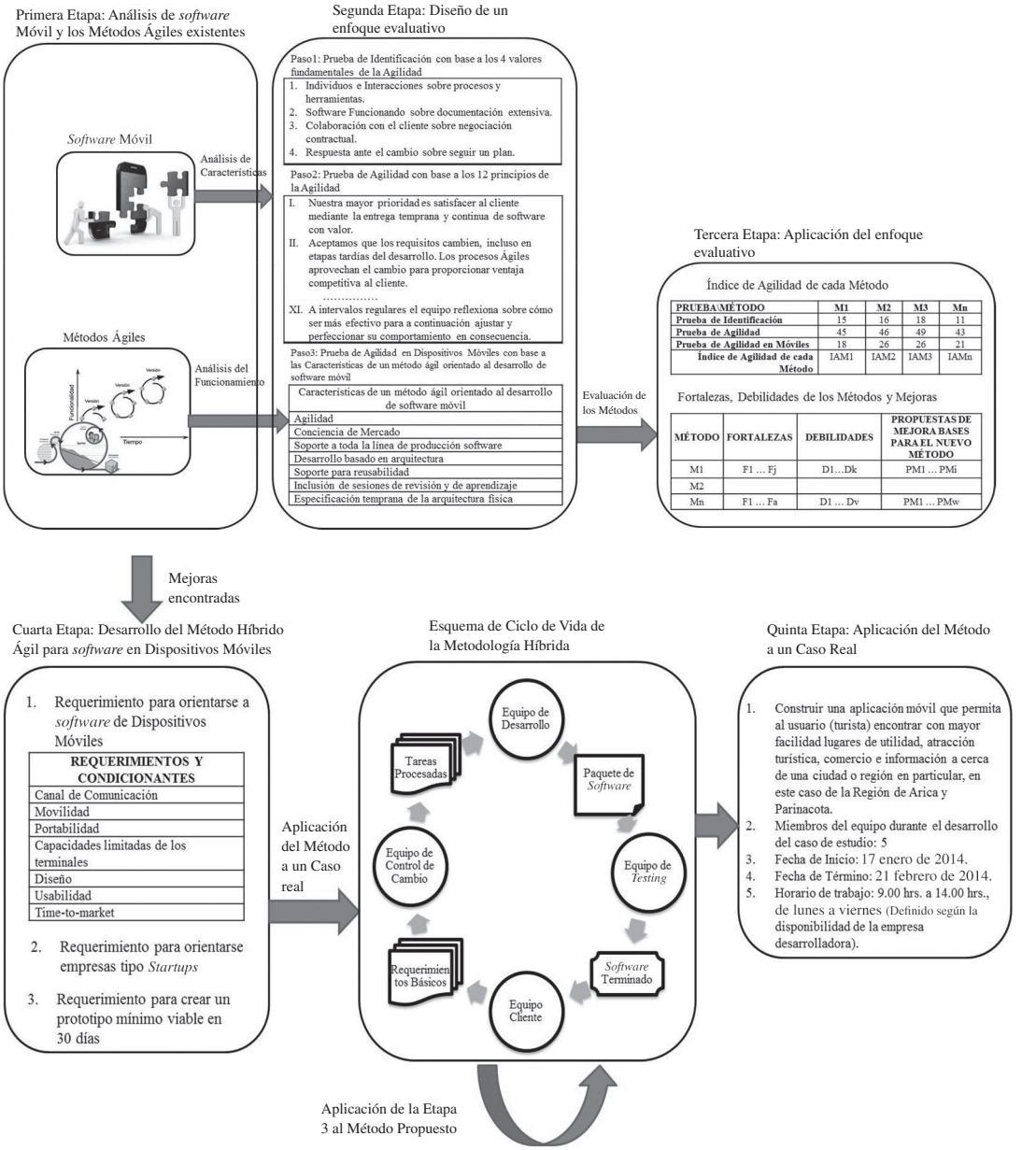

Figura 1. Diagrama del diseño conceptual del proceso para construir el método.

uno (30 días). Cada macrociclo es un paquete de funcionalidades o características del *software* final. Para desarrollar cada característica se divide cada macrociclo en 4 microciclos de 1 semana cada uno (5 a 7 días, dependiendo de la disponibilidad del equipo). Dentro de cada microciclo los días corresponden a distintas etapas de desarrollo (similares a las del modelo en cascada) enfocados en desarrollar la característica de la semana. En la Tabla 3 se muestra el cronograma de trabajo por cada macrociclo. Cada día se divide en dos jornadas y cada semana en 7 días de trabajo.

Ciclo de vida del método

El ciclo de vida en formato cíclico es una de las herramientas más reconocidas en las metodologías ágiles, porque permiten mantener un mejor control del avance e ir construyendo *software* de manera

incremental. Se ha decidido optar por dicho formato para que el cliente siempre pueda detener el proceso de desarrollo y pueda disponer de un producto de *software* funcional aunque prematuro. Además permite modularizar todo el proceso, con el objetivo de hacer más manejable la construcción de *software* de mediana o gran envergadura.

Como se muestra en la Figura 2, el equipo fue subdividido en 4 roles principales, el equipo/rol de: desarrollo, control de cambio, cliente y *Testing*. La interacción entre ellos después de cada entrega es cíclica: El cliente propone mejoras o cambios, el equipo cliente los procesa como nuevos requerimientos, el equipo de control de cambio transforma esos requerimientos en tareas a realizar por el equipo de desarrollo, estas tareas son procesadas y decretadas como terminadas una vez que han pasado por el equipo de *testing* y este haya decretado su aprobación, en este punto el producto debería estar preparado para ser presentado nuevamente al cliente.

Tomando en consideración las filosofías de cada una de las metodologías estudiadas en la tesis "Aplicación de metodologías ágiles para el desarrollo de *software* en dispositivos móviles" [13], se ha optado por establecer los 4 roles principales mencionados en el párrafo anterior, esto porque se considera que el proceso cíclico de desarrollo de *software* debe tener una mecánica de transformación, desde los requerimientos hasta el producto de *software*, para ello es necesario que exista un especialista en la toma de requerimientos y control del usuario/cliente (equipo cliente), una persona o grupo de personas que pueda convertir cada uno de los requerimientos en tareas y estructure la construcción del *software* de forma ordenada (equipo de control de cambio), un equipo que construya el *software* lo más rápido y ordenadamente posible, siguiendo todas las directrices establecidas por el equipo de control de cambio (equipo de desarrollo) y finalmente un equipo de personas que se encargue de evaluar en toda su magnitud el *software* desarrollado y asegure que cumple con los criterios de aceptación expuestos por el cliente (equipo de *testing*).

Roles

Programador (equipo de desarrollo): El equipo de desarrollo se compone regularmente de 5 personas, uno de ellos es un líder que regula la comunicación entre el equipo de desarrollo y los demás equipos. Se encarga de administrar la prioridad de las tareas y sus responsables, además de producir el *software* definido en las tareas entregadas por el equipo de control de cambio.

Diseñador/Arquitecto (equipo de control de cambio): El equipo de control de cambio se

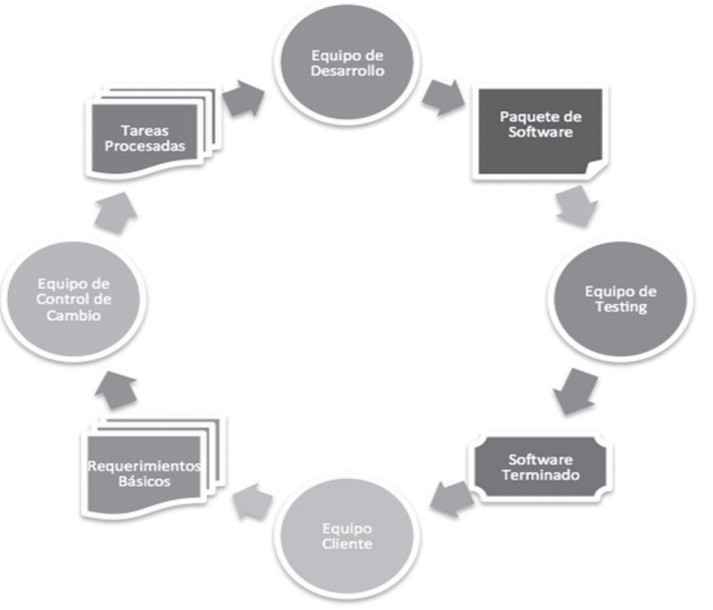

Figura 2. Esquema de ciclo de vida de la nueva metodología.

compone regularmente de 2 a 3 personas. Suelen ser programadores experimentados o analistas capaces de visualizar el *software* como un todo. Su trabajo consiste en tomar los requerimientos procesados por el equipo cliente y armar la arquitectura del *software*, además de planificar y organizar el *Product Backlog*.

Cliente (equipo cliente): Equipo generalmente compuesto de 1 persona. Es el responsable de mantener un contacto permanente con el cliente y hacer suyo los requerimientos que el cliente solicita. Se responsabiliza por transformar las características funcionales y no funcionales del futuro *software* en requerimientos o características entendibles por el equipo de control de cambio. También es el encargado de representar al equipo completo ante el cliente y de hacer las veces de intermediario o mediador ante posibles conflictos.

Tester (equipo de testing): El equipo de *testing* se compone de 2 personas. Uno de ellos se encarga de generar las pruebas necesarias para las distintas características del *software* mientras que el otro se encarga de documentar el proceso y hacer *tracking* de los errores y correcciones generadas. Quien cumpla con este rol debe dar por terminada una tarea una vez que haya pasado por el debido proceso de *Testing*.

El uso de los 4 roles obedece al hecho de que la herramienta de evaluación considera en la primera prueba, como primer postulado de evaluación, la definición clara de roles y una correcta distribución de las tareas y actividades, con el fin de satisfacer el primer postulado fundamental del manifiesto ágil: *Valorar a los individuos e interacciones sobre procesos y herramientas*. Así, la metodología basa su productividad en liderazgos impulsados por el valor que cada persona representa para el equipo, más que por los procesos y actividades previamente planificadas.

Para un equipo de desarrollo de *software* convencional (no grupos pequeños) se recomienda un mínimo de 10 personas para tener un equipo de desarrollo óptimo (una persona para el equipo cliente, dos personas para el equipo de control de cambio, dos personas para el equipo de *testing* y cinco personas para el equipo de desarrollo). Mientras que para el caso de pequeñas empresas, debido a lo difícil que es mantener un equipo de tal magnitud, se recomienda reducir el equipo a solo cuatro personas, una por cada rol (mínimo posible para la metodología). La versión de cuatro personas utiliza exactamente los mismos principios, artefactos y códigos, la única diferencia se verá en la capacidad de producción y en los tiempos de entrega, que serán notablemente más reducidos que en el equipo óptimo de la versión para 10 personas.

En la actualidad muchas de las empresas desarrolladoras de *software* comienzan a trabajar con equipos bastante reducidos y sin mucho control de lo que realizan, lo que desemboca frecuentemente en no cumplir con las fechas establecidas, no porque no se tenga la capacidad para cumplir con el objetivo, si no por una falta de orden y optimización de la distribución de las tareas, además de la intervención descontrolada del cliente en el equipo de desarrollo, generando un estado de presión permanente, obstaculizando el desarrollo sostenido y constante que debería tener un equipo normal de desarrollo de *software*.

Prácticas y artefactos

Ganking: Corresponde a una estrategia de colaboración que permite a cada miembro del equipo ayudar a quien más lo requiere en una etapa específica del desarrollo del proyecto. El *ganking* varía según la etapa del proyecto, en etapas tempranas (*early game*), el equipo de control de cambio es quien se lleva la mayor parte de la carga y es *gankeado* o auxiliado por el equipo cliente y el equipo de desarrollo. Del mismo modo, se auxilia al equipo de desarrollo en la etapa media de proyecto (*middle game*), incentivándose al equipo de control de cambio y al equipo de *testing* a dar soporte al Equipo de desarrollo quien se lleva la mayor parte del trabajo. Durante las etapas finales del proyecto (*late game*), el apoyo está dirigido al equipo de *testing*, por parte del equipo de desarrollo y del equipo cliente.

La práctica del *ganking* viene a solucionar el constante problema de comunicación que tienen metodologías ágiles como XP o FDD. La productividad y agilidad dependen del talento de las individualidades. El *ganking* propone un desarrollo primordialmente colaborativo, todas las partes contribuyen, en mayor o menor medida, al desarrollo del producto de *software* final.

Semana de 40 Horas: La semana no debe superar las 40 horas de trabajo. Se considera que es el tiempo ideal para mantener un desarrollo sostenido y ágil.

Estrategia de Diversificación Holística: Del mismo modo que en *crystal clear*, la idea central es incluir múltiples especialidades en un solo equipo. También permite formar pequeños equipos con el conocimiento necesario para el desarrollo del proyecto, considerando asuntos como la localización física del equipo, comunicación y documentación, y coordinación de múltiples grupos de trabajo. Facilita la expansión del equipo dependiendo de la longitud del proyecto, permitiendo atacar múltiples funcionalidades a la vez.

Reuniones Diarias (Reunión de Coordinación): Una de las prácticas más comunes en la mayoría de los métodos ágiles son las Reuniones Diarias. Permiten coordinar y establecer un plan de acción válido por el día. Permite establecer qué es lo que se ha hecho, lo que se hará y cómo se ejecutará una tarea en particular, además de definir quiénes serán los responsables de cada tarea.

Entregas Semanales: La estructura general de la metodología es entregar *software* funcional al final de cada semana. El mercado tiende a ser bastante exigente y la metodología debe tener en consideración que el proyecto puede acabarse en cualquier minuto y debe existir un *software* funcional y con posibilidades de crecimiento futuro.

Líder Diario: Dependiendo de la actividad que se presente durante el día, existirá un líder diario. Generalmente este líder es quien recibe la mayor cantidad de carga de trabajo. El líder puede administrar y gestionar la ayuda recibida por sus pares, además de dirigir la atención a las tareas más pesadas, para optimizar de mejor manera el proceso productivo.

Unidades de Proceso: Macrociclo y Microciclo: El proceso de desarrollo se estructura en macrociclos, constituidos cada uno por cuatro microciclos de una semana de cada uno. Esta estructura permite modularizar el proceso de desarrollo, dando la facilidad de gestionar el proceso semana a semana.

Jornadas de Reflexión, Retroalimentación y Comunicación: Estas jornadas están incluidas como parte de las actividades normales durante la semana. Permiten mejorar la comunicación, el nivel de entendimiento entre las partes, aclarar dudas y mejorar la cohesión del equipo.

Semana de Planificación (Semana 0): Esta semana es una etapa crucial en el proceso de desarrollo. Permite armar el *feature backlog* y el *product backlog*, organizar los equipos, definir el *chatlog* y en general definir la forma de llevar el proyecto.

Feature backlog: Corresponde al conjunto de tareas que componen una característica del *software*. Se recomienda establecer, seguir y completar un *feature backlog* a la semana por equipo. cada *feature backlog* posee tres secciones: "Por hacer", "En desarrollo" y "Listo", al igual que el *sprint backlog* de SCRUM, las tareas escritas en pequeñas fichas (el formato de cada ficha queda a libre disposición del equipo), pasan de la sección "Por Hacer" a "En Desarrollo" una vez que se ha establecido los responsables de cada tarea. Durante esa semana, las tareas que se completen deben pasar a la sección "Listo" para evaluar el avance al final de la semana en las sesiones de *feedback* o Comunicación.

Product backlog: Corresponde al conjunto de *features* o características del producto de *software* final. Ellas se transforman en tareas y se agrupan posteriormente en varios *feature backlog*. El *product backlog* se genera a partir de los requerimientos extraídos por parte del Equipo Cliente en la Semana de Planificación.

Programación y testing XP: El proceso de producción del Equipo de Desarrollo y el Equipo de *Testing* se realiza siguiendo los parámetros establecidos por XP. Debido que una de las características principales de esta metodología es la velocidad de producción.

Pizarra de Ideas: Se propone como artefacto auxiliar para exponer ideas o mejoras al *software* en desarrollo. Este artefacto permite que el equipo de desarrollo pueda ofrecer un mejor producto desde la perspectiva de funcionalidad.

Chatlog: El equipo requiere de un medio para comunicarse fluidamente y que al mismo tiempo que sirva de documentación para mantener un registro de los cambios. Este medio puede ser tanto escrito como audiovisual. Se recomienda que sea una

herramienta o instrumento colaborativo en la nube para poder tener acceso permanente y en todo lugar. Este artefacto permite incentivar la proactividad y la comunicación exponiendo ideas, prácticas, dificultades y soluciones. Se utiliza también como método de documentación ágil. Es una plataforma de comunicación compartida escogida por los mismos miembros del equipo de acuerdo a su propia conformidad. Ejemplos de *Chatlog*: Grupo de *Facebook*, Conversaciones de *Twitter*, Listas de Correos Electrónicos, Grabadoras de Audio, Grabadoras en Video, etc. No se recomienda el uso de documentación formal debido a que su extensión pone lento el proceso de desarrollo.

Modelo Canvas: Se propone como un artefacto auxiliar para visualizar el producto final como un todo más que como la suma de sus partes. El Modelo Canvas es un diseño que permite visualizar en solo una hoja los nueve aspectos fundamentales de un negocio. El objetivo es que el equipo mantenga una visión global del mercado en el que está inserto su desarrollo.

Objetivo Diario: Debido a que existe un líder por jornada, se decidió aumentar el nivel de concentración proponiendo cumplir al menos con un objetivo al día. Este objetivo puede ser una o un conjunto de tareas. La idea es que se cumpla a cabalidad para mantener un ritmo mínimo de avance en el proyecto.

Procesos

El desarrollo de *software* para cada macrociclo consta de tres fases principales. *early game, mid game, late game*.

Early game: Corresponde a la fase inicial del proyecto, generalmente hace referencia a la Semana 0 y 1, justo antes de la primera entrega. En esta fase el proyecto se encuentra en una etapa de ajuste, el equipo comienza a acomodarse para dar a luz los primeros avances del *software*, es por esto que el equipo de control de cambio se lleva la mayor parte del peso en esta etapa, ya que es el principal responsable de la descomposición de las tareas, asignación de prioridades y supervisión de los responsables de las mismas, así también como de los cambios y ajustes del proyecto que comienzan a sentirse a medida que se desarrollan las primeras tareas.

Durante esta etapa, el *ganking* se aplica de la siguiente manera:

1. Equipo control de cambio (*leader*): Es el protagonista de esta etapa. Requiere de la colaboración conjunta del equipo cliente y de desarrollo en las tareas de ajuste de tareas, cambios en las responsabilidades, *feedback* y repriorización de las tareas o actividades planificadas en la semana 0.
2. Equipo cliente (*ganker*): Colabora desde el punto de vista del cliente, ajustando los requerimientos, tareas, y funciones que necesitan que la aplicación realice. Provee la información necesaria para que las tareas sean lo más acotadas posibles, además de contribuir en cualquier duda o consulta que el equipo de control de cambio le solicite.
3. Equipo de desarrollo (*ganker*): Colabora desde el punto de vista del desarrollo del *software*. Provee toda la información necesaria para que las tareas y actividades encajen dentro de la arquitectura de desarrollo del sistema.

Middle game: Corresponde a la etapa media del macrociclo, generalmente desde la semana dos a la semana tres, hasta justo antes del comienzo de los procesos relacionados con el *testing* de la aplicación. Durante esta etapa, el objetivo de la metodología es mantener un desarrollo sostenido, rápido y constante del producto de *software*. Para lograr este objetivo, el equipo de desarrollo requiere de la asistencia del equipo de control de cambio y del equipo de *testing* en términos de optimización de las tareas y de orientación del *software* al *testing*.

Durante esta etapa, el *ganking* se aplica de la siguiente manera:

1. Equipo de desarrollo (*leader*): Este equipo es quien se lleva la mayor carga de trabajo. Durante esta etapa el equipo de desarrollo debe generar *software* en el menor tiempo posible, de una manera rápida y sostenida. Para mantener la fluidez del trabajo, el equipo de desarrollo debe mantenerse contantemente bajo retroalimentación respecto de los detalles funcionales de las tareas a realizar, así también como de la organización y arquitectura del sistema que se está desarrollando. Además de lo anterior, el equipo de desarrollo requiere

conocer la dirección en la que el equipo de *testing* dirige las pruebas de *software* con tal de orientar los esfuerzos en satisfacer aquellas pruebas antes de comenzar las actividades relacionadas con el proceso de *testing*.

2. Equipo control de cambio (*ganker*): el equipo de control de cambio ayuda al equipo de desarrollo entregando soporte de diseño, arquitectura y funcionalidad a todas las tareas y trabajos asignados al equipo de desarrollo. Todas las dudas y consultas relacionadas con la arquitectura general de la aplicación, su diseño y su aplicabilidad deben ser resueltas por el equipo de control de cambio.
3. Equipo de *testing* (*ganker*): debe asistir al equipo de desarrollo en aspectos relacionados con la entrega de información acerca de las pruebas que se realizarán durante las últimas fases del proyecto. El equipo de desarrollo debe mantenerse permanentemente informado acerca de los factores que influirán al momento de evaluar las entregas preliminares.

Late game: Corresponde a la etapa final del macrociclo e involucra la participación del Equipo de desarrollo y del equipo de *testing*. Durante la última semana, el equipo de *testing* es quien se lleva la mayor parte de la carga, ya que la mayoría de los paquetes de código ya han sido armados y deben ser testeados antes de ser presentados al cliente. Cuando alguno de los paquetes de *software* necesita mejoras, el equipo de desarrollo está siempre atento para generar estas modificaciones lo antes posible y así entregar al equipo cliente un *software* funcional y acorde a los requerimientos del cliente. El cliente también tiene una participación clave en esta etapa, ya que sin él no se dispondría del *feedback* necesario para dar por completada la etapa.

Durante esta etapa, el *ganking* se aplica de la siguiente manera:

1. Equipo de *testing* (*leader*): Es quien lidera este proceso, mantiene un estrecho vínculo con el equipo de desarrollo en el sentido de generar *software* funcional y sin errores orientado a satisfacer las necesidades del cliente.
2. Equipo desarrollo (*ganker*): Provee de la colaboración necesaria para satisfacer las necesidades de cambio del equipo de *testing*. Está a la escucha permanente para solucionar los problemas identificados por el equipo de *testing*.
3. Equipo cliente (*ganker*): Genera una retroalimentación permanente al equipo de *testing*, advirtiendo detalles funcionales y no funcionales del sistema. Ayuda a guiar el proceso de modificación y ajuste de requerimientos.

EVALUACIÓN DEL MÉTODO

El método desarrollado fue sometido a dos evaluaciones. La primera fue aplicando el enfoque de evaluación (ver Figura 1). La segunda fue la aplicación del método a un caso real.

Evaluación conceptual del método

Con el objetivo de probar adecuadamente el método se ha sometido a evaluación, según el enfoque expuesto en la Figura 1, y sus resultados en detalle son descritos en [13].

El puntaje de cada prueba corresponde a la suma de puntos asignados en Tablas de criterios que evaluaban si la metodología ágil propuesta era efectivamente una metodología ágil (prueba de identificación), qué tan ágil era (prueba de agilidad) y si efectivamente tenía un grado de aplicabilidad en la industria de los dispositivos móviles (prueba de agilidad en móviles).

Del proceso de evaluación, cuyos resultados finales se muestran en la Tabla 4, se obtuvieron las siguientes consecuencias:

Tabla 4. Resumen de la evaluación.

Prueba\Método	Método ágil híbrido
Prueba de identificación	18
Prueba de agilidad	58
Prueba de agilidad en móviles	32
Índice de agilidad	108

Prueba de identificación:

- El método posee una puntuación que denota un alto nivel de agilidad, sobre todo en aspectos vinculados a la relación con el cliente y con el desarrollo de *software* funcional, ambos considerados aspectos fundamentales en el desarrollo ágil de *software*.
- Destaca en menor medida que el método requiere un mínimo de planificación, solo orientada a

mantener una autoorganización adecuada y así cumplir con los objetivos propuestos.

Prueba de agilidad:

- El nuevo método alcanza casi el máximo de la puntuación, debido a que se ha creado tomando en consideración la mayoría de los aspectos que influyen en el manifiesto ágil, estos son también los aspectos a considerar al crear el enfoque de evaluación.
- Abarca casi todos los aspectos del manifiesto ágil, cumpliéndolos casi todos a cabalidad. Como aspecto negativo, posee y aplica las actividades y prácticas con frecuencia, pero su aplicación no es fundamental para el desarrollo del proyecto.

Prueba de agilidad en móviles:

- Como se presenta en la Tabla 4, la puntuación orientada al desarrollo de SDM es bastante alta, ello permite a esta metodología rendir adecuadamente en entornos de trabajo orientados al desarrollo de SDM.
- El alto nivel de agilidad en conjunción con los altos niveles de conciencia de mercado y retroalimentación hacen que el desarrollo de SDM sea mucho más sencillo y fácil de orientar.

Evaluación mediante caso real

En esta sección se resume el proceso de desarrollo para la creación de un SDM, usando el nuevo método. A continuación los detalles relevantes del proyecto:

Empresa Ejecutora: UANACO[2]
Nombre del Proyecto: GOUTU
Propuesta de Valor: Construir una aplicación móvil que permita al usuario (turista) encontrar con mayor facilidad lugares de utilidad, atracción turística, comercio e información acerca de una ciudad o región en particular, en este caso de la Región de Arica y Parinacota, en Chile.

Miembros del equipo durante el desarrollo del caso de estudio:

- Patricio (equipo de desarrollo y testing en android).
- Sebastián (equipo de desarrollo y testing en android).
- Gustavo (equipo de desarrollo en ios, equipo cliente y control de cambio). Respecto del rol múltiple ejercido por este miembro del equipo, debemos destacar que él es el líder de la empresa y es, a su vez, el gestor de la idea y por ende a quien se le debe desarrollar el *software* (cliente). En este caso, no existe un cliente externo a la empresa.
- Felipe (equipo de desarrollo y *testing* en IOS).
- Piero (diseño de interfaz).
- Ignacio (método de desarrollo, equipo de control de cambio).

Fecha de Inicio: 17 enero de 2014.
Fecha de Término: 21 febrero de 2014
Horario de trabajo: 9.00 hrs. a 14.00 hrs., de lunes a viernes (Definido según la disponibilidad de la empresa desarrolladora).
Cronograma del Proyecto: Se describe en la Tabla 5.

Número de macrociclos planificados: 1
Número de microciclos planificados: 4
Número de microciclos no planificados: 1

Contexto: Industrial - Empresarial.
Número de casos de Uso: 8.
Número de módulos a Desarrollar: 4
Número de funciones: 4 por módulo.

[2] https://www.facebook.com/Uanaco

Tabla 5. Cronograma de trabajo método híbrido para el caso práctico

	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7		
S0	SEMANA DE PLANIFICACIÓN DEL PROYECTO Y DEL PRIMER MACROCICLO								
S1	AR/DP	P/I	P/OBS	P/FB	T/E				
S2	AR/DP	P/I	P/OBS	P/FB	T/E				
S3	AR/DP	P/I	P/OBS	P/FB	T/E				
S4	AR/DP	P/I	P/OBS	P/FB	T/E				
	SEMANA DE *FEEDBACK*, PLANIFICACIÓN DEL SIGUIENTE MACROCICLO								

Nivel de rendimiento al final del proceso: 100%, se cumple con todos los casos de usos y módulos al final del proceso de desarrollo.

Discusión caso de estudio

El desarrollo del proyecto se llevó a cabo con casi total normalidad, salvo por algunos inconvenientes propios de los equipos que se están recién conociendo, en este caso, el equipo no había trabajado antes en proyectos de este tipo, es más, el equipo de desarrollo tenía muy poca experiencia antes de llevar a cabo este proyecto. Sin embargo, pese a las dificultades técnicas y de tiempo, la metodología ayudó al equipo a autoorganizarse para sortear adecuadamente estas dificultades y poder cumplir con los plazos establecidos.

En este caso, durante las fases de *ganking* se tomó la decisión de apoyar al equipo en labores de investigación y análisis, para así conseguir, primero, mejoras en la comunicación, discusión abierta, uso del *chatlog* como medio de comunicación y, además, con el fin de promover el aprendizaje mutuo, ya que este permitiría que las ayudas se entregaran cada vez con mayor precisión y prontitud.

En cuanto a la planificación, resultó como se había esperado, con un microciclo adicional libre en caso de cualquier eventualidad, el que terminó siendo usado para realizar ajustes generales del *software*.

Deben considerarse algunas limitaciones que influyeron en mayor o menor medida en el desarrollo del proyecto, como por ejemplo, el nivel de cohesión del equipo, el que en este caso era bastante alto. La capacidad de autoorganización mantenía la metodología funcionando apropiadamente. Además, no podemos dejar de lado el nivel de experiencia del equipo, este, a pesar de tener varios proyectos desarrollados, ninguno era tan ambicioso como el proyecto actual, poniendo al límite las capacidades del equipo.

Al momento de escribir este artículo el proyecto seguía su curso usando el mismo método. Se estimó que para completar el proyecto solo se requerirán tres a cuatro macrociclos más.

El empresario manifiesta su conformidad respecto del avance realizado, en conjunto con declarar que seguirá usando la metodología para el resto del desarrollo.

Respecto de los resultados obtenidos, el método propuesto mejoró las expectativas que se tuvieron con SCRUM, método usado en experiencias anteriores de la empresa. En el mismo periodo (30 días) el equipo experimentó un avance notablemente superior con la nueva metodología en comparación con la primera (SCRUM). De hecho, la cantidad de tareas realizadas y los objetivos alcanzados aumentaron en aproximadamente 30%. Las razones de este aumento aún están en evaluación, pero se estima que el grado de avance es producto de las mejoras en las prácticas de comunicación y en trabajo colaborativo que se implementaron durante el proyecto, fueron factores determinantes.

CONCLUSIONES

El método propuesto en este documento incorpora las mejores prácticas y artefactos de las metodologías mejor evaluadas del mercado, generando un nivel de productividad considerablemente alto en conjunto con elevados niveles de comunicación y un crecido componente de desarrollo enfocado al cliente.

El uso del *ganking* como práctica cotidiana y sistemática promueve la participación del equipo en diversas áreas del desarrollo, generando en cada miembro del equipo un alto nivel de conciencia acerca del producto que se está desarrollando, más como un todo que como la suma de sus partes.

El uso del *chatlog* constituye una práctica que satisface la necesidad de mantener al equipo permanentemente comunicado. La comunicación se mantiene registrada y es de libre acceso para todo el equipo. Esta práctica es muy poco frecuente en casi todos los métodos ágiles. No se documentan ni se promueven la innovación ni la mejora al mismo tiempo que se desarrolla el producto de *software*. Esta nueva metodología intenta no solo cumplir con los requerimientos del cliente en un tiempo determinado, sino también contar con un proceso documentado y registrado por la metodología, esto permitirá estudiar y en un futuro replicar el proceso.

El uso del método para el desarrollo de SDM permite a sus usuarios contar con un prototipo en un tiempo aproximado de 30 días. Este impacto solo se logra

gracias a la disposición del equipo al momento de recibir una nueva idea. Generalmente en equipos altamente cohesionados, recibir un nuevo método no es agradable, ya que tienen que reestructurar la mayoría de los cánones previamente aprendidos. Una de las ventajas de este método es que permite conservar estos cánones siempre y cuando su práctica no afecte el continuo desarrollo o la actividad de los roles responsables en el proyecto. Este nivel de flexibilidad se extrae de uno de los objetivos fundamentales de los métodos ágiles, mantener el desarrollo de *software* como una actividad flexible y humana, no como un proceso mecánico y estructurado.

Finalmente, cabe destacar el aporte del trabajo desarrollado, el que dice relación con una mejora significativa de las estadísticas de desarrollo de prototipos, ya que organiza las actividades, artefactos y roles de forma tal, que el análisis, diseño, construcción y pruebas se ejecute en un tiempo prudente, manteniendo siempre el control acerca de la evolución del producto y la transparencia del desarrollo hacia el cliente. Lo anterior se hace fundamental al momento de establecer una línea base antes de proseguir con una versión más acabada del *software*, lo que contribuye en gran medida a la transparencia y claridad en el desarrollo de *software* para aplicaciones en dispositivos móviles en fase temprana.

AGRADECIMIENTOS

Se agradece a los evaluadores, los que contribuyeron en el perfeccionamiento continuo de las ideas presentadas en él.

REFERENCIAS

[1] Rodrigo Andrés Mazzo Iturriaga, "Plan de Negocios para una Agencia de Comunicaciones que Administra y Desarrolla Plataformas Digitales". Tesis Magíster, 2014. Universidad de Chile. http://tesis.uchile.cl/bitstream/handle/2250/116156/cf-mazzo_ri.pdf?sequence=1, Fecha de consulta: julio de 2014.

[2] Agile Manifesto. 13 de enero de 2014. Disponible http://www.agilemanifesto.org/

[3] G. Forman and J. Zaharjon. "The challenges of mobile computing". IEEE Computing, Vol. 27 Nº 4, pp. 38-47. April, 1994. ISSN:0018-9162. DOI : 10.1109/2.274999.

[4] I.S. Heyes. "Just Enough Wireless Computing". Pretince Hall. 1era Edición. USA. 2002. ISBN:0130994618.

[5] M. Dunlop and S. Brewster. "The Challenge of Mobile Devices for Human Computer Interaction". Personal and Ubiquitous Computing. Vol. 6 N° 4, pp. 235-236. September, 2002. ISSN: 1617-4909. DOI: 10.1007/s007790200022.

[6] V. Rahimian and R. Ramsin. "Designing and agile methodology for mobile *software* development: a hybrid method engineering approach". Segunda Conferencia Internacional en Desafíos de Investigación en Ciencias de la Información. Marruecos. 3-6 Junio de 2008.

[7] K.T. Ulrich and S.D. Eppinger. "Product Design and Development". McGraw-Hill. 3ra edición. New York, USA. Vol. 1. 2004. ISBN: 0072471468.

[8] P. Clements and L. Northrop. "Software Product Lines, course notes of Product Line Systems Program". Software Engineering Institute, Carnegie Mellon University. Pittsburgh, Pensilvania, USA. 2003.

[9] P. Blanco, J. Camarero, A. Fumero, A. Werterski y P. Rodríguez. "Metodología de desarrollo ágil para sistemas móviles, introducción al desarrollo con Iphone y Android". Universidad Politécnica de Madrid, España. 2009.

[10] M. Visconti and C. Cook. "An Ideal Process Model for Agile Methods". Lecture Notes in Computer Science. Vol. 3009 (capítulo 31), pp. 431-441. Berlín, Heidelberg. 2004.

[11] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen and O. Salo. 2004. "Mobile-D: an agile approach for mobile application development". In Companion To the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (Vancouver, BC, CANADA, October 24-28, 2004). OOPSLA '04, pp. 174-175. ACM. New York, NY, USA. 2004.

[12] P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta. "Agile Software Development Methods". Review and Analysis. VTT Publications 478. 2002.

[13] I. Leiva. "Aplicación de Metodologías Ágiles para el desarrollo de Software en Dispositivos Móviles". Tesis para optar al grado de Magíster en Ingeniería de Software. Universidad de Tarapacá. Arica, Chile. 2013.