Metodología de desarrollo ágil para sistemas móviles
Introducción al desarrollo con *Android* y el *iPhone*

Paco Blanco, Julio Camarero, Antonio Fumero, Adam Werterski, Pedro Rodríguez
Universidad Politécnica de Madrid

1. Introducción

Aunque los sistemas móviles comerciales no han resultado funcionalmente tan satisfactorios como prometían, los operadores móviles y los proveedores de software de valor añadido para móviles tienen la esperanza de que el efecto del 3G/3.5G y la cantidad de móviles no-tontos que se están vendiendo actualmente den un cambio drástico a la industria de las aplicaciones móviles, consiguiendo expandir el uso de dichas aplicaciones y servicios comercialmente. Ejemplos de aplicaciones comerciales pueden ser: servicios de información, anuncios, servicios basados en contenido, servicios basados en localización o aplicaciones de pago sobre móvil.

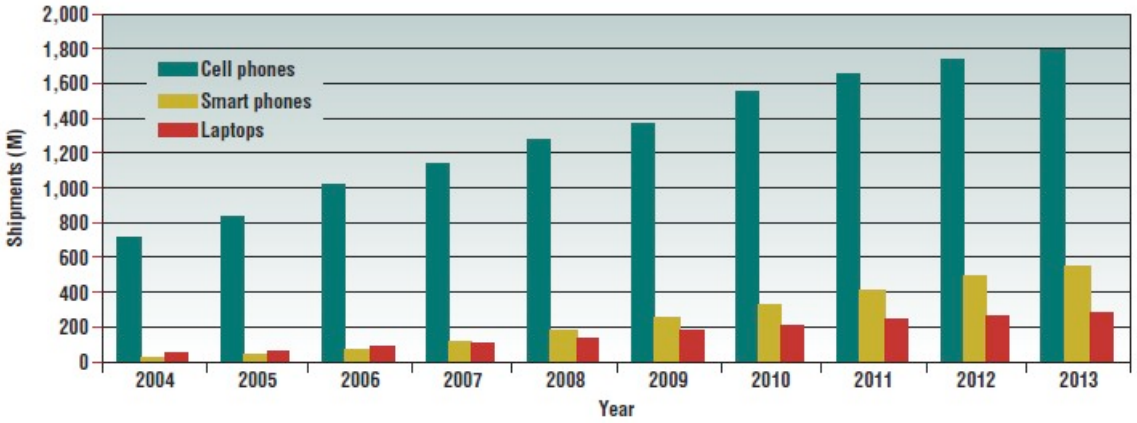

Figura 1. Evolución de los pedidos de móviles, terminales inteligentes y portátiles en el mundo [1a]

Así, en este último año, la fiebre de los móviles con pantalla táctil, desatada primeramente por el iPhone, ha revolucionado el mercado, ya no sólo de la venta de móviles sino todo el verdadero y jugoso negocio de las aplicaciones móviles. Y es que estos terminales se han convertido en el ordenador del futuro con el valor añadido de disponibilidad total y de la comodidad y portabilidad.

Un indicador nada riguroso, pero muy eficaz, para medir la emergencia de ciertas tendencias en el ámbito del desarrollo de software, es la actividad de las listas de correo y

foros de las comunidades de desarrollo. En O'Rielly Media hicieron el ejercicio[1] a mediados del año pasado, tras unos meses después del lanzamiento de Android, obteniendo el resultado que se puede observar en la figura 2. Sirva como justificación ad-hoc de la elección de esas dos plataformas en este breve trabajo de investigación.

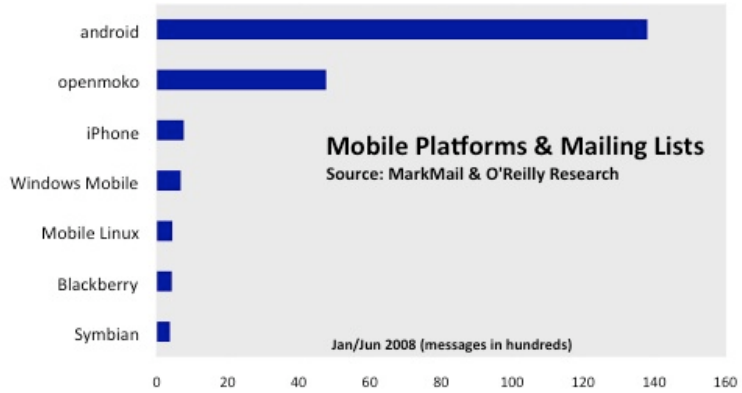

Figura 2. Listas de correo de desarrolladores interesados en determinadas plataformas abiertas y propias de fabricante (O'Reilly Media)

El sistema operativo que posee esta clase de terminales ofrece un sinfín de posibilidades, como si el de un ordenador se tratará; y todo esto combinado con la actual potencia de procesamiento y capacidad de almacenamiento que permiten la ejecución no sólo de aplicaciones ligeras sino de una gran parte de las pesadas; sin olvidar otras funcionalidades incorporadas a los móviles, como son el GPS, pantallas táctiles, llamadas telefónicas, posibilidad de enviar mensajes cortos, etc. Por eso, se va a hacer la comparativa con lo que se cree que son los dos sistemas que más dinero y expectación están generando y generaran, al menos en un futuro próximo, iPhone y Android.

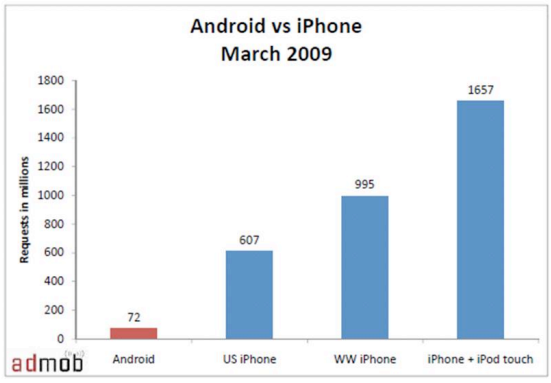

Figura 3. iPhone frente al Android en número de pedidos (fuente: *blog* Admob[2])

[1] http://radar.oreilly.com/2008/07/interest-in-the-iPhone-android.html
[2] http://blog.admob.com/2009/04/page/2/

A medida que el negocio de las aplicaciones móviles se va expandiendo y haciéndose rentable, se tienen que investigar las metodologías óptimas de desarrollo software para tales aplicaciones y entornos que lleven dicho desarrollo a éxito de una forma atractiva y eficiente; todo esto antes de que sea demasiado tarde y que el mercado esté tan maduro que las empresas hayan optado por las metodologías ya implementadas de software tradicional. Aunque éstas difieran bastante de las metodologías necesarias debido a que el software móvil debe satisfacer requerimientos y restricciones especiales. A pesar de estas características tan especiales, el software producido debe exigir un alto nivel de calidad para que este puede operar propiamente en la cantidad ingente de terminales que llenan el mercado actual y el venidero. El desarrollador de aplicaciones móviles se enfrenta, además, con un escenario muy fragmentado, formado por multitud de plataforma incompatibles, como Symbian, Windows Mobile, Brew, iPhone SDK, Android, Linux o Java. Todo esto hace que el proceso de desarrollo para plataformas móviles sea más complejo.

En las siguientes secciones se verá la idoneidad de las metodologías de tipo ágiles para el desarrollo de software móvil, haciendo primero una explicación de lo que se entiende por metodologías ágiles, después se plantean las características especiales y problemáticas que posee el desarrollo en entornos móviles continuando con las características que debería tener una metodología ágil para el desarrollo de software en móviles. Después se comenta dos metodologías ágiles específicas para el desarrollo de aplicaciones móviles. A continuación, se compara la primera iteración de la metodología híbrida explicada sobre los dos sistemas que se cree que ocuparán los dos primeros puestos en el negocio de las aplicaciones móviles en el futuro próximo, Android y iPhone.

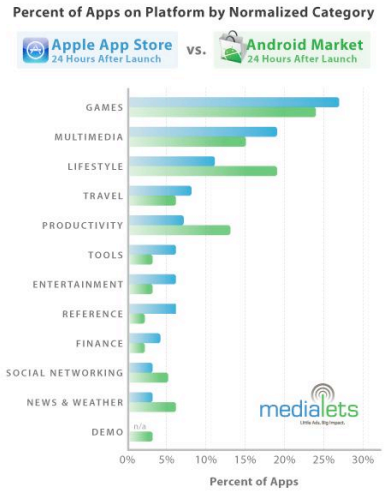

Figura 4. Comparativa en volumen y tipos de aplicaciones descargadas durante las primeras 24 horas del *Apple App Store* y el *Android Market* (vía *Android Central*[3])

[3] http://www.androidcentral.com/android-market-iphone-app-store-compared

El desarrollo de los espacios o almacenes de aplicaciones, *"Application Stores"*, es un fenómeno en sí mismo. De hecho podemos ver en la Figura 4 la explosividad del mercado con la evolución de las descargas en las primeras 24 horas de Android y iPhone[4]. Pero tiene muchas contrapartidas, como el hecho de la falta total de viralidad,: las tiendas no tienen buenos sistemas de comunicación entre los usuarios para que aprovechen este efecto de red. Tanto Android como iPhone, tienen la capacidad de intercambiar aplicaciones. Pero se encuentra limitada en origen; de momento, Google ha comentado que están analizando la manera de hacerlo, muy probablemente porque intenten ver si este intercambio se puede rentabilizar de alguna manera para no suscitar las quejas de los desarrolladores.

[4] http://www.medialets.com/blog/2008/10/23/android-market-vs-iPhone-app-store-the-first-24-hours/].

2. Metodologías ágiles

En febrero del 2001, tras una reunión celebrada en Utah[5], nace el término "ágil" aplicado al desarrollo software. El objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y responder a los cambios que pueden surgir a lo largo del proyecto. Esto pretende ser una alternativa a los procesos de desarrollo tradicionales caracterizados por su total rigidez y muy dirigidos a la documentación que se genera tras cada una de las actividades desarrolladas.

Según [1], esta nueva idea tiene dos motivaciones claras: un alto número de proyectos que se retrasan o fracasan; y la baja calidad del software que se desarrolla. La búsqueda de la solución pasa por una serie de factores: la mayor parte del esfuerzo es un proceso creativo y requiere de personas con talento, estos procesos son difícilmente planificables, modificar software es barato, las pruebas y revisión de código son la mejor forma de conseguir calidad y los fallos de comunicación son la principal fuente de fracaso.

Tras la reunión se creó *The Agile Alliance*[6] dedicada a promover el desarrollo ágil de software y ayudar a las empresas que lo adoptaran. El punto de partida fue el Manifiesto Ágil[7], documento que resume esa filosofía y que expone cuatro valores a tener en cuenta [2]:

- El individuo y las interacciones del equipo de desarrollo están por encima del proceso y las herramientas. Construir un buen equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.
- Desarrollar software que funciona más que conseguir buena documentación. No producir documentos a menos que sean necesarios de una forma inmediata. Si el software no funciona, los documentos no valen de nada.
- La colaboración con el cliente es más importante que la negociación de contratos. Tiene que haber una interacción constante entre el cliente y el equipo de desarrollo.
- La respuesta ante el cambio es más importante que el seguimiento de un plan. La planificación no debe ser estricta sino flexible y abierta, la habilidad de responder a los cambios que surjan determina el éxito o fracaso del proyecto.

Los valores anteriores inspiraron los doce principios del manifiesto. Son características que diferencian un proceso ágil del tradicional. Resumen el espíritu ágil y las metas y organización del proceso a seguir y del equipo de desarrollo.

[5] La reunión se celebró en febrero de 2001, entre el 11 y el 13 de ese mes, en Snowbird, una estación de ski muy conocido, situada en las montañas Wasatch. El manifiesto lo firmaron 17 autores, (Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas) representantes de las por entonces más populares y modernas metodologías que a partir de entonces pasaría a conocerse como «ágiles», como *eXtreme Programming*, *Crystal Clear*, *Pragmatic Programming*, DSDM o ASD.

[6] http://www.agilealliance.org/

[7] *Agile Manifesto,* http://agilemanifesto.org/

Como se señala en [3], existen cinco factores principales que afectan a la agilidad de un proceso de desarrollo software: cultura de operación (operating culture, normas de comportamiento y expectativas que gobiernan la conducta de las personas, tanto en su trabajo como en las interacciones con los demás), tamaño del equipo de desarrollo, criticidad del software (tanto en el tiempo de desarrollo como en características específicas que tenga que cumplir el software o que vengan impuestos por los elementos donde vaya a ejecutarse), competencia técnica de los desarrolladores y, por último, la estabilidad de los requerimientos.

También argumentan que un método de desarrollo de software funciona mejor cuando se aplica a situaciones con características muy específicas, a esta clase de situaciones las llama "*home ground*" (bases) del método de desarrollo de software. En la Tabla I se puede observar la comparación entre las *bases* de los métodos ágiles y las de los procesos de desarrollo por planes o "planeados" (*plan-driven*).

Área	Metodología ágil	Métodos clásicos
Desarrolladores	Colaborativos, unidos, ágiles y entendidos	Orientados al plan con una mezcla de habilidades
Clientes	Son representativos y se les entrega poder	Mezcla de niveles de aptitud
Confianza	Conocimiento tácito interpersonal	Conocimiento explícito documentado
Requerimientos	En gran parte emergentes y con rápidos cambios	Conocibles tempranamente y bastante estables
Arquitectura	Diseñada para los requerimientos actuales	Diseñada para los requerimientos actuales y los del futuro próximo
Refactorización	Económica	Costosa
Tamaño	Productos y equipo pequeños	Productos y equipos más grandes
Premium	Valor rápido	Alta seguridad

Tabla 1. Bases para métodos ágiles y planeados (Tomado de [4])

En definitiva, el desarrollo ágil de software intenta evitar los tortuosos y burocráticos caminos de las metodologías tradicionales, enfocándose en las personas y los resultados. Promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto. Desarrollando software en cortos lapsos de tiempo se minimizan los riesgos, cada una de esas unidades de tiempo se llama iteración, la cual debe durar entre una y cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Cada iteración no debe añadir demasiada funcionalidad, para justificar el lanzamiento del producto al mercado, sino que la meta debe ser conseguir una una versión funcional sin errores. Al final de cada iteración, el equipo volverá a evaluar las prioridades del proyecto.

3. Metodología ágiles para el desarrollo de software móvil

Aunque muchas metodologías ágiles han sido revisadas en la literatura[8] durante la última década, casi ninguna se ha centrado en los requerimientos tan específicos que el desarrollo móvil necesita. Como se verá a continuación, las metodologías ágiles poseen ciertas propiedades que las hacen totalmente aplicables al dominio del software en los móviles. En [5] se identifican los métodos ágiles como la solución potencial para el desarrollo de software en móviles. Se apoya en las bases (*home ground*) haciendo un análisis comparativo para probar la idoneidad de los métodos ágiles sobre el desarrollo de software para móviles. Esas características ideales y su motivación en cada caso se muestran en la Tabla 2.

Características ágiles	Motivación lógica	En el caso del desarrollo para plataformas móviles
Alta volatilidad del entorno	Debido a la alta frecuencia en el cambio que sufren los requerimientos, tendremos menos necesidad de diseño y planificación inicial y mayor necesidad de desarrollos incrementales e iterativos.	Alta incertidumbre, entornos dinámicos, cientos de nuevos terminales cada año
Equipos de desarrollo pequeños	Capacidad de reacción más rápida, trabajo basado en la compartición de la información, menos documentación.	La mayor parte de los proyectos de desarrollo software para plataformas móviles se lleva a cabo en microempresas y PyME.
Cliente identificable	Desaparecen los malentendidos.	Potencialmente, hay un número ilimitado de usuarios finales, pero los clientes son fáciles de identificar.
Entornos de desarrollo orientados a objetos	Mayoría de las herramientas de desarrollo ágil existen bajo plataformas orientadas a objetos.	Por ejemplo, Java y C++ se usan, algunos problemas en herramientas como refactorizaciones o primeros tests.
Software crítico no asegurado	Los fallos no causan gran impacto, como la pérdida de vidas. Se puede buscar mayor agilidad en el desarrollo.	La mayoría del software es para entretenimiento. Los terminales no son fiables.
Software a nivel de aplicación	Sistemas embebidos grandes requieren comunicación exhaustiva y mecanismos de verificación.	Mientras los sistemas móviles son complejos y altamente dependientes, las aplicaciones son muy autónomas
Sistemas pequeños	Menos necesidad de diseño inicial.	Las aplicaciones, aunque variables en tamaño, no suelen superar las 10.000 líneas de código.
Ciclos de desarrollo cortos	Propósito de realimentación rápida.	Periodos de desarrollo de 1 a 6 meses.

Tabla 2. Comparativa entre las características básicas o bases (*home ground*) ágiles y los rasgos observados en el desarrollo de software móvil (adaptado de [7])

[8] Entre las que han contado con más aceptación podemos destacar *Adaptive Software Development*, la familia de metodologías *Crystal*, el Método de Desarrollo de Sistemas Dinámicos *(Dynamic System Development Methodology, DSDM), eXtreme Programming (XP), Feature-Driven Development (FDD), Lean Software Development, Scrum, Agile Modeling o Pragmatic Programming, Agile Model-Driven Development, Agile Unified Process, Rational Unified Process*.

3.1 Características y requerimientos específicos del entorno móvil

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 clásico. Esto es porque el software móvil tiene que satisfacer una serie de requerimientos y condicionantes especiales ([6], [12] y [13]) que lo hace más complejo:

- **Canal** radio: consideraciones tales como la disponibilidad, las desconexiones, la variabilidad del ancho de banda, la heterogeneidad de redes o los riesgos de seguridad han de tenerse especialmente en cuenta en este entorno de comunicaciones móviles.
- **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 frecuéntemente.
- **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.
- **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.
- **Capacidades** limitadas de los terminales: aquí incluimos 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.
- **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.
- **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.
- ***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.

El diseño de sistemas de software móvil es, por tanto, bastante más complejo que el tradicional visto en los otros proyectos de desarrollo, forzando a los investigadores a reconsiderar el uso de las metodologías actuales de desarrollo de software. Como se ha visto, el uso de metodologías ágiles es el medio más apropiado para el desarrollo de tecnología en móviles, aunque las características especiales de los terminales y de las redes de telefonía móvil demandan algunos ajustes sobre las actuales metodologías ágiles.

Los investigadores han intentado mejorar este *statu quo* a través de mejoras o invenciones de metodologías. Por desgracia, estas iniciativas han sido relativamente pocas y lejanas entre ellas. La mayoría del trabajo desarrollado ha estado enfocado a aspectos del desarrollo software de bajo nivel (orientado a implementación), mientras que el alto nivel (orientado a metodologías) aun permanecía sin ser convenientemente atendido. La mayoría del desarrollo de software usa metodologías o elementos de las mismas con más de treinta años de vigencia, que se basan en la introducción de ciertas modificaciones orientadas a las necesidades específicas del desarrollo móvil.

3.2 Metodología ágil ideal para el desarrollo móvil

En [7], además de diseñar una metodología híbrida concreta que se revisará más abajo, se realiza un ejercicio de definición que muestra lo que podrían ser las características básicas de una metodología ideal para el proceso de desarrollo de softwar para plataformas móviles:

- **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 tests, 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 móvil 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 [8]) haciendo un análisis del mercado. Toda esta información mitigará las dudas y los riesgos. Procesos orientados al mercado seguirán una planificación bastante estricta a lo que se refiere a requerimientos de time-to-market. Antes los procesos estaban orientados a las actividades técnicas ahora tienen que hacer un balanceo entre las actividades orientadas a mercado y las técnicas.
- **Soporte a toda la línea de producción 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" [9]. Esto hace que el ciclo de vida sea más corto, pudiendo desarrollar una familia de productos software móviles en un solo intento,

reduciendo costes. Debe tener especial importancia la línea de producción para la mejora de la calidad del software.

- **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 ahorra costes 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 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.

3.3 Metodologías ágiles en la práctica

Ha pasado ya mucho tiempo desde el estreno de plataformas abiertas para el desarrollo de aplicaciones móviles como Java o Symbian. La disponibilidad de herramientas y la facilidad para construir y comercializar software para el móvil ha hecho que su número se dispare. A eso se ha añadido, tal como se menciona más arriba, la aparición de de las metodologías ágiles adaptadas para el desarrollo de aplicaciones para el escritorio.

Así las cosas, no es difícil entender que el desarrollo --o más bien la experimentación- con metodologías concretas se haya ido desarrollando desde dos aproximaciones bien distintas: por un lado, el enfoque puramente pragmático nacido del espíritu de desarrollo ágil más puro y, por el otro lado, el punto de vista de la ingeniería metodológica que, en general se ha quedado en el "simple" ejercicio metodológico, valga la redundancia. En esta sección presentamos dos ejemplos que implementan las teorías de desarrollo que se han descrito. Aunque se han propuesto un gran número de ellas a lo largo de los años, pensamos que estas dos nos ofrecen un ejemplo suficientemente ilustrativo.

La aproximación metodológica, más rigurosa, está representada por la propuesta de Rahimian y Ramsin en [7] y será la que presentaremos en primer lugar. El caso más pragmático estará representado por el modelo Mobile-D [10] propuesto por Pekka Abrahamsson y su equipo del VTT (*Valtion Teknillinen Tutkimuskeskus,* en inglés *Technical Research Centre of* Finland) en Finlandia que lideran una corriente muy importante de desarrollo ágil [11] muy centrada en las plataformas móviles.

3.3.1. Un modelo híbrido para el desarrollo ágil

La aproximación metodológica de [7] se apoya en una combinación del desarrollo adaptativo de software (*Adaptive Software Development, ASD*) y el diseño de nuevos productos (*New Product Development, NPD*). Esto supone una decisión crítica para decantarse más del lado del desarrollo de productos que del lado de la gestión de proyectos, lo cual quiere decir que una de las característica más sensibles, desde el punto de vista metodológico, para la consolidación de una metodología propia de un entorno móvil, es la presión de los plazos para llegar al mercado, un mercado volátil y altamente dinámico.

Figura 5. Ciclo de vida clásico para un proceso de desarrollo software [7]

Para empezar, a pesar de ser uno de los pocos intentos metodológicos serios recientes, debemos mostrarnos críticos con la propuesta que encontramos en [7]. Se apoya en un proceso iterativo de diseño híbrido de metodologías (Hybrid Methodology Design, HMD) y parte del ciclo de vida tradicional representado en la figura 5.

En la primera iteración se divide la fase de análisis con la intención de mitigar riesgos de desarrollo; de la misma forma, el diseño también se segmenta para introducir algo de diseño basado en arquitectura. La implementación y las pruebas sin embargo se fusionan introduciendo conceptos de desarrollo orientado a pruebas (*Test-Driven Development, TDD*[9]). Aparece además una fase de comercialización, incidiendo en el sesgo hacia el desarrollo de producto que se imponen en el escenario del desarrollo de aplicaciones para plataformas móviles. Desde el punto de vista metodológico, los autores afirman haberse apoyado en metamodelos como SPEM (*Software Processes Engineering Metamodel*, soportado por el entorno de desarrollo de Eclipse, por ejemplo[10]) y OPF, (*Open Processes Framework*[11]), así como en conceptos genéricos de ciclos de vida orientados a objetos como OOSP (*Object-Oriented Software Processes*).

Figura 6. Primera iteración en el diseño híbrido: instanciación de análisis y diseño [7]

Tras la instanciación realizada en la primera iteración metodológica, la segunda realiza una integración de ciertas partes de los modelos NPD, añadiendo la generación de

[9] Se trata de un concepto muy ligado a la idea de "prueba primero" de la filosofía del *eXtreme Programming*, http://en.wikipedia.org/wiki/Test-driven_development
[10] http://www.omg.org/technology/documents/formal/spem.htm
[11] http://www.opfro.org/

ideas en el inicio del ciclo y una prueba de mercado antes de lanzar la fase de comercialización.

Figura 7. Segunda iteración en el diseño híbrido: integración hacia el desarrollo de producto [7]

La tercera iteración integra directamente el "motor de desarrollo" de los métodos de desarrollo adaptativo (ASD) muy orientados al aseguramiento de la calidad en los procesos de desarrollo.

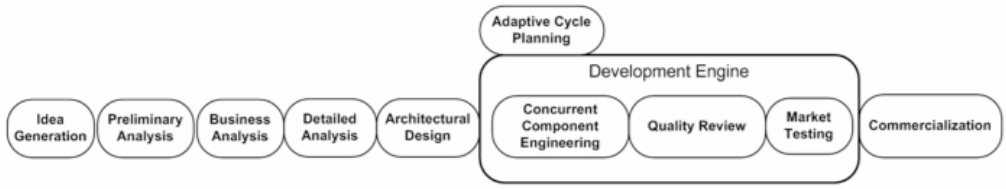

Figura 8. Tercera iteración en el diseño híbrido: motor de desarrollo [7]

Pensando en aquella propiedad "ideal" de disponer de la arquitectura física en una fase temprana del proceso, en la cuarta iteración se añaden elementos de prototipado; se refina, además, la fase de iniciación del proyecto, sobre la base del mismo elemento de los procesos adaptativos.

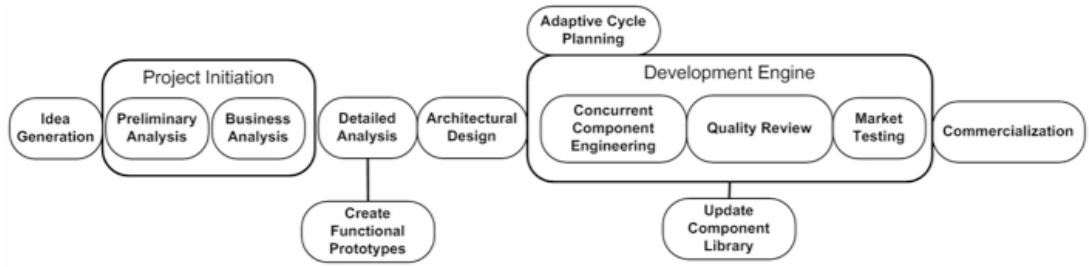

Figura 9. Cuarta iteración en el diseño híbrido: prototipado [7]

3.3.2. Mobile-D

Se podría pensar que Mobile-D [14,17] es una creación un tanto antiguo, ya que se desarrolló como parte de un proyecto finlandés, ICAROS, allá por 2004. Sin embargo, creemos que vale la pena mencionarlo por dos razones. Primera: fue creado mediante un proyecto de cooperación muy estrecha con la industria. El grueso del trabajo fue realizado por los investigadores del VTT. Aun así la metodología de diseño se elaboró con una participación importante de las empresas de TI finlandesas. Tal como se puede ver en los experimentos que se han documentado (véase 3.3.3) esto consiguió que la investigación llevada a cabo no se alejara demasiado de las reglas de desarrollo de las aplicaciones comerciales

Segundo, Mobile-D es una mezcla de muchas técnicas. Tal como se verá luego, los investigadores no dudaron en echar mano de las prácticas habituales de desarrollo software. Pero, al mismo tiempo, consiguieron crear una contribución original para el nuevo escenario del desarrollo de aplicaciones para sistemas móviles. Creemos que este ejemplo ilustra perfectamente cómo se pueden usar conjuntamente diferentes metodologías y técnicas en el contexto del desarrollo ágil.

En las siguientes subsecciones hablamos de la motivación del proyecto (sección 3.3.1), los fundamentos de la metodología (sección 3.3.2) y los experimentos prácticos que se llevaron a cabo durante la investigación (sección 3.3.3). Si fuera necesaria una documentación más extensa, el lector se puede dirigir a la documentación completa de esta metodología diaponible en el sitio web del proyecto [15].

3.3.2.1. Motivación

La metodología se creo en un periodo de intenso crecimiento en el terreno de las aplicaciones móviles. Por tanto, en ese momento no existían demasiados principios de desarrollo a los que acudir. Los autores de Mobile-D apuntan a la necesidad de disponer de un ciclo de desarrollo muy rápido para equipos muy pequeños. De acuerdo con sus suposiciones, Mobile-D está pensado para grupos de no más de 10 desarrolladores colaborando en un mismo espacio físico. Si trabajan con el ciclo de desarrollo propuesto (sección 3.3.2), los proyectos deberían finalizar con el lanzamiento de productos completamente funcionales en menos de diez semanas.

3.3.2.2. Principios básicos

La aproximación de Mobile-D se ha apoyado en muchas otras soluciones bien conocidas y consolidadas: *eXtreme Programming* (XP) [18], *Crystal methodologies* [19] y *Rational Unified Process* (RUP) [21]. Los principios de programación extrema se han reutilizado en lo que se refiere a las prácticas de desarrollo, las metodologías *Crystal* proporcionaron un input muy valiosos en términos de la escalabilidad de los métodos y el RUP es la base para el diseño completo del ciclo de vida.

El ciclo del proyecto se divide en cinco fases: exploración, inicialización, productización, estabilización y prueba del sistema (figura 9). En general, todas las fases (con la excepción de la primera fase exploratoria) contienen tres días de desarrollo distintos: planificación, trabajo y liberación. Se añadirán días para acciones adicionales en casos particulares (se necesitarán días para la preparación del proyecto en la fase de inicialización, por ejemplo).

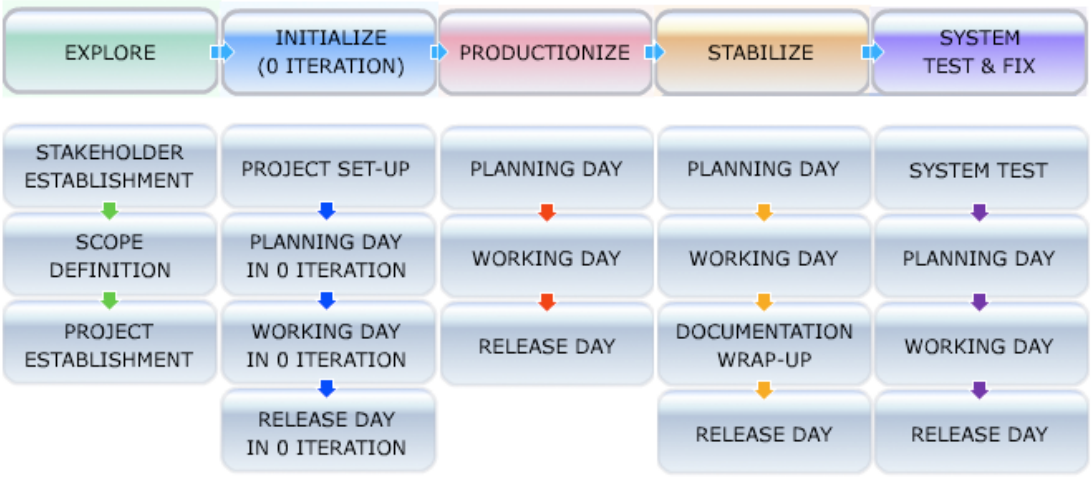

Figura 10. Ciclo de desarrollo Mobile-D (reproducido de [15]).

La fase de exploración, siendo ligeramente diferente del resto del proceso de producción, se dedica al establecimiento de un plan de proyecto y los conceptos básicos. por lo tanto, se puede separar del ciclo principal de desarrollo (aunque no debería obviarse). Los autores de la metodología ponen además especial atención a la participación de los clientes en esta fase.

Durante la fase de inicialización, los desarrolladores preparan e identifican todos los recursos necesarios. Se preparan los planes para las siguientes fases y se establece el entorno técnico (incluyendo el entrenamiento del equipo de desarrollo). Los autores de Mobile-D afirman que su contribución al desarrollo ágil se centra fundamentalmente en esta fase, en la investigación de la línea arquitectónica. Esta acción se lleva a cabo durante el día de planificación. Los desarrolladores analizan el conocimiento y los patrones arquitectónicos utilizados en la empresa (extraidos de proyectos anteriores) y los relacionan con el proyecto actual. Se agregan las observaciones, se identifican similitudes y se extraen soluciones viables para su aplicación en el proyecto. Finalmente, la metodología también contempla algunas funcionalidades nucleares que se desarrollan en esta fase, durante el día de trabajo.

En la fase de "productización" se repite la programación de tres días (planificación-trabajo-liberación) se repite iterativamente hasta implementar todas las funcionalidades. Primero se planifica la iteración de trabajo en términos de requisitos y tareas a realizar. Se preparan las pruebas de la iteración de antemano (de ahí el nombre de esta técnica de *Test-Driven Development, TDD*). Las tareas se llevarána a cabo durante el día de trabajo,

desarrollando e integrando el código con los repositorios existentes. Durante el último día se lleva a cabo la integración del sistema (en caso de que estuvieran trabajando varios equipos de forma independiente) seguida de las pruebas de aceptación.

En la fase de estabilización, se llevan a cabo las últimas acciones de integración para asegurar que el sistema completo funciona correctamente. Esta será la fase más importante en los proyecto multi-equipo con diferentes subsistemas desarrollados por equipos distintos. En esta fase, los desarrolladores realizarán tareas similares a las que debían desarrollar en la fase de "productización", aunque en este caso todo el esfuerzo se dirige a la integración del sistema. Adicionalmente se puede considerar en esta fase la producción de documentación.

La última fase (prueba y reparación del sistema) tiene como meta la disponibilidad de una versión estable y plenamente funcional del sistema. El producto terminado e integrado se prueba con los requisitos de cliente y se eliminan todos los defectos encontrados.

3.3.3. Enfoque pragmático

Los autores de esta metodología dicen haberlo probado obteniendo una certificación CMMI[12] de nivel 2. Esto parece ser una ventaja comparativa importante frente a otras metodologías, puesto que la contratación de empresas para la externalización del desarrollo software se rige por la auditoría de los ciclos y técnicas de desarrollo que utilizan (y CMMI es una de las métricas de aseguramiento de calidad más aceptadas en el sector).

Adicionalmente, sus creadores han introducido Mobile-D en numerosos proyectos de desarrollo con clientes reales. La base inicial de 4 casos de estudio [20] se ha desarrollado durante años y afirman sus autores que los cilos de desarrollo se han actualizado y mejorado a partir de la experiencia obtenida [16].

[12] http://www.sei.cmu.edu/cmmi/general/index.html

4. Introducción al ANDROID

Android es una solución completa de software de código libre para teléfonos y dispositivos móviles. Es un paquete que engloba un sistema operativo, un "runtime" de ejecución basado en Java, un conjunto de librerías de bajo y medio nivel y un conjunto inicial de aplicaciones destinadas al usuario final (todas ellas desarrolladas en Java). Android se distribuye bajo una licencia libre permisiva (Apache) que permite la integración con soluciones de código propietario.

Android surge como resultado de la Open Handset Alliance[13] un consorcio de 48 empresas distribuidas por todo el mundo con intereses diversos en la telefonía móvil y un compromiso de comercializar dispositivos móviles con este sistema operativo. El desarrollo viene de la avalado principalmente por Google (tras la compra de Android Inc. en 2005) y entre las compañías encontramos compañías de software (Ebay, LivingImage...), operadores (Telefónica, Vodafone, T-Mobile...), fabricantes de móviles (Motorola, Samsumg, acer, LG, HTC...) o fabricantes de Hardware (nVidia, Intel o Texas Instruments).

4.1. Arquitectura

Android presenta una arquitectura basada en 4 niveles (figura 10), que detallamos a continuación por orden ascendente:

- **Un kernel linux** versión 2.6 que sirve como base de la pila de software y se encarga de las funciones más básicas del sistema: gestión de drivers, seguridad, comunicaciones, etc.
- **Una capa de bibliotecas de bajo nivel** en C y C++, como SQLite para persistencia de datos; OpenGL ES para gestión de gráficos 3D, con aceleración 3D opcional y Webkit como navegador web embebido y motor de renderizado HTML.
- **Un framework para el desarrollo de aplicaciones**, dividido en subsistemas para gestión del sistema como el "Administrador de paquetes", el "Administrador de telefonía" (para la gestion del hardware del teléfono anfitrión) o el acceso a APIs sofisticadas de geolocalización o mensajería XMPP. Los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar el reuso de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Éste mismo mecanismo permite que los componentes sean reemplazados por el usuario.También incluye un sistema de vistas para manejar el interfaz de usuario de las aplicaciones, que incluyendo posibilidad de visualización de mapas o renderizado html directamente en el interfaz gráfico de la aplicación.
- **Aplicaciones:** Las aplicaciones base incluyen un teléfono, cliente de email, programa de envío de SMS, calendario, mapas, navegador, contactos... que pueden a su vez ser usados por otras aplicaciones.

[13] http://www.openhandsetalliance.com

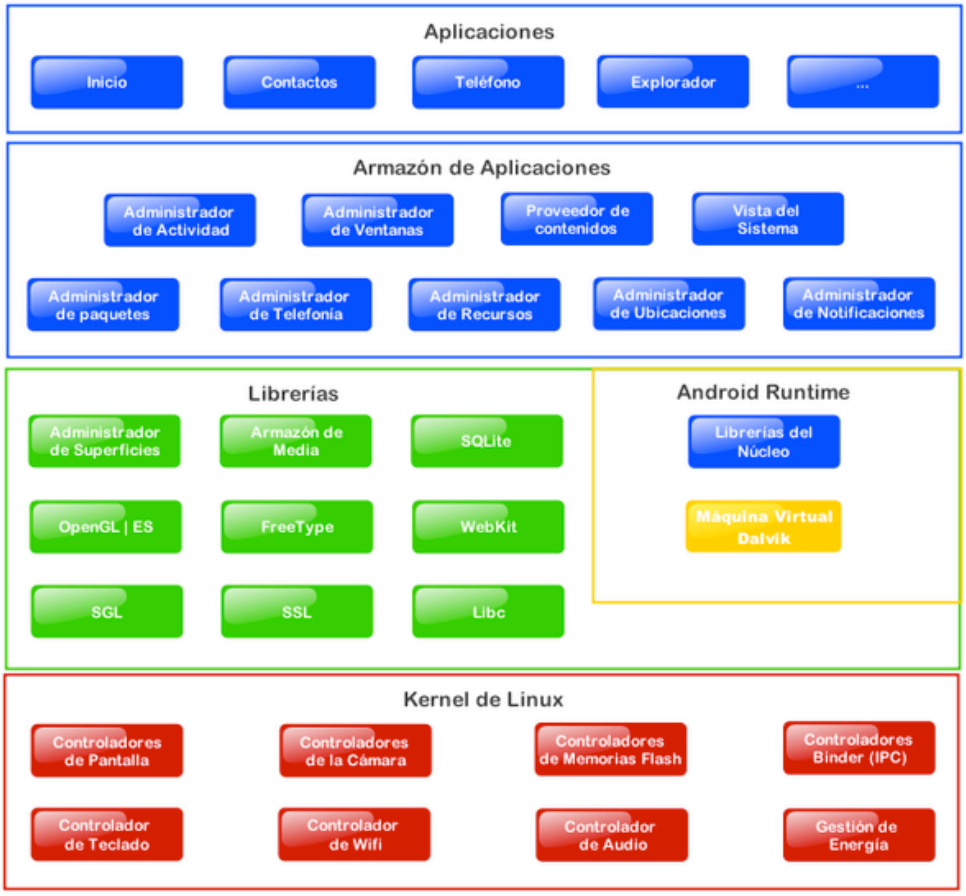

Figura 11. Arquitectura de Android (reproducida de Wikipedia[14])

Las aplicaciones Android están programadas en Java, pero no corriendo sobre Java ME, sino sobre Dalvik, una máquina virtual Java desarrollada por Google y optimizada para dispositivos empotrados. La creación de una VM propia es un movimiento estratégico que permite a Google evitar conflictos con Sun por la licencia de la máquina virtual, así como asegurarse el poder innovar y modificar ésta sin tener que batallar dentro del JCP. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalkiv ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente.

4.2. Desarrollo en ANDROID

La comunidad de desarrollo de Android tiene como objetivo hacer el desarrollo de aplicaciones para Android muy sencillo y accesible al mayor número de sistemas posible.

[14] http://es.wikipedia.org/wiki/Android

Por ello, todas las herramientas, entornos, demos e instrucciones que se ofrecen son gratuitas y se pueden descargar de su página web, http://developer.android.com.

4.2.1. Entorno de Desarrollo

Android ofrece un plugin para Eclipse que extiende la funcionalidad de éste y facilita el desarrollo de aplicaciones para Android. Además, ofrece las herramientas que utiliza este plugin como scripts de ant para que puedan ser utilizados también desde otros entornos como Netbeans o Intellij IDEA[15]. Entre las funcionalidades de este plugin se encuentra:

- Emulador de Android. Permite elegir entre distintos terminales móviles y la versión del sistema operativo.
- El acceso a herramientas de desarrollo de Android como tomar capaturas de pantalla, la redirección de puertos, la posibilidad de depurar con puntos de parada o ver el estado de las hebras y los procesos corriendo en el sistema.)
- Asistentes para la creación rápida de aplicaciones Android
- Editores de código para los distintos archivos de configuración (XML) que facilitan su comprensión y desarrollo
- Interfaces gráficas que permiten el desarrollo de componentes visualmente.

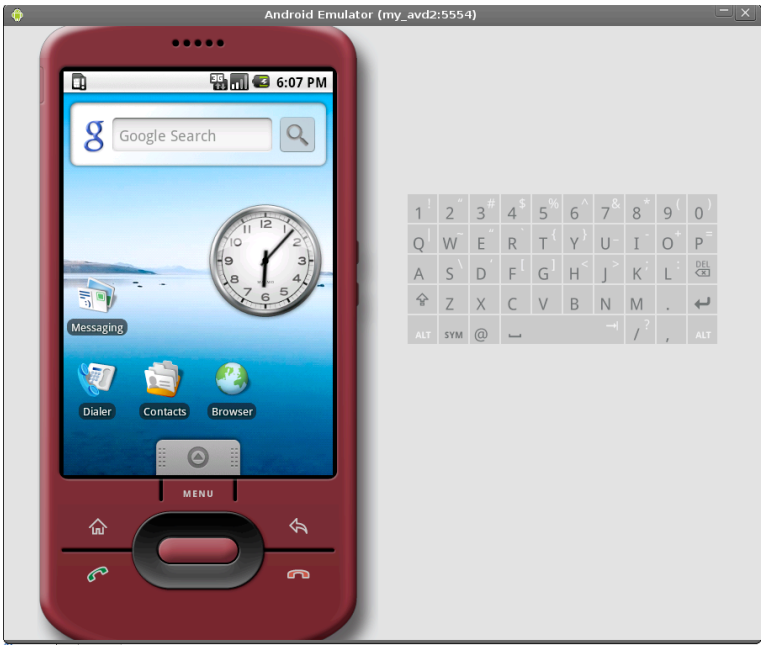

Figura 11. Vista del emulador de Android

[15] http://www.jetbrains.com/idea/

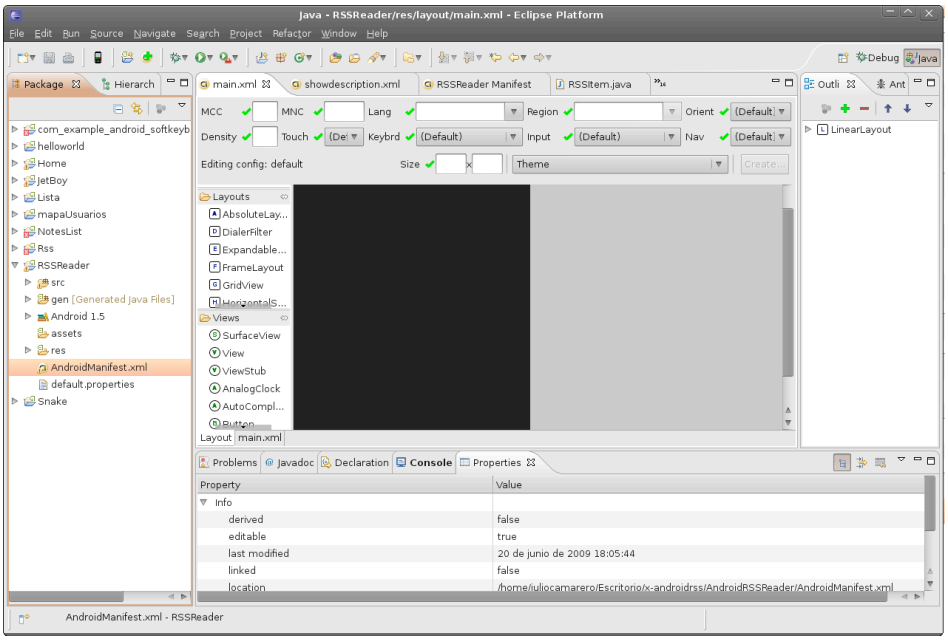

Figura 12. Vista del entorno de desarrollo para Android

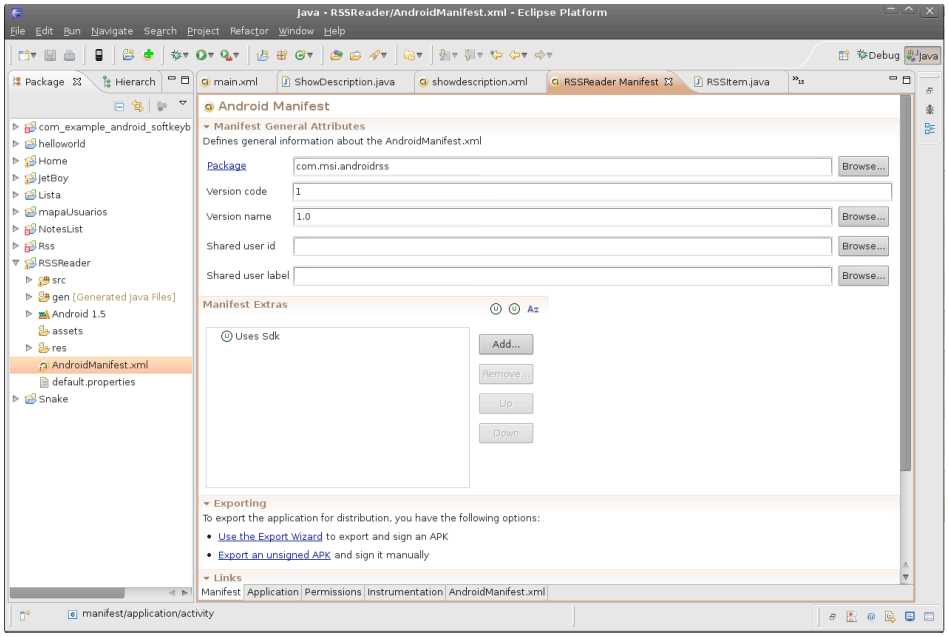

Figura 13. Vista de la plataforma Eclipse

4.2.2. Modelado de las aplicaciones

Cada aplicación de Android corre su propio proceso de Linux, y a su vez, cada uno de estos procesos corre su propia Máquina virtual Java. (Se aísla la ejecución entre aplicaciones). Para facilitar la reutilización de código y agilizar el proceso de desarrollo, las aplicaciones Android se basan en componentes. Por ejemplo, si una aplicación quiere utilizar una lista con scroll puede utilizar una aplicación que sea simplemente esa lista para hacer la suya sin tener que duplicar el código. Los componentes pueden ser de 4 tipos:

- **Actividades.** Son interfaces visuales esperando alguna acción del usuario. (Por ejemplo, una aplicación para el envío de mensajes podría tener una actividad que fuera la lista de contactos de la que el usuario elige a uno, otra que fuera el editor de textos para que el usuario escriba el mensaje y otra con parámetros de configuración de la aplicación). Una aplicación puede tener una actividad o más, y desde una actividad se puede invocar a otras y volver nuevamente a la original. Todas las actividades extienden la clase Activity. El contenido visual de cada actividad lo proporcionan una serie de objetos derivados de la clase View. Android proporciona muchos de estos objetos prediseñados como botones, selectores, menús...
- **Servicios**. Los servicios no tienen interfaz gráfica. Un ejemplo sería la reproducción de una canción. Para una aplicación reproductora podríamos tener varias actividades para mostrar listas de canciones o un reproductor con botones, pero el usuario esperará que la canción siga sonando aún al salir de la aplicación (terminar la actividad), por lo que esta aplicación deberá controlar un servicio para que se reproduzca la música. Cualquier servicio extiende la clase Service.
- **Receptores de Eventos.** Estos componentes simplemente están escuchando a que se produzcan determinados eventos (batería baja, cambio del idioma del dispositivo, la descarga de una imagen nueva...) Cualquier aplicación puede tener tantos receptores para tantos eventos como quiera, cada uno debe extender la clase BroadCastReceiver. Un ejemplo sería una galería de imágenes que indexa en la aplicación cualquier imagen que el usuario se descargue mediante el navegador un otra aplicación.
- **Proveedores de contenidos**. Permite que una aplicación ponga ciertos datos a disposición de otras aplicaciones. Por ejemplo, una grabadora de sonidos puede compartir esos datos con un reproductor de música. Estos datos pueden almacenarse en el sistema de ficheros o en base de datos. Para proveer contenidos, se debe extender la clase ContentProvider.

Además, todas las aplicaciones Android deben tener un fichero AndroidManifest.xml donde se definan todos los componentes de la aplicación así como los permisos que requiere, o los recursos y librearías que utiliza. A continuación se muestra un ejemplo de este fichero que utiliza dos actividades y requiere permiso para acceder a Internet.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" android:versionName="1.0"
package="com.msi.androidrss">
    <application android:icon="@drawable/icon">
        <uses-permission android:name="android.permission.INTERNET" />
        <activity android:name=".RSSReader"
android:label="@string/app_name" />
        <activity android:name=".ShowDescription" />
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
```

4.2.3. Caso Práctico. Lector de RSS

Para poner en práctica las metodologías ágiles en el desarrollo de una aplicación Android, decidimos hacer una aplicación que mostrase las entradas de un blog que exportase sus entradas en formato RSS 2.0. Para este ejemplo, vamos a realizar únicamente la primera iteración que proponen estas metodologías, por lo que nuestra aplicación será únicamente una primera versión como resultado de una vuelta al ciclo de vida. El único objetivo con esta interacción es conseguir una aplicación que funciona, aunque no tenga todas las funcionalidades requeridas. Así, por ejemplo, en esta primera iteración se han aprendido los conceptos básicos del desarrollo en Android y se ha desarrollado una aplicación que permite leer los feeds de un sitio usando RSS 2.0. Una vez terminada la aplicación, se han identificado diversas funcionalidades que podrían ser añadidas o mejoras al producto que realizar en siguientes iteraciones como podrían ser: almacenar los feeds en la memoria del teléfono, poner un botón de actualizar los feeds, leer otros formatos con RSS 1.0 o Atom.

El ejemplo desarrollado se compone de dos actividades, una que muestra el listado de feeds (extendiendo OnItemClikListener que a su vez extiende a ListActivity) y otra que muestra un feed completo (que simplemente extiende de Activity). Cada actividad tiene su vista asociada a través de un fichero xml. A continuación se muestra a modo de ejemplo como se define la vista de un feed que se compone de una caja de texto (con el texto de la entrada) y un botón de volver.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical">
       <TextView  android:text="story goes here ...."
android:id="@+id/storybox" />
       <Button android:text="Back" android:id="@+id/back" />

</LinearLayout>
```

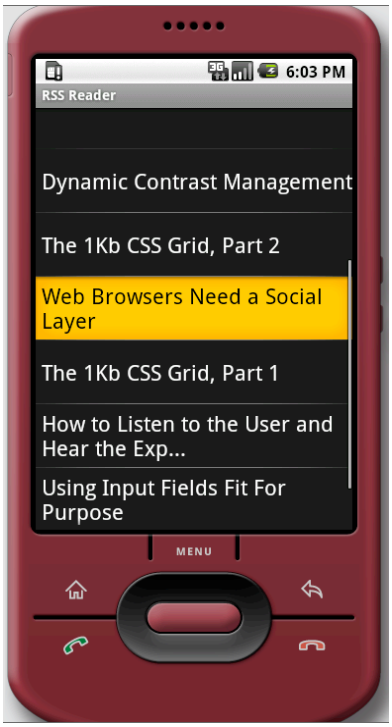

Figura 14. Vista del lector de feeds RSS en el amulador

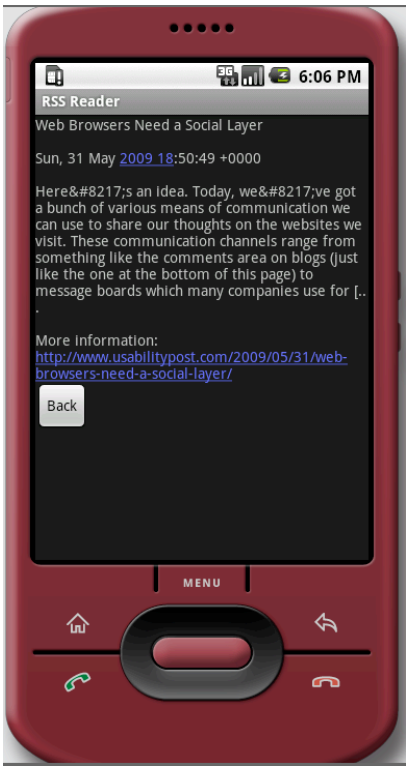

Figura 15. Vista de una noticia seleccionada

4.2.4. Publicación de las aplicaciones

Las aplicaciones para Android pueden ser distribuídas y comercializadas a través del *Android Market*[16], un servicio alojado por Android que facilita a los usuarios encontrar y descargar aplicaciones para Android y a los desarrolladores publicarlas. El sistema se encarga además de gestionar las distintas versiones de una aplicación y su compatibilidad con las distintas versiones del sistema operativo. Cuando se creó Android se decidió que todas las aplicaciones deberían ser gratuitas, lo que frenó mucho su desarrollo. Visto el éxito del iPhone, hace varios meses se decidió dar cabida también a las aplicaciones de pago. Para subir tu aplicación a Android Market simplemente hace falta disponer de una cuenta de Google, el proceso es sencillo y gratuito.

[16] http://www.android.com/market/

5. Introducción al iPhone

El iPhone de Apple es un *smartphone* (teléfono inteligente) diseñado y distribuido por Apple. Desde el primer momento entre las características del dispositivo destacaba la interfaz casi totalmente basada en la pantalla táctil siendo el interfaz de usuario por hardware mínimo. Entre otras características permite usarlo de cámara, como reproductor multimedia y conectarse a Internet de manera bastante parecida a como lo hace un PC.

Otra de los puntos que diferencian al iPhone (sobre todo en los primeros monentos) del resto de productos similares es la existencia de la App Store. En esta tienda los usuarios pueden adquirir directamente las aplicaciones para su terminal, de hecho, es la única manera de obtener software nativo para el iPhone fuera de las aplicaciones incluidas por defecto. Por tanto, los desarrolladores están obligados a introducir sus trabajos en este sistema, previo pago de una módica suma. Además Apple obtiene un 30% de los beneficios de cada venta y tiene control absoluto sobre la tienda, pudiendo añadir o quitar aplicaciones a su gusto. Esto por un lado es un problema para los desarrolladores, pero parece que ha funcionado con los usuarios que no tienen que preocuparse de la procedencia de lo que instalan ya que, idealmente, todo es fiable.

5.1. iPhone OS

El sistema operativo utilizado por el iPhone se llama, desde la publicación del primer SDK, OS X iPhone o, más comúnmente iPhone OS. Como el primer nombre indica, está fuertemente basado en Mac OS X, la línea de sistemas operativos que desarrolla Apple y que se incluye en todos los ordenadores Macintosh desde 2002. Al igual que su "padre", iPhone OS deriva de la fundación Darwin que, a diferencia de Mac OS X, es de código abierto.

Partiendo de esta base, el sistema operativo varía en gran medida de sus antecesores debido a las características propias del iPhone como su interfaz táctil y la menor cantidad de memoria disponible así como su arquitectura de procesador (basado en ARM en lugar de x86) lo que provoca que los programas desarrollados para Mac OS X deban ser adaptados para la nueva plataforma. Aún así, ese origen común permite, como veremos más adelante, que tanto las herramientas como el proceso que se sigue en un desarrollo sean bastante similares en los dos casos.

5.2. iPhone SDK

Con la salida del iPhone en 2007, no se proveía un SDK propiamente dicho para el desarrollo de aplicaciones nativas, Apple se guardaba el monopolio absoluto del software que podía correr en el teléfono.

Como compensación se ofrecía la posibilidad de desarrollar *web apps* las cuales corrían en Safari, el navegador propio del iPhone. Estas "aplicaciones web" estaban escritas, como es de esperar, en HTML y JavaScript ofreciendo Apple los recursos necesarios (imágenes, animaciones) para que éstas tuvieran una apariencia similar a la de

las aplicaciones nativas así como un mecanismo para colocarlas en el menú principal. No obstante, al no ser posible copiarlas al iPhone, era necesario en todos los casos tener un servidor web que sirviera las páginas a los clientes.

Hasta Marzo de 2008 no se publicó un SDK propiamente dicho para el desarrollo de aplicaciones para el iPhone. Para dar una visión general del SDK es conveniente separarlo en dos partes: las herramientas proporcionadas para el desarrollo y la arquitectura.

5.3. Herramientas

- ***XCode:*** Es la principal herramienta del SDK. Es la aplicación en la que se realiza la mayor parte del desarrollo. Nos permite administrar el proyecto e ir añadiendo código. Además incluye un depurador gráfico bastante potente que corre sobre gdb, el debugger clásico.
- ***Instruments:*** Permite medir distintas características de rendimiento y de uso de memoria de la aplicación desarrollada mostrando gráficas que se actualizan en tiempo real. Como se ha dicho anteriormente, al desarrollar para plataformas móviles, los desarrolladores más acostumbrados a sistemas con mayor cantidad de memoria deben tener muy en cuenta las limitaciones del entorno para el que están trabajando. Herramientas de este tipo facilitan esa labor.
- ***DashCode:*** Destinada al desarrollo de aplicaciones web para el iPhone. Su principal finalidad es facilitar la tarea de escribir páginas que sean compatibles para la versión de Safari incluida en el dispositivo. Incluye una serie de plantillas de ejemplo y de imágenes así como código JavaScript que ayudan a la hora de que la apariencia de la página sea consistente con el diseño general del interfaz de usuario del sistema operativo. Adicionalmente, incluye herramientas para depurar el código.
- ***iPhone Simulator:*** Como su nombre indica, es un simulador del iPhone que permite probar las aplicaciones desarrolladas si necesidad de pasarlas a un iPhone (y pagar la tarifa correspondiente). Implementa el API completa del iPhone y, además, incluye Safari con lo que es posible probar tanto programas nativos como las aplicaciones web desarrolladas con DashCode. Es importante resaltar que se trata de un simulador y no un emulador, es decir, implementa el API sobre la plataforma x86 existente en los ordenadores Mac actuales y no intenta emular el hardware del iPhone. Este detalle es importante sobre todo si atendemos a la velocidad ya que la máquina sobre la que corre el simulador va a ser siempre más rápida.

5.4. Arquitectura

El desarrollo en el iPhone se basa en el uso de una variedad de frameworks y tecnologías sobre las que el programador se irá apoyando para construir su nueva aplicación. El SDK permite el acceso a estos recursos mediante una arquitectura por capas partiendo de las funciones mas básicas y cercanas a la máquina y terminando por las tareas más sofisticadas como los efectos gráficos.

A continuación se detalla un poco más cada una de estas capas:

- **Core OS:** Es la capa inferior y es, por lo tanto, la que más cerca está del sistema operativo y de la máquina. Permite acceso a utilidades como: Hebras (POSIX), Redes (Sockets de BSD), entrada y salida estándar, sistema de ficheros, manejo de memoria…
- **Core Services:** Principalmente compuesta por *Core Foundation* y *CFNetwork*. *Core Foundation* es una serie de librerías construidas sobre la capa anterior que facilitan el manejo de colecciones, fechas URLs, Streams y mucho más. *CFNetwork* se centra en el uso de protocolos tales como http, FTP, bonjour, etc. Además en Core Services se ofrecen frameworks que ayudan en temas de seguridad así como manejo de bases de datos SQLite para las aplicaciones y XML.
- **Media:** Contiene todo lo que tenga que ver con audio, video y gráficos. Ofrece varias tecnologías para dibujar en 2d y 3d con mayor o menor nivel de abstracción.
- **Cocoa Touch:** Es principalmente la capa del interfaz de usuario. Es la versión para iPhone del Cocoa de Mac Os X con lo que conserva muchas de sus características como el modelo vista controlador. Además ofrece la posibilidad de integrar aplicaciones entre sí mediante mensajes.

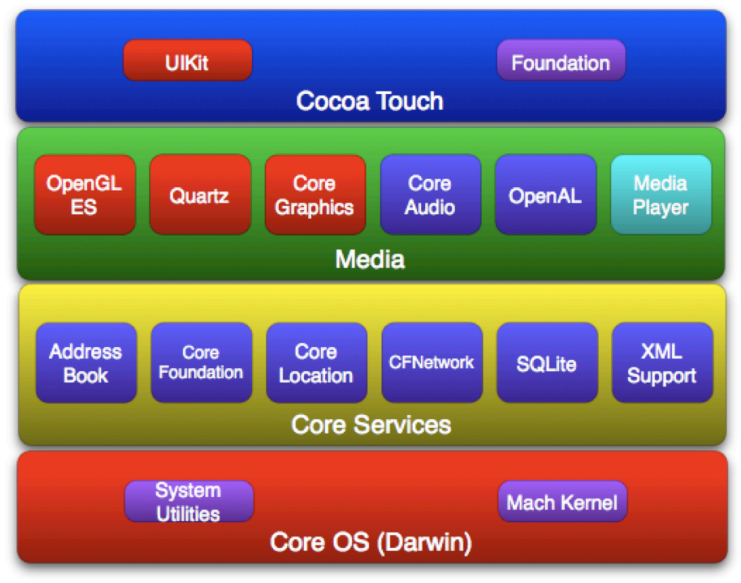

Figura 16. Arquitectura del iPhone.

6. Conclusiones y trabajos futuros

Si bien podemos, sobre el papel, establecer algún tipo de paralelismo entre el enfoque metodológico del modelo híbrido y la metodología ágil del Mobile-D del VTT, no está claro que podamos evaluar la bondad de una implementación de las mismas si no se dispone de un escenario donde las presiones para llegar al mercado y la complejidad de unos requisitos cambiantes nos permitan diferenciarlas con algún criterio o alguna métrica de cierta rigurosidad.

No resulta sencillo definir un escenario de desarrollo donde puedan implementarse ambos métodos en paralelo para seleccionar un conjunto de métricas del rendimiento de estos métodos de desarrollo ágil en entornos móviles. Por un lado, las métricas cuantitativas se suelen relacionar con los procesos tradicionales de desarrollo; y, por otro lado, la propia filosofía ágil de desarrollo pone el foco en aspectos como el "progreso" del proyecto, es decir la velocidad o el software real liberado (*Running Tested Features, RTF*), la moral de los equipos de desarrollo y, sobre todo, en el valor real de negocio ganado (*Earned Business Value, EBV*)[17]. Esa situación hace que la literatura nos ofrezca muchos trabajos centrados en relacionar el desarrollo ágil con metodologías de gestión como Value-Based IT Management (VBIM)[18].

Si comparamos, sobre el papel, ambas aproximaciones, lo primero que hay que destacar es la orientación de Mobile-D a una métrica tipo RTF, es decir a la disponibilidad en cada iteración de software funcional "corriendo", precisamente por la forma en que se ha implementado cada fase como slots verticales dentro de un proceso horizontal. El énfasis del modelo híbrido, sin embargo, está más orientado quizás al valor añadido de negocio, integrando desde la primera iteración la prueba de mercado con la instanciación de un ciclo adaptativo propio de la metodología ASD; aunque Mobile-D implica al cliente en la fase de exploración con un criterio similar de EBV, mientras que la inicialización en el modelo híbrido se centra en la reducción de riesgos.

Si tenemos que valorar un punto sobresaliente de cada uno de los dos ejemplos presentados, sin duda el modelo híbrido incorpora elementos directamente relacionados con la gestión de operaciones y la teoría de sistemas complejos que ayudan, sin duda, a conferirle cierta rigurosidad en su planteamiento; aunque eso no nos proporcione mayor satisfacción que el pragmatismo de los casos de aplicación de Mobile-D en el mundo real. En el caso de este último, su mayor aportación no es ni siquiera esa, sino quizás el modelo utilizado para su elaboración, que incorporaba a los desarrolladores en la propia labor de investigación metodológica del VTT. Aun así, como crítica extensible a ambos planteamientos, debemos constatar el hecho de que, aun estando ambos dirigidos a entornos de alta volatilidad y dinamismo en los que se ha establecido como elemento clave el talento y la organización de los pequeños equipos de desarrollo, se sigue acudiendo en los

[17] Deborah Hartmann, Robin Dymond, Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value, http://www.innovel.net/wp-content/uploads/2007/07/appropriateagilemeasurementagilemetrics.pdf

[18] Value-Based Management and Agile Methods, John Favaro, Proceedings of the 4th International Conference on XP and Agile Methods, http://www.favaro.net/john/home/publications/vbmams.pdf

ejercicios metodológicos a herramientas y elementos más cercanos a la gestión de operaciones que a la gestión del talento. Una línea de investigación, en ese sentido, con cierto recorrido ,podría ser la elaboración de una aproximación al problema metodológico de la gestión de entornos de desarrollo de software ágiles desde una óptica del '*management*' moderno, centrado en la gestión del talento y el liderazgo.

Desde el punto de vista práctico resulta de utilidad poner en perspectiva las dos plataformas sobre las que se ha realizado la demostración, bajo los términos que en [7] se referían como específicos del desarrollo ágil para sistemas móviles. Es lo que hacemos con la siguiente tabla que, no pretendiendo en absoluto ser exhaustiva, sí quiere ser un análisis preliminar meramente descriptivo que nos ayude en una potencial implementación de un experimento de ciclo de desarrollo ágil sobre las mismas.

Área	Android	iPhone
Análisis preliminar		
Número de móviles existentes con cada uno	HTC Dream, HTC Magic, Samsung I7500 (muchos por llegar en los próximos meses: LG, Sony Ericsson, Motorola, Acer…)	3: iPhone, iPhone 3G y iPhone 3G S
Dinero que mueve cada mercado	Se espera haber vendido 8 millones de terminales en 2009	Como ejemplo: Un millón de Iphone 3G S entre el Viernes y el Martes.
Diseño de la arquitectura		
Si existen aplicaciones anteriores a la nuestra o componentes que podemos reutilizar	Se permite la reutilización otros componentes y otras aplicaciones.	Fácil con aplicaciones propias pero existe muy poco código libre disponible.
Limitaciones de la plataforma: licencia, lenguaje de programación…	Lenguage Java. Licencia de la plataforma: Apache. Las aplicaciones pueden tener sus propias licencias.	Lenguaje Objective-C, C. Licencia: Para publicar aplicaciones es necesario pagar a Apple y obtener su aprobación
Implementación y Test		
Entornos de desarrollo, su precio, calidad….	Plugin para Eclipse y herramientas para otros IDES. Todo gratuito, calidad razonable.	Varias aplicaciones suministradas por apple. Gratuitas y muy maduras.
Experiencia con el IDE	Bastante satisfactoria. Todo muy bien integrado y funcionando bien (depuración muy bien hecha).	Muy buena. El mejor entorno para C y derivados que he probado.
Tests, emuladores…	Emulador muy bueno.	Simulador, no emulador e Instruments para recabar datos de iPhones reales.
Comercialización		
Licencias, costes	Gratuíto para el desarrollador. Sólo requiere una cuenta Google.	Para publicar es necesario tener una subscripción de Apple developer Connection (US$ 99 particulares, US$ 299 empresas)
Éxito de aplicaciones existentes	Retrasado respecto al iPhone pero con igual evolución	Apple dice que mas de 2.000 millones de descargas

Tabla 3. Comparativa Android vs. iPhone (elaboración propia)

7. Referencias

[1] Ferrer, J., Presentación sobre metodologías ágiles. Disponible en http://libresoft.dat.escet.urjc.es/html/downloads/ferrer-20030312.pdf

[1a] Want, R., When Cell Phones become Computers, Pervasive Computing, IEEE. Volume 8, Issue 2, Page(s): 2 – 5, Apr-June 2009.

[1b] Reynolds, F., Web 2.0–In Your Hand, Pervasive Computing, IEEE, Volume 8, Issue 1, Page(s):86 – 88, Jan.-March 2009.

[2] Canos, J., Letelier, P., Penadés, C., Metodologías ágiles en el desarrollo de Software.

[3] Boehm, B., Turner, R., Balancing agility and discipline: A guide for the perpexled, Addison-Wesley, 2003.

[4] Boehm, B., Agile and plan-driven methods, USC-CSE Affilities'Workshop, 2002.

[5] Abrahamsson, P., Keynote: Mobile software development – the business opportunity of today, en Proc. of the International Confrence of Software Development, 2005.

[6] Forman, G., Zaharjon, J., The challenges of mobile computing. IEEE Computing, Vol. 27, No. 4, 1994.

[6b] Venkatraman, S., Mobile Computing Models – Are they Meeting the Mobile Computing Challenges? .ACM New Zealand Bulletin, 2005. Disponible en http://oldwww.acm.org/chapters/acm_nz/bulletin/vol1/issue1/articles/id01.pdf

[7] Rahimian, V., Ramsin, R., Designing and agile methodology for mobile software development: a hybrid method engineering approach. Second International Conference on Research Challenges in Information Science, 2008. RCIS 2008, 3-6 June 2008, Marrakech. pp. 337-342.

[8] Ulrich, K. T., S.D. Eppinger. Product Design and Development, 3º edición. McGraw-Hill, 2004.

[9] Clements, P., Northrop, L., Software Product Lines, course notes of Product Line Systems Program, Software Engineering Institute, Carnegie Mellon University, 2003.

[10] Abrahamsson, P. et al., Mobile-D: an agile approach for mobile application development, Conference on Object Oriented Programming Systems Languages and Applications, Poster Session, Vancouver, Canada, October, 2004.

[11] Abrahamsson, P. et al., "Agile Software Development of Mobile Information Systems", Pekka Abrahamsson, VTT Technical Research Center of Finland, 2007. Disponible en http://www.springerlink.com/content/0420084tn1577357/

[12] Heyes, I. S., Just Enough Wireless Computing, Pretince Hall, 2002.

[13] Dunlop, M., Brewster, S., The Challenge of Mobile Devices for Human Computer Interaction, Personal and Ubiquitous Computing Volume 6, Issue 4, 2002.

[14] Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P., and Salo, O. 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. ACM, New York, NY, 174-175

[15] Mobile-D homepage, http://agile.vtt.fi/mobiled.html

[16] Hedberg, H., Iisakka, J., Technical Reviews in Agile Development: Case Mobile-D™, *Quality Software, International Conference on*, pp. 347-353, Sixth International Conference on Quality Software (QSIC'06), 2006.

[17] Ihme, T., Abrahamsson, P., The Use of Architectural Patterns in the Agile Software Development of MobileApplications, International Journal of AgileManufacturing, vol. 8, issue 2, 97-112, 2005.

[18] Beck, K., Extreme Programming Explained: EmbraceChange, Addison-Wesley, 2000.

[19] Cockburn, A., Crystal Clear, A Human-PoweredMethodology for Small Teams, Addison-Wesley, 2004.

[20] Mobile-D case studies homepage, http://agile.vtt.fi/case.html

[21] Kruchten, P., The Rational Unified Process: An Introduction, Addison-Wesley Professional, 1999.