Métodos para el desarrollo de aplicaciones móviles

Robert Ramírez Vique

PID_00176755

Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

En este módulo nos centraremos en los problemas que surgen en el desarrollo de aplicaciones y servicios móviles. En él os mostraremos, desde un punto de vista general, las oportunidades y dificultades propias del entorno.

El desarrollo de una aplicación o servicio conlleva una gran incertidumbre. Sin embargo, existen sistemas para paliar los riesgos asociados. En el caso del las aplicaciones móviles, las dificultades son mayores, si cabe (algunos problemas ya existían con los primeros desarrollos móviles, como la fragmentación o la calidad del servicio de las redes de telefonía). Con el tiempo, han ido apareciendo nuevas dificultades, como el acceso a la información del entorno o el control de las diferentes capacidades de los dispositivos. Al mismo tiempo, las oportunidades de negocio aparecen constantemente, lo que permite crear desde juegos de gran complejidad (reservados hasta ahora a consolas de gran potencia) hasta aplicaciones que nos ayuden a amueblar nuestro hogar.

Debido a esta situación, resulta muy difícil ofrecer una receta mágica para el desarrollo de aplicaciones móviles y, por tanto, se hace imprescindible aprender y adaptar los métodos y los conocimientos adquiridos. En este módulo os explicaremos las situaciones, los métodos y las estrategias oportunas para minimizar estos riesgos e implementar las soluciones móviles, así como para conseguir el mejor rendimiento de las capacidades de los dispositivos.

En el pasado se ha hablado de las aplicaciones móviles y, a pesar de que los móviles ya tenían una gran penetración en el mercado y de que su uso como herramienta de trabajo o elemento de la vida diaria era bastante común, las aplicaciones móviles no habían acabado de despegar. Las razones son varias, desde el intento infructuoso de conseguir aplicaciones ejecutables en todos los dispositivos, hasta el coste asociado a las mismas, lo que ha hecho que solo algunas aplicaciones hayan sido ampliamente usadas (como, por ejemplo, el SMS y el MMS).

SMS

SMS (*short message service)*

MMS

MMS (*multimedia message system*)

Actualmente, más del 70% de la población dispone de dispositivos móviles. El número de *smartphones* no para de crecer (el 90% de los nuevos dispositivos son *smartphones*, según los estudios de Gartner). Es, sin lugar a dudas, el sector que mayor innovación y expectación está generando y generará. Actualmente se dan muchos factores que hacen que casi nadie quede fuera del ecosistema móvil, por lo que es un momento perfecto para conocer mejor sus entresijos. Algunos de estos factores son los que explicamos a continuación:

- Las mejoras en las características *hardware* de los dispositivos móviles gracias a la inclusión de los fabricantes de la electrónica de consumo, que han visto un nicho de negocio y no quieren perder la oportunidad.

- La diversidad en las plataformas y dispositivos, de manera que se puede cubrir un gran abanico de posibles consumidores. Además, aparecen novedades a un gran ritmo, que no parece decaer. Sin duda, hay un papel especial para algunas apariciones, como son las de iOs (iPhone, iTouch y iPad) y Android, que han dado una perspectiva diferente.

- El uso generalizado de los dispositivos móviles (*smartphones*, *tablets pc*, televisores, etc.) en muchos aspectos de la vida cotidiana, que ha permitido que entren en muchos mercados. Lo que antes parecía reservado a las escenas de ciencia ficción, hoy está al alcance de la mano.

- La popularización (en aumento) de las tarifas de Internet móvil para conseguir una mayor cuota de mercado.

- La aparición de una gran cantidad de nuevas aplicaciones a diario, disponibles para el gran público gracias a las tiendas de aplicaciones o *market places*.

- Las nuevas formas o facilidades de venta de las aplicaciones, que hacen más atractivo para las empresas el desarrollo de aplicaciones para este tipo de dispositivos.

- La aparición de las redes sociales, cuyo propósito se ve complementado y potenciado con las aplicaciones móviles.

Sin duda, esto nos obliga, como profesionales del sector, a conocer los retos y posibilidades de este entorno.

En este módulo veremos, para empezar, una introducción a la situación del desarrollo de aplicaciones móviles. En ella, veremos por qué es peculiar y qué lo diferencia de otros procesos de construcción de aplicaciones.

Después, explicaremos detalladamente un método de desarrollo de aplicaciones móviles y expondremos las mejores prácticas en cada una de las fases del desarrollo.

Finalmente, repasaremos las opciones de negocio posibles en mundo de los móviles.

Objetivos

Con este módulo queremos proporcionaros un conocimiento amplio y variado de las alternativas para el desarrollo de aplicaciones móviles. En concreto, con el estudio de este módulo, pretendemos que consigáis los siguientes objetivos:

1. Que conozcáis los problemas de los desarrollos de aplicaciones para móviles.

2. Que veáis las restricciones y posibilidades de dichas aplicaciones.

3. Que conozcáis un método de desarrollo (pondremos especial atención a los problemas de las aplicaciones para dispositivos móviles).

4. Que conozcáis las herramientas necesarias para aplicar dicho método a las nuevas tecnologías emergentes.

5. Que seáis capaces de afrontar todas las fases de un proyecto relacionado con el desarrollo de aplicaciones móviles y dispongáis de herramientas para afrontarlo con garantías.

1. Ecosistema de aplicaciones móviles

Por ecosistema móvil nos referimos al conjunto de actores necesarios para poder tener los dispositivos móviles y a las aplicaciones para los mismos. En concreto, en el ecosistema móvil se incluyen las operadoras de telecomunicaciones, los fabricantes de *hardware* y todos los elementos de *software* que intervienen en la ejecución de la aplicación.

Todas las aplicaciones se ejecutan dentro de un ecosistema. Por lo tanto, para conseguir un desarrollo satisfactorio, es ideal conocerlo. Existen varios factores que afectan al ecosistema, como la infraestructura de la aplicación, el sistema operativo, los métodos de entrada de información, los propios usuarios, los canales de distribución de la aplicación, etc.

Por ejemplo, en el caso de las aplicaciones web, un punto característico es que debemos acceder a ellas mediante un navegador; esto condiciona muchas otras cosas, y para poder hacer una buena aplicación web, se debe conocer, sin duda, esta información. En el caso de las aplicaciones de sobremesa, tenemos un mayor control, pero también tenemos mayor diversidad, debido a los diferentes sistemas operativos disponibles. Lo mismo sucede con los servidores y con las diferentes redes o protocolos que tienen que soportar.

En el caso de las aplicaciones móviles, el ecosistema es aun más heterogéneo que en el resto de desarrollos. Pueden ejecutarse en diferentes tipos de dispositivo, ya sea en un móvil antiguo o bien en uno nuevo, un *smartphone* o un *table PC*, o incluso en aparatos menos evidentes, como un televisor o una *smart card*. Estos dispositivos suelen estar conectados a Internet mediante una conexión que se contrata con una operadora. Todo esto compone, como podéis ver en la siguiente tabla, un ecosistema con muchos actores a tener en cuenta para el desarrollo de aplicaciones móviles.

Smart cards

Las smart cards o tarjetas inteligentes son tarjetas que tienen un circuito integrado de tamaño de bolsillo en el que se puede programar algún tipo de lógica. Un ejemplo son las tarjetas de crédito con microchip.

Ecosistema de los dispositivos móviles

Las diferentes capas de actores que influyen hasta conseguir un servicio (como, por ejemplo, SMS o Internet móvil).

Servicios
Aplicaciones
Framework de aplicaciones
Sistemas operativos
Plataformas
Dispositivos
Redes (GPRS, 3G, etc.)
Operadoras

Sin duda, el ecosistema de una aplicación para dispositivos móviles es algo que no nos planteamos en un primer momento. Este ecosistema constituye una de las mayores dificultades en lo que respecta al desarrollo de aplicaciones, ya que acaba causando, entre otras cosas, una mayor fragmentación de la aplicación. Esto se implica que el desarrollador debe tener en cuenta muchos factores para que la aplicación funcione como desea.

Dentro de este ecosistema, disponemos de información que puede ser muy útil para nuestras aplicaciones, desde la información de la red de datos actual para adaptar los contenidos hasta la información del propio dispositivo (como, por ejemplo, su posición geográfica). También encontramos capacidades que en otros entornos no encontraríamos, como la de localizar otros dispositivos en movimiento, la de dar información sobre nuestro entorno (localización, orientación, presión atmosférica, etc.), la de conseguir información sobre el usuario (contactos, calendario, etc.) o, incluso, la de disponer de medios de pago mucho más directos (mediante la operadora o mediante el propio dispositivo).

En este apartado os mostraremos los retos y las oportunidades que nos ofrece este ecosistema móvil.

1.1. Fragmentación

Uno de los principios básicos para desarrollar aplicaciones consiste en intentar tener el código más simple posible, de manera que se reduzca la complejidad, se eviten los posibles errores y se facilite el mantenimiento. Esto ha resultado muy difícil debido a la fragmentación, que existe en los entornos de aplicaciones más conocidos.

> La fragmentación es una situación, o el conjunto de condicionantes de una situación, en la que no es posible compartir una misma aplicación entre diferentes ecosistemas. Es decir, la fragmentación impide que se pueda compartir la aplicación sin adaptar los ecosistemas.

Esta fragmentación puede ocurrir por muchos factores, los cuales provocan diversidad y entropía en las aplicaciones que queremos crear. La fragmentación puede originarse por los siguientes motivos:

- ***Hardware* diferente**: Como, por ejemplo, dispositivos con componentes distintos: tamaño o densidad de la pantalla, teclado, sensores, capacidad de proceso, etc.

- ***Software* diferente**:

  - **Plataforma diferente**. Una plataforma, un *framework*, un sistema operativo (o las versiones de cualquiera de ellos) puede generar la fragmentación de las aplicaciones.

  - **Diferencias en las implementaciones**. Por ejemplo, diferencias en la implementación del estándar, o bien errores conocidos de versiones concretas.

- **Variaciones de las funcionalidades**. Por ser una versión con menos privilegios (versión de pago y versión gratuita) o según los roles de los propios usuarios de la aplicación.

- **Preferencias de usuario**. Las más habituales son las localizaciones de la aplicación (idioma, orientación del texto, etc.).

- **Diversidad del entorno**. Derivado de la infraestructura, como pueden ser los operadores y sus API, los problemas de cortafuegos, las limitaciones de las redes, el *roaming*, etc.

Roaming

El roaming o itinerancia es un concepto relacionado con la capacidad de un dispositivo de moverse de una zona de cobertura a otra.

Esta fragmentación puede afectar a todo el proyecto de desarrollo, desde el modelo de negocio hasta el despliegue, además de la implementación y las pruebas. Es imprescindible tratarla muy seriamente.

Si no se trata correctamente, esta fragmentación puede causar muchos problemas. A continuación, os mostramos algunos:

- Reducir la calidad del producto. Debido a la mayor complejidad de las soluciones fragmentadas, se pueden generar más errores.

- Limitar el número de dispositivos soportados. Para evitar este problema, se puede decidir soportar un número menor de dispositivos (con posibilidades de ampliaciones en un futuro).

- Alargar cualquier fase del proyecto, desde las fases iniciales a la implementación, además del mantenimiento. Esta dilatación en el tiempo supondrá un sobreprecio, así como posible fracaso de dicho proyecto.

- Grandes costes asociados a las pruebas sobre dispositivos reales.

Sin duda, la fragmentación ha sido, y seguramente será, la mayor dificultad y el mayor riesgo en el desarrollo de aplicaciones móviles.

La fragmentación puede tener diferentes grados. No es lo mismo atacar la fragmentación de una aplicación que debe ejecutarse sobre un televisor y sobre un teléfono móvil, que la de una aplicación que debe ejecutarse sobre dos versiones de la misma plataforma. Por esta razón, existen diferentes estrategias para combatirla, y cada una tiene un sentido según el caso concreto.

1.1.1. Un desarrollo para cada escenario

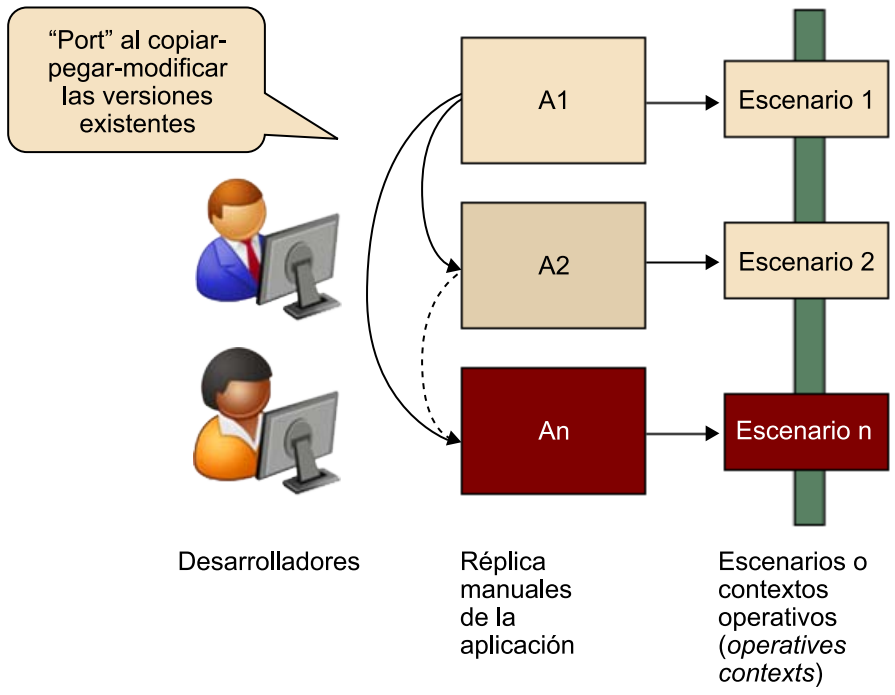
Un escenario es un caso de fragmentación que presenta cualquiera de las posibles causas de fragmentación o varias de ellas.

Es decir, se realiza todo un desarrollo para cada fragmentación que nos podamos encontrar, sin compartir nada. Esto suele ser útil en los casos en que los escenarios son muy diferentes.

El proceso de adaptar la aplicación a un nuevo escenario se llama portar la aplicación.

Esta estrategia es la más costosa de todas, pues no se puede aprovechar nada (o casi nada) sin adaptaciones del código realizado en otros escenarios. Sin embargo, podemos aprovechar al máximo la capacidad del dispositivo y del lenguaje, así como las últimas novedades.

Fragmentación de aplicaciones debido a factores de diversidad



1.1.2. Parte común y derivaciones

La **derivación** es la estrategia más habitual. Según esta estrategia, una parte de nuestra aplicación es común a todos nuestros escenarios, y para cada uno de ellos podemos definir la parte específica correspondiente.

Existen diversas variantes de esta estrategia, en función de cómo se realice la derivación a los diferentes escenarios. Estas derivaciones realizan los cambios específicos de cada escenario para que todos funcionen de la misma manera. Tenemos, pues, las siguientes opciones:

- **Derivación selectiva**. Las modificaciones necesarias están localizadas en unos elementos concretos (ya sean clases del código, ficheros de marcado, estilos u otros recursos), y existe un sistema o herramienta que genera las diferentes versiones mediante la captura y el empaquetado de estos elementos para cada escenario.

- **Derivación usando meta programación**. Se trata de programar algo que se va a ejecutar en varios escenarios. Para conseguir distintos comportamientos hay varias opciones:
  - Mediante la inyección de objetos o recursos (imágenes, ficheros XMLS, etc.) en el código, de manera que nuestra aplicación deje estos objetos vacíos y se rellenen, en tiempo de ejecución, los objetos específicos de cada escenario. Esta estrategia se baja en el patrón de diseño *Inversion of control*.

  - Utilizando preprocesadores, que se encargan de cambiar o ampliar nuestro código para adaptarlo a los diferentes escenarios antes de ejecutar.

- **Generación automática**. El *software* se debe escribir de una manera específica y solo una vez. A posteriori, existe un proceso que genera automáticamente las aplicaciones correctas para cada escenario, normalmente transformando nuestra aplicación al código particular de cada escenario. En este punto existen más variantes.

Esta estrategia puede ser menos costosa que la anterior, ya que se puede aprovechar parte del desarrollo de las alternativas y reducir así los costes de implementación. Sin embargo, a menudo requiere conocer tanto los lenguajes o entornos de los diferentes dispositivos como las herramientas (o, incluso, los lenguajes) necesarias para llevar a cabo las adaptaciones. En este caso, se suelen conseguir aplicaciones aprovechando, en gran medida, el potencial de los dispositivos, aunque según la estrategia escogida, puede ocurrir que perdamos el control sobre el código generado y, por lo tanto, potencia de desarrollo.

1.1.3. Adaptación única

Mediante la adaptación única, podemos conseguir una versión que funcione en todos los casos sin necesidad de realizar más cambios. Existen varias maneras de afrontar esta estrategia:

- **Mínimo común denominador**. Se trata de conseguir una aplicación mediante la reducción de los puntos de fragmentación, de manera que no exista la necesidad de adaptar la aplicación.

- **Todos en uno**.
  - La aplicación es capaz de conocer la información necesaria para poder adaptarse a todos los dispositivos. Por ejemplo, para evitar el problema de diferentes pantallas, se genera una aplicación con ventanas autoescalables.

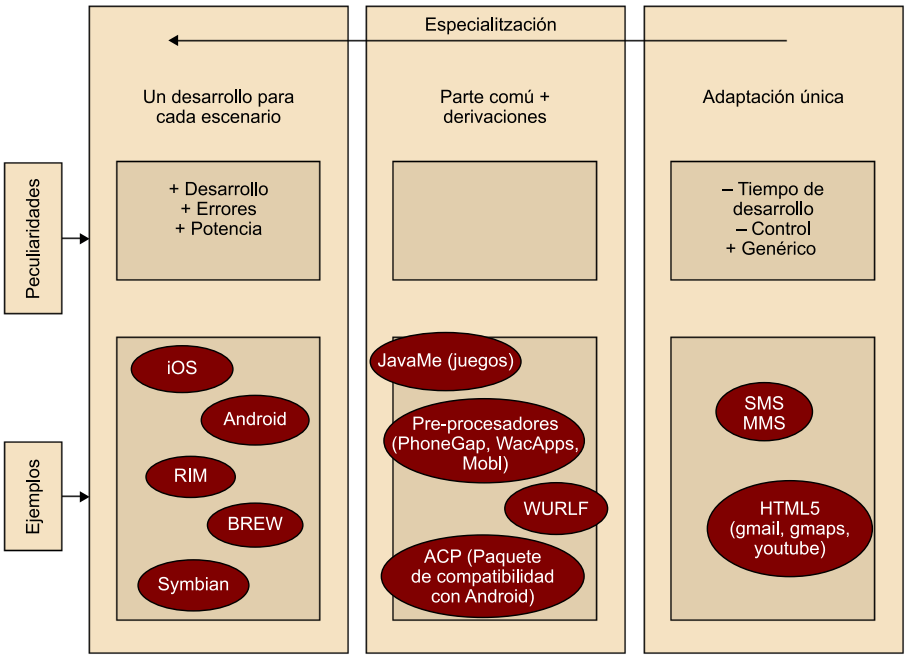
  - Son los dispositivos los que se adaptan; es decir, el *software* se escribe de manera abstracta y, cuando se llega al problema de la fragmentación, se pasa el testigo al dispositivo que sabe cómo tratarlo. Un ejemplo puede ser el acceso a los contactos o a las llamadas de teléfonos mediante API[1] abstractas.

(1) *Application program interface*

Esta estrategia es la más económica en lo que respecta a desarrollos distintos y conocimientos requeridos. Como podéis intuir, también es la que se queda en la capa más superficial del potencial de los dispositivos.

A pesar de esto, se consiguen soluciones potentes, ya que con este tipo de aplicaciones se pueden realizar una gran variedad de aplicaciones, desde aplicaciones muy simples y generales (como puede ser una aplicación basada en SMS) hasta aplicaciones como Google Places, que conoce nuestra localización, u otras que pueden trabajar con datos sin conexión (modo "fuera de línea").

Categorización de tipos de adaptaciones a los problemas de fragmentación



1.2. Contexto

El contexto se define como las informaciones conjuntas de la situación actual, el usuario, la información del dispositivo y la información de otras aplicaciones en un momento dado del tiempo.

Las aplicaciones móviles pueden aprovechar mucho más el contexto en el que están ejecutando, sobre todo si las comparamos con aplicaciones tradicionales. Esto se debe a diferentes factores, entre los que se encuentran las nuevas capacidades de los dispositivos, la capacidad de acceder a la información que el propio dispositivo tiene del usuario (incorporando, de esta manera, las capacidades o relaciones sociales del mismo) y las capacidades que puede aportar el entorno en el que estamos o el momento en que usamos la aplicación.

1.2.1. Capacidades de los dispositivos

Los nuevos dispositivos nos aportan mucha información sobre nuestro entorno. Por ejemplo, la más clara y conocida es la posición geográfica actual, que nos permite realizar aplicaciones basadas en la localización (LBS). Además, existen otras informaciones, (como, por ejemplo, la orientación, la presión, la luz, etc.). La posibilidad de grabar imágenes, vídeos y audio también nos aporta más información sobre el contexto (las aplicaciones que reaccionan al habla o las de realidad aumentada son ejemplos de aplicaciones que aprovechan este tipo de contexto). Existen capacidades más obvias (como la de saber la hora actual, el idioma, o la zona horaria) que ayudan a realizar aplicaciones

Aplicaciones basadas en la localización

Las aplicaciones basadas en la localización, en inglés *location based services* (LBS), son servicios que intentan dar un valor añadido gracias al conocimiento de la ubicación geográfica del usuario.

más contextualizadas. Otro aspecto que ha hecho mejorar la situación es, sin duda, la mejora de las comunicaciones inalámbricas a partir de la aparición del 3G o del UMTS, o la mejoras de Bluetooth o nuevos estándares de WIFI.

Realidad aumentada

La realidad aumentada (RA) es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta a tiempo real.

Muchas de las aplicaciones aprovechan varias capacidades. Por ejemplo, una aplicación de realidad aumentada aprovecha varias de ellas:

- GPS, para conocer la posición geográfica y saber qué se debe mostrar.
- Brújula, para saber la orientación actual
- Acelerómetro, para saber cuál es la orientación exacta de nuestro dispositivo y superponer las capas.
- Cámara, para poder captar nuestro alrededor y así ampliar la información (en ocasiones, incluso, varias cámaras).
- Conexión a Internet, para poder obtener la información para ampliar nuestra realidad. Esta conectividad puede venir mediante Internet móvil o WiFi, entre otros.
- Capacidad de procesamiento gráfico muy mejorada, con chips de aceleración gráfica potentes.

Además, están apareciendo nuevos protocolos o capacidades que ayudarán a mejorar las aplicaciones móviles, como es el caso de *NFC*. NFC, además de permitir realizar pagos con el móvil, permitirá comunicar información de entorno, como puede hacerse ahora con RFID, para llegar a aplicaciones móviles (por ejemplo, una visita guiada en un museo con información constante sobre lo que estamos viendo).

Sin duda, el salto en los últimos años ha sido muy grande con respecto a los móviles que "únicamente" eran capaces de llamar y enviar mensajes. Ahora tenemos al alcance de la mano muchas funciones que aportan posibles aplicaciones para conocer mucho mejor nuestro entorno.

1.2.2. Ubicuidad

La ubiquidad (o la omnipresencia) se define como la capacidad de acceder a toda la información o a todos los servicios que necesita el usuario en cualquier momento y circunstancia mediante el dispositivo que tengamos actualmente.

Con la situación actual de las aplicaciones móviles, estamos muy cerca de conseguir esta ubicuidad, pues las aplicaciones móviles pueden ir con el usuario continuamente. De hecho, se han convertido en accesorio diario, tanto en el plano profesional como en el personal. Gracias a las nuevas capacidades, tanto de conectividad como de proceso, podemos acceder a casi todos los servicios disponibles en Internet. El acceso a estos servicios puede llegar a ser más práctico que desde un ordenador personal, y en eso se centra el aprovechamiento de la ubicuidad.

En este sentido, en cualquier momento podemos recibir información útil para nosotros en el momento actual, como por ejemplo un correo electrónico o un mensaje instantáneo, pero también puede ser información sobre el lugar donde estamos en ese momento o recordatorios relacionados con el sitio o la persona con quien estamos.

También podemos conocer el estado del tiempo, del tráfico y muchos otros parámetros en el momento y lugar que nos interesa. En el caso de aplicaciones para otro tipo de dispositivos, como un televisor, también podemos tener información contextualizada, como juegos, concursos o simplemente comentarios relacionados con el programa que estemos viendo en ese momento.

Por lo tanto, tenemos una serie de beneficios relacionados con la ubiquidad en lo que respecta a las aplicaciones para dispositivos móviles:

- El dispositivo móvil es el primer medio de comunicación masivo real, dado que es capaz de llegar a casi todos los usuarios y en todo momento.
- El primer medio de comunicación permanentemente encendido, pues es capaz de captar y enviar información aun cuando está apagado (apagado en el sentido del usuario, es decir, cerrado, pero no totalmente apagado).
- Primer medio que está siempre con el usuario.
- Medio masivo que tiene incorporado un sistema de pago, que en este caso es mediante la operadora.

Media masivo

Media masivo o medio de comunicación de masas son los medios de comunicación recibidos simultáneamente por una gran audiencia.

1.2.3. Contexto social

Otro punto importante a tener en cuenta y que, sin duda, ayuda mucho a que una aplicación triunfe actualmente en el sector es su componente social. Esto significa la posibilidad de interaccionar mediante nuestras aplicaciones con nuestros amigos, nuestra familia y otros conocidos (o incluso desconocidos).

Ejemplos de interacción

A continuación exponemos algunos ejemplos de interacción:

- Compartir nuestra puntuación en un juego.
- Ver usuarios que están actualmente interaccionando con la aplicación, o bien con el tema que nos interesa en ese momento (por ejemplo, poder comentar una serie de televisión mientras la vemos).
- Publicar en cualquier momento lo que se estamos haciendo y ver lo que hacen nuestros amigos.
- Recibir avisos de cuando nuestros amigos están cerca, llegan a un lugar concreto o hacer alguna acción destacable.

Obviamente, el gran crecimiento de las redes sociales actuales es el caldo de cultivo ideal para este tipo de aplicaciones sociales móviles (MoSoSo[2]), pues actualmente hay una gran penetración de las redes en Internet a nivel mundial. Incluso hay casos, como Android, donde para sacarle el mayor partido a nuestro dispositivo es necesario acceder con unas credenciales de Google y su capa social, cosa que automáticamente nos conecta con todos nuestros amigos de dicho servicio y permite sacarle el máximo partido a muchas de nuestras aplicaciones.

(2)MoSoSo (*mobile social software*)

Las aplicaciones sociales gozan de gran proyección en el móvil debido a la capacidad de omnipresencia (o ubicuidad) de dichas aplicaciones; es decir, que nos acompañan en el momento adecuado. Por ejemplo, podemos subir las fotos que acabamos de hacer a nuestra red social o acceder a la información de nuestros contactos.

1.2.4. Costes

Sin duda, que esté contextualizada es un gran punto a favor para nuestra aplicación, pero como toda aportación, tiene su desventaja, que en este caso tiene que ver con los costes. Así, para poder tener este contexto, necesitamos lo siguiente:

- Acceso a Internet mediante redes WiFi o MWWAN.
- Proximidad de otros dispositivos para compartir información.
- Estar constantemente conectado a las redes sociales, lo que requiere que dispongamos de cuentas en esas redes sociales y que tengamos activado el acceso (con los posibles problemas de privacidad).
- Conexiones a otras redes inalámbricas, Bluetooth, GPS, NFC, etc.
- Grandes capacidades de proceso en nuestros teléfonos para realizar las acciones, ya sea la capacidad de proceso interno o bien mediante accesorios.
- Debido a innovaciones y cambios, tenemos que pagar el coste de actualizar nuestra aplicación. Es decir, durante o después del lanzamiento de nuestra aplicación, pueden aparecer nuevas fuentes de fragmentación *hardware* o

software que provoquen que nuestra aplicación se tenga que actualizar. Esto sucede en todos los desarrollos, pero especialmente en el desarrollo de aplicaciones para dispositivos móviles, debido a la gran velocidad del cambio en este sector.

Los costes finales se traducen en:

- **Limitación de la vida de las baterías**. Esto significa que, al utilizar muchas de estas capacidades (por ejemplo, el GPS), hacemos que nuestro terminal pierda autonomía, pues se necesita destinar energía a estos periféricos, y lo mismo sucede con el resto de capacidades. En la materia que nos incumbe, el desarrollo de aplicaciones, tenemos que tener esto muy presente a la hora de diseñar y escribir nuestras aplicaciones, ya que puede afectar mucho a su rendimiento.

- **Vulneración de la privacidad**. Nuestros dispositivos móviles contienen cada vez más información personal y confidencial, y requieren acceso a esta información para poder sacarle el mayor partido y conseguir integrarse en el contexto. Por eso, siempre que realicéis una aplicación que requiera acceso a información o acceda a datos de otras fuentes (como las redes sociales), debéis tener el permiso explícito del usuario, y este se debe poder revocar.

- **Necesidades de *hardware***. Por ejemplo, la necesidad de mayor velocidad de transmisión. Si nos encontramos en una zona con poca cobertura para nuestra red de transmisión de datos, puede ocurrir que nuestras aplicaciones no funcionen correctamente.

- **Necesidades de inversión no previstas debido a novedades del mercado**. Si aparece una fuente de fragmentación nueva (por ejemplo, un nuevo dispositivo), debemos invertir en dar soporte a ese nuevo dispositivo. Esta inversión puede suponer no tener que hacer nada o realizar un desarrollo nuevo.

1.2.5. Conclusiones

Sin duda, un aspecto diferenciador de cualquier aplicación móvil debe ser el uso del contexto, de uno o varios modos, pues eso constituye un valor diferenciador con respecto a las aplicaciones de otros soportes.

2. Características de un proyecto de desarrollo para dispositivos móviles

Como hemos visto anteriormente, los desarrollos de aplicaciones sobre dispositivos móviles tienen grandes oportunidades y posibilidades, pero también algunas dificultades añadidas que pueden llegar a ser un riesgo para conseguir que los proyectos sean un éxito.

Por lo tanto, al afrontar un proyecto de desarrollo de *software* para dispositivos móviles, o bien proyectos en los que una parte esté orienta a dispositivos móviles, tendréis que tener un método que, además de soportar la problemática habitual del desarrollo de *software*, se encargue de dar soluciones y de minimizar riesgos, para el caso concreto del desarrollo de *software* móvil.

En este apartado veréis una visión general del tipo de aplicaciones para dispositivos móviles que os podéis encontrar y las compararemos para que podáis elegir la mejor alternativa cuando os enfrentéis a un proyecto. Este punto es muy importante porque condiciona todas las fases del desarrollo. De hecho, podríamos decir que, según el tipo de aplicación, los desarrollos son muy diferentes entre sí.

Después de ver los tipos de aplicaciones que existen, es necesario que conozcáis las opciones disponibles para desarrollar dicha aplicación. En este punto daremos un repaso a las diferentes estrategias, lo cual os ayudará a entender mejor el mundo del desarrollo para dispositivos móviles.

Lo siguiente que veréis en este apartado serán los métodos de desarrollo existentes aplicados a desarrollo de aplicaciones móviles. Veréis las peculiaridades generales del desarrollo móvil y cómo se pueden utilizar estos métodos para solucionar las problemáticas.

A continuación veréis las fases del desarrollo. Haremos hincapié en las diferencias entre las fases de un desarrollo de aplicación normal y las de un desarrollo para dispositivos móviles.

Para que podáis abordar los proyectos, en este apartado veréis una introducción a las estrategias que existen en el desarrollo de aplicaciones móviles. Así conoceréis las diferentes alternativas.

Después os explicaremos detalladamente un método de desarrollo y sus fases, y podremos especial atención a las peculiaridades del desarrollo de aplicaciones móviles.

2.1. Tipos de aplicaciones

Existen muchos tipos de aplicaciones, ya que el tipo de dispositivo que tenemos en mente puede ser muy versátil. Además, hay aplicaciones en las que, seguramente, no pensamos, como las aplicaciones para dispositivos especiales (por ejemplo, un televisor o una consola), pues a pesar de que no son dispositivos móviles, suelen estar programados con las mismas tecnologías y limitaciones.

Las aplicaciones se pueden clasificar en función de la utilidad que queramos darles, o bien según las necesidades del dispositivo y de la complejidad de la propia aplicación.

Cuando os enfrentéis a un proyecto para dispositivos móviles, es importante que conozcáis las opciones que tenéis, sus puntos fuertes y débiles. Con esta información, podréis elegir mejor la aplicación a realizar.

A continuación os mostramos una división según el tipo de desarrollo.

2.1.1. Aplicaciones básicas

La **aplicaciones básicas** son aplicaciones de interacción básica con el dispositivo que únicamente envían o reciben información puntual del usuario.

La aplicaciones básicas se pueden gestionar simplemente con el envío de mensajes de texto (SMS o MMS). Estas aplicaciones existen desde hace mucho tiempo y, aunque han tenido gran aceptación y uso, actualmente están comenzando a dejar paso a aplicaciones más complejas.

Las aplicaciones básicas tienen las siguientes ventajas:

- simplicidad
- facilidad de venta
- gran cantidad de usuarios potenciales

También presentan algunas desventajas:

- poca o casi nula capacidad de procesamiento del contexto
- muy baja complejidad de las aplicaciones realizadas
- limitaciones impuestas por la tecnología sobre los diseños de las aplicaciones (ciento sesenta caracteres de texto)

Ejemplo de aplicación básica

Una ejemplo de aplicación es una aplicación para conocer cuál es el tiempo de llegada de un autobús (se envía un SMS a un número concreto y se recibe una respuesta).

2.1.2. Webs móviles

Las **webs móviles** son aquellas webs que ya existen actualmente y que son adaptadas específicamente para ser visualizadas en los dispositivos móviles. Adaptan la estructura de la información a las capacidades del dispositivo, de manera que no saturan a los usuarios y se pueden usar correctamente desde estos dispositivos.

Lenguaje de marcas

Lenguaje de marcado o lenguaje de marcas (*markup language*) es el lenguaje que estructura la información mediante marcas o etiquetas (*tags*).

Dependiendo de los dispositivos a los que queramos llegar, deberemos adoptar un lenguaje de marcado u otro, pues hay dispositivos que solo soportan un tipo de marcas. En los dispositivos de tipo *smartphone* actuales, podemos visualizar una web que no esté adaptada a dispositivos móviles, pero en ese caso no estaríamos hablando de una aplicación para dispositivos móviles, sino de una capacidad del dispositivo.

Este tipo de aplicaciones son aplicaciones básicas que, por lo general, no usan objetos dinámicos como Javascript. Por tanto, no tienen todo el potencial de un navegador web de sobremesa. Se utilizan estándares web como, por ejemplo, XHTML, WML, XHTML-MP, c-html, etc. y, en general, versiones previas a la nueva versión del estándar HTML: HTML 5. Están pensadas para dar soporte a dispositivos de media y baja gama.

Las ventajas de las webs móviles son las siguientes:

- **Fácil implementación, testeo y actualización**. Incluso se puede realizar gran parte del desarrollo sin necesidad de utilizar dispositivos móviles ni emuladores, hasta llegar a las fases finales del desarrollo.
- **Lenguaje conocido y estándar**. Los lenguajes de marcas son muy conocidos hoy en día por la mayoría de los desarrolladores, y en la mayoría de los casos se trata de subconjuntos de lenguajes conocidos.
- Pueden soportar múltiples dispositivos con un único código fuente. Para soportar fragmentación entre dispositivos, es necesario utilizar técnicas especiales (como el WURLF).

WML

Casi todos los dispositivos móviles actuales tienen soporte para algún tipo de lenguaje de marcado. Inicialmente era WML (*wireless markup language*), pero también se utilizan otros.

WURLF

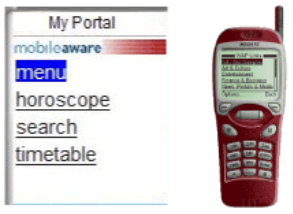
WURLF es un repositorio de información que identifica, a partir de la metainformación de una petición web de un dispositivo móvil, sus capacidades y limitaciones. Para saber más sobre WURLF, podéis visitar el siguiente sitio web:
http://wurfl.sourceforge.net/

Los inconvenientes de las webs móviles son los siguientes:

- Es difícil soportar múltiples dispositivos, así como conseguir la misma experiencia de usuario con varios tipos de navegadores.
- Ofrecen grandes limitaciones a la hora de realizar programas, tanto de proceso como de acceso a la información del dispositivo y del usuario. Por tanto, es difícil conseguir aplicaciones contextualizadas.
- En muchos casos está pensado para ser visualizado con conexiones lentas, pero dichas conexiones pueden ser demasiado lentas y provocar una experiencia de usuario deficiente.
- En la actualidad, la mayoría de los dispositivos nuevos están incorporando estándares más nuevos (como HTML 5), por lo que no se está trabajando en mejorar estos estándares.
- El número de dispositivos que solo pueden ver una página web con este tipo de lenguajes de marcas está disminuyendo.

Ejemplos de web móviles

Cualquier web pública con información para el contribuyente es un buen ejemplo de web móvil. Por ejemplo, http://wap.bcn.cat.



2.1.3. Aplicaciones web sobre móviles

Las aplicaciones web sobre móviles son aplicaciones que no necesitan ser instaladas en el dispositivo para poder ejecutase. Están basadas en tecnologías HTML, CSS y Javascript, y que se ejecutan en un navegador. A diferencia de las web móviles, cuyo objetivo básico es mostrar información, estas aplicaciones tienen como objetivo interaccionar con el dispositivo y con el usuario. De esta manera, se le saca un mayor partido a la contextualización.

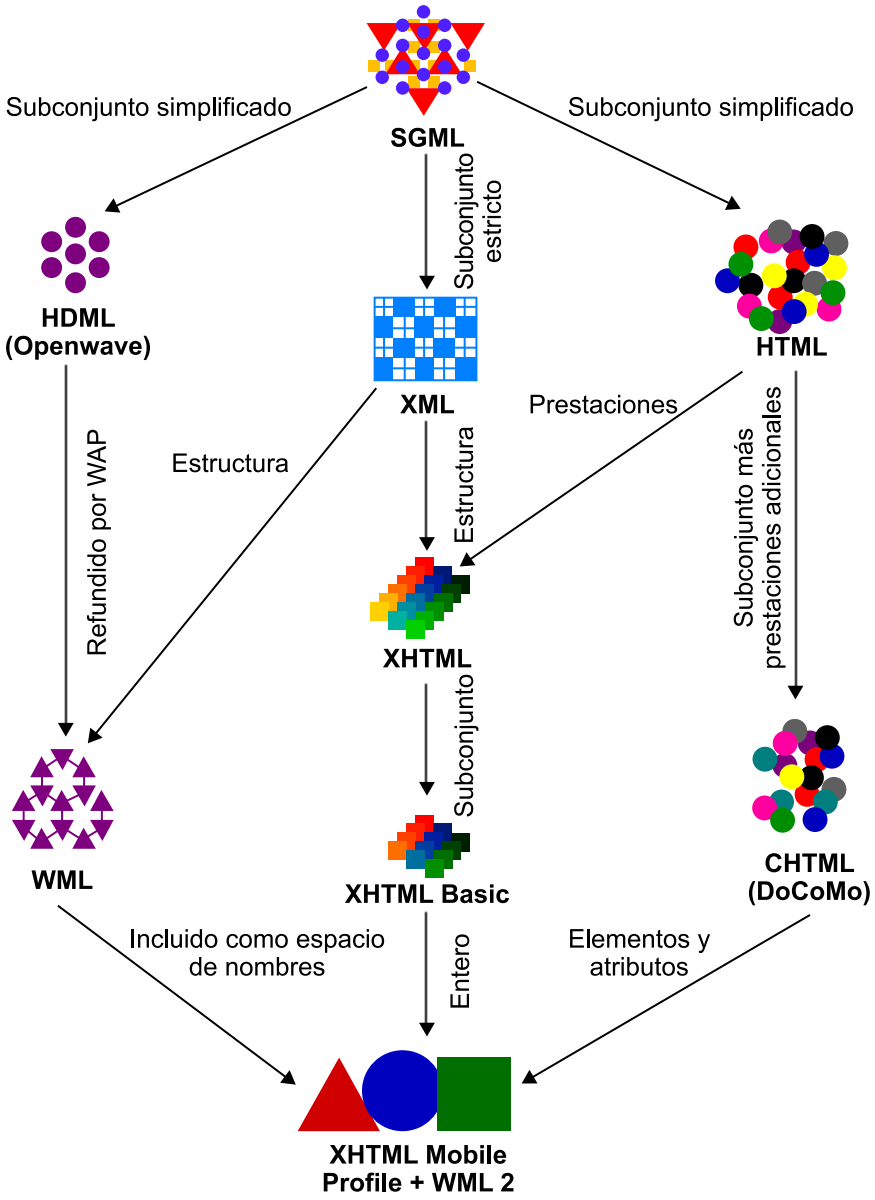
Son aplicaciones especialmente diseñadas para trabajar en el móvil y que intentan aprovechar al máximo sus posibilidades (en ocasiones lo hacen accediendo a datos del contexto: posición geográfica, datos guardados, etc.).

Este tipo de aplicaciones han existido desde hace tiempo, pero con la aparición de navegadores más potentes (por ejemplo, los basados en *WebKit*, como los navegadores de Android, iOs, Nokia S60 series, etc.) se ha pasado básicamente del XHTML a soportar HTML y, sobre todo, HTML 5. Gran parte de este avance se consigue con HTML 5 y CSS3, que nos ofrece, entre otras cosas, la posibilidad de tener aplicaciones muy parecidas y con mucho potencial con respecto a las aplicaciones nativas.

WebKit

WebKit es una plataforma de aplicaciones que funciona como base de varios navegadores (como Google Chrome, Safari, etc.). Están en el motor de renderizado XHTML del navegador Konqueror de KDE.

A continuación presentamos la evolución de los lenguajes de marcado, partiendo del original SGML. Veremos que, finalmente, se llega al que hasta ahora era el lenguaje más usado en el desarrollo de webs para dispositivos móviles: XHTML-MP (Mobile profile). Como se puede observar han existido muchas alternativas, y algunas que no hemos visto, lo cual provocaba una gran dificultad para desarrollar dichas webs:



Las ventajas de las aplicaciones web sobre móviles son las siguientes:

- Posibilidad de acceso a mucha información del dispositivo para realizar aplicaciones relativamente complejas.
- Desarrollo, distribución y pruebas sencillas.
- Convergencia entre aplicaciones de sobremesa y de dispositivos móviles, lo cual tiene muchas implicaciones, como, por ejemplo, que los desarrolladores solo tienen que conocer una tecnología.
- Uso de estándares de la web (claramente definidos).
- Ampliamente soportado por la industria, de manera que la mayoría de los nuevos dispositivos tienen soporte para este tipo de aplicaciones.

Por el contrario, también tienen los siguientes inconvenientes:

- Se necesita un navegador que pueda dar soporte a este tipo de tecnología. Aunque la mayoría de los nuevos dispositivos lo pueden dar, los antiguos no.
- Su rendimiento es menor con respecto a las aplicaciones nativas, pues se ejecuta todo mediante el Javascript del navegador, cuya potencia es limitada.
- Imposibilidad de acceder a todas las posibilidades del dispositivo. No se puede acceder al *hardware* ni a muchos periféricos (como sensores o cámaras). No se puede acceder a mucha de la información del usuario (contactos, sus citas, etc.).

Ejemplo de aplicación móvil

Actualmente existen muchos ejemplos de aplicaciones web realizadas para ser ejecutadas en sobremesa, que han sido rápidamente portadas a los dispositivos móviles, como son mobile.twitter.com, facebook.com, maps.google.com, etc.

Introducción a HTML 5

HTML5 es un gran paso adelante para mejorar los estándares web y conseguir desarrollos para dispositivos móviles más sostenibles. Es por ello que vamos a ver una pequeña introducción a dicho estándar.

Antes de empezar a definirse HTML 5, el consorcio W3C[3] ya estaba trabajando en un estándar para sustituir la gran variedad de estándares web que existían: XHTML 2.0.

(3)World Wide Web Consorcium (W3C) es un consorcio internacional que produce recomendaciones, que no estándares, para la World Wide Web (WWW). El creador, y actual líder, de este consorcio es Tim Bernes-Lee, considerado el creador de la web y concretamente creador de la URL, HTTP y HTML.

Logotipo oficial del estándar HTML 5

XHTML 2.0, al igual que su predecesor XHTML 1.0, contenía elementos XML por lo que era incompatible con HTML 4. Debido a la lentitud del W3C para generar nuevos estándares (HTML 4.01 fue aprobado en 1999), el WHATWG[4] decide tomar como referencia HTML 5 y abandonar la estructura HTML en favor de XML. Finalmente, en 2006, W3C decide participar en la elaboración de HTML 5. Se desarrolla en paralelo XHTML 2.0 y HTML 5. De esta manera, se ha conseguido el soporte de los principales navegadores y se sigue siendo compatible con las versiones anteriores.

(4)Web Hypertex Application Techonolgoy Working Group. Este grupo lo forman las empresas responsables de los principales navegadores web: Apple, Opera, Mozilla, Microsoft y Google.

Con XHTML 5 se quiere conseguir:

- Añadir novedades al HTML y CSS actuales.
- Dar soporte a todos los navegadores, incluyendo los navegadores móviles, evitando una ruptura total con las versiones actuales del estándar.
- Aprovechar las características de los dispositivos actuales, como pueden ser aceleración gráfica o múltiples procesadores.
- Tener mayor libertad de desarrollo, evitando al máximo la necesidad de extensiones propietarias que existen actualmente.

HTML 5 se engloba dentro del estándar Web Applications 1.0, que incorpora algunos estándares más. Este estándar da soporte, por ejemplo, a las versiones XHTML y HTML del estándar y define las API para poder realizar las acciones dinámicas (con ECMAScript[5]).

(5)Estándar sobre el que está basado el popular lenguaje Javascript.

- Visualización de información e interacción con dicha información más potente, a través de los lienzos (*canvas*) y los formularios (*web forms*). En los formularios se tiene validación nativa del navegador.

- Modo sin conexión. Guardado de la información del usuario en el dispositivo físico, mediante el almacenamiento local para por ejemplo soportar falta de conectividad.

- Almacenamiento de datos en el navegador como una base de datos.

- Acceso a datos como la posición geográfica.

- Renderización de objetos gráficos aprovechando la potencia de las tarjetas gráficas de los dispositivos, para así mejorar su rendimiento (gracias en parte al soporte de SVG[6]).

(6)SVG (*scalable vector graphics*), se trata de especificación para definir gráficos bidimensionales, tanto estáticos como animados. Desde 2001 es una recomendación de W3C. SVG está integrado en los estándares web de manera que se puede definir con etiquetas estándar HTML los objetos gráficos.

- Soporte para video y audio (etiquetas <video> y <audio>) sin necesidad de extensiones propietarias.

- Nuevas etiquetas semánticas, como *section, article, header, nav*, etc.

- API para coger y arrastrar (*Drag & drop*).

- Trabajos pesados en segundo plano (*Web Workers*).

Enlaces de interés

En la siguiente dirección web encontraréis diversos ejemplos de las posibilidades de HTML 5:

http://html5rocks.com

Y en esta dirección encontraréis el soporte actual de los navegadores para los diferentes estándares:

http://html5readiness.com/

Paralelamente a la definición de los estándares HTML, se han ido actualizando los estándares correspondientes al estilo de las páginas, en concreto la nueva versión es CSS3. Esta versión está dividida en módulos, que han ido siendo aprobados como especificaciones en momentos diferentes. Algunos de los

más importantes son CSS Selectors, CSS Colors, backgrounds & borders, CSS Multi-column layout, CSS Namespace, Media Queries, CSS Speech, CSS Animations, CSS 3D Transformations, etc. Todos estos estándares también contribuyen a conseguir que la experiencia de los usuarios de webs para dispositivos móviles sea más completa y su programación pueda ser más estructurada.

2.1.4. Aplicaciones web móviles nativas

Existe un tipo de aplicaciones, llamadas aplicaciones web móviles nativas, que no son aplicaciones web propiamente ni tampoco nativas. Se ejecutan con un navegador o, mejor dicho, con un componente nativo que delega en un navegador, y tienen algunas de las ventajas de las aplicaciones nativas.

Este tipo de aplicaciones pueden ser instaladas en el dispositivo, con lo que pueden utilizar los canales estándares de distribución de aplicaciones nativas, o bien incorporarse como accesos directos (como el resto de aplicaciones). Sin embargo, estas aplicaciones no tienen la potencia de las aplicaciones nativas, sino que simplemente ejecutan código en un navegador embebido (generalmente con HTML 5).

Las ventajas de las aplicaciones web móviles nativas son las siguientes:

- Todos los puntos a favor de las aplicaciones web móviles.
- Se pueden considerar, en lo que respecta a la instalación y la distribución, como aplicaciones nativas.

En cambio, sus inconvenientes son los siguientes:

- La mayoría de los inconvenientes de las aplicaciones web móviles, a excepción de la instalación en el cliente.

- La experiencia del usuario es, en ocasiones, contradictoria, pues a pesar de tratarse de una aplicación nativa, requiere de conexión a Internet para poder trabajar y funciona según los tiempos de respuesta del navegador.

Ejemplo de aplicaciones móviles nativas

Aplicaciones de acceso a información confidencial, como son las entidades financieras "en línea", que distribuyen la aplicación, pero por su naturaleza no es recomendable que haya datos en el dispositivo físico.

Existen también los llamados *mobile widgets*, que se pueden englobar como un subtipo de aplicación web móvil. Ejecutan aplicaciones web como aplicaciones nativas, pero a modo de *widget*, es decir, que se pueden ejecutar continuamente o están integradas en la plataforma destino. Un ejemplo de este tipo de aplicaciones sería un buscador de resultados en Wikipedia, o un traductor. Tienen la mayoría de las ventajas de las aplicaciones móviles, pero con limitaciones en lo que respecta a la dimensión, ya que suelen ser ejecutadas en una parte limitada de la pantalla.

Ejemplos de *mobile widgets*

Algunos ejemplos de los *mobile widgets* son WRT (Web Runtime Widgets de Sybiuan), Blackberry Widget SDK, etc.

2.1.5. Aplicaciones nativas

Las aplicaciones nativas son las aplicaciones propias de cada plataforma. Deben ser desarrolladas pensando en la plataforma concreta. No existe ningún tipo de estandarización, ni en las capacidades ni en los entornos de desarrollo, por lo que los desarrollos que pretenden soportar plataformas diferentes suelen necesitar un esfuerzo extra.

Estas aplicaciones son las que mayor potencial tienen, pues aprovechan al máximo los dispositivos y consiguen, de esa manera, una mejor experiencia de usuario.

Existen muchas plataformas, una gran parte de ella ligadas al tipo de dispositivo, aunque también hay plataformas, como Android, que existen para diferentes tipos de dispositivos.

Algunas de las más conocidas son iOs, Android, Blackberry, bada, Java Me, Windows Phone (antes Windows Mobile o Windows Ce), Symbian, Web OS, Brew, etc. Todas ellas tienen diferentes tipos de dispositivos con una base común entre ellos.

Las ventajas de las aplicaciones nativas son las siguientes:

- Acceso total al contexto, con todas las posibilidades que eso conlleva. Consigue las mejores experiencias de usuario.
- Posibilidad de gestión de interrupciones en la aplicación o en las capacidades del dispositivo. Desde saber si tenemos conexión de datos o conexión de localización hasta tener información sobre la batería.
- Son relativamente fáciles de desarrollar si solo se contempla una plataforma.
- Se pueden distribuir por los canales conocidos de aplicaciones que permita la plataforma, con lo que se pueden vender más fácilmente.
- Todas las novedades llegan primero a este tipo de aplicaciones, pues es en este tipo de aplicaciones donde se prueban.

En cambio, tienen sus inconvenientes:

- Portar aplicaciones es costoso. En el caso de querer realizar una aplicación para más de una plataforma, se complica el desarrollo, debido a los problemas de la fragmentación.
- Dependiendo de la plataforma elegida, puede haber fragmentación dentro de cada plataforma, debido a los diferentes tipos de dispositivos o versiones de la plataforma.
- No existe un estándar, por lo que cada plataforma ofrecerá sus peculiaridades.
- Normalmente, para desarrollar, distribuir o probar estas aplicaciones en dispositivos reales, es necesario tener una licencia de pago, dependiendo de la plataforma.
- Las ganancias por estas aplicaciones suelen repartirse entre el creador de la aplicación y la plataforma de distribución.

Ejemplos de aplicaciones nativas

Algunos ejemplos de aplicaciones nativas son los siguientes:

- Aplicaciones para la gestión de la agenda o para encontrar amigos de la agenda con su posición geográfica.

- Alarmas, aplicaciones correo electrónico, etc.

- Aplicaciones sociales, como las aplicaciones de las redes sociales y otras aplicaciones que permiten utilizar el acceso a estas aplicaciones, como los agregadores de redes sociales.

Las siguientes imágenes representan aplicaciones nativas que aprovechan las funcionalidades del dispositivo, como, por ejemplo, aplicaciones de realidad aumentada o juegos.

2.2. Estrategias de desarrollo de aplicaciones móviles

A la hora de emprender un proyecto móvil, es importante que conozcáis las alternativas. En ocasiones es inviable conocerlas todas y, normalmente, resulta muy difícil conocerlas todas en detalle, pero sí es muy recomendable que tengáis una visión general de las opciones y alternativas del mercado. En este módulo os daremos una visión general y ampliaremos algunos aspectos.

Dada la gran fragmentación de plataformas y tipos de aplicaciones que existen, lo primero que tenéis que hacer es intentar minimizar al máximo el abanico de posibilidades. Para ello, veréis los tipos de aplicaciones a los que os vais a enfrentar.

Una vez tengáis claro el tipo de aplicación que queréis hacer, deberéis elegir el tipo de estrategia a utilizar para llevarla al dispositivo. Hay diferentes tipos de estrategias, y dentro de estas estrategias existen muchas alternativas concretas. Veréis también los puntos fuertes y débiles de cada alternativa.

Así, podremos ver las diferentes estrategias para desarrollar en nuestro móvil.

2.2.1. Desarrollos web

Dentro de este subapartado englobamos todas las aplicaciones que están basadas en lenguajes de marcas, lo cual añade la facilidad de poder programar y probar sin necesidad de un emulador o un dispositivo real. A estas aplicaciones se accede directamente mediante la red. Quedan excluidas las aplicaciones que requieren de un preproceso para poder ser distribuidas.

En este punto también se incluyen aplicaciones web basadas en aplicaciones propietarias, como, por ejemplo, Flash y Flash lite. Estas aplicaciones tienen el mismo modelo de desarrollo.

1) Prerequisitos. En general, se puede utilizar cualquier entorno de desarrollo conocido. En el caso de las aplicaciones de tipo *widget* o entornos de ejecución propietarios, los fabricantes suelen dar soporte a estos desarrollos mediante entornos de desarrollo específicos.

2) Fragmentación. La fragmentación en este tipo de aplicaciones existe, aunque suele ser menor que en el resto. Para poder adaptar nuestra aplicación a las capacidades del dispositivo, y dado que estamos en una arquitectura de navegador web, la mejor opción es intentar reconocer el dispositivo cuando se recibe la primera petición. También se puede intentar mostrar de manera diferente la información en el navegador, pero normalmente las capacidades del navegador hacen inviable esta opción.

Una vez que sabemos el dispositivo, tenemos que conocer sus capacidades. Existen varias maneras de adaptar nuestra aplicación a dicho dispositivo:

a) Teniendo el conocimiento (por ejemplo, el proyecto WURLF, que consiste en una base de datos de todos los dispositivos y sus capacidades).

b) Utilizando servidores que incorporen ese conocimiento y lo automaticen, de manera que lleguen a realizar transformaciones automatizadas. Esta opción tiene el inconveniente de que las últimas versiones no suelen estar soportadas.

Ejemplo

A continuación exponemos ejemplo de código de validación de las capacidades de un dispositivo mediante la utilización de la librería de WURLF para reconocer el dispositivo y sus capacidades.

```
public class HelloWorld extends HttpServlet {
  ...
  protected void processRequest(
     HttpServletRequest request,
     HttpServletResponse response)
     throws ServletException, IOException {

        WURFLHolder wurflHolder = (WURFLHolder) getServletContext()
                    .getAttribute("net.sourceforge.wurfl.core.WURFLHolder");

        WURFLManager wurfl = wurflHolder.getWURFLManager();

        Device device = wurfl.getDeviceForRequest(request);

        MarkUp markUp = device.getMarkUp();
        request.setAttribute("markUp", markUp);
        request.setAttribute("device", device);
     ...
  }
}
```

3) Pruebas. Las pruebas se pueden empezar con navegadores de escritorio que soporten HTML 5 o el correspondiente lenguaje de marcado, pero después se deberán hacer pruebas reales, o bien con emuladores. Es muy recomendable que se realicen las pruebas con dispositivos reales. Hay que prestar especial atención a aquellas partes de los estándares que no son soportadas por todos los dispositivos.

En el mundo del desarrollo web existen muchas herramientas para probar las aplicaciones web. La gran mayoría de ellas se puede aplicar a este tipo de aplicaciones.

4) Distribución. La distribución es lo más sencillo, ya que es igual que la distribución de una aplicación web cualquiera. Solo cuando haya cambios incompatibles con los datos guardados "fuera de línea" de una aplicación, habrá que realizar acciones informativas para que el usuario actualice dichos datos.

2.2.2. Entornos de desarrollo nativos

Como ya hemos visto, las aplicaciones nativas son las que ofrecen una mejor experiencia de usuario. Son aquellas que están especialmente diseñadas e implementadas para el contexto de ejecución (plataforma o dispositivo) en el que van a ejecutarse, y pueden sacarle partido a todas las capacidades de dichos dispositivos. En ocasiones, también están sujetas a normas específicas de los fabricantes de dispositivos o responsables de las plataformas.

1) Prerrequisitos. Para poder desarrollar una aplicación nativa, generalmente se necesita el entorno de desarrollo o IDE de cada plataforma. Por ejemplo, para Android necesitamos su SDK, y es recomendable usar Eclipse y añadirle algunos *plugins*. En cambio, en el caso de iOS, necesitamos xCode; para Blackberry *apps* necesitamos también su SDK; para Windows Phone necesitamos Miscrosoft Visual Studio, etc. Estos IDE pueden tener una licencia de pago, la cual dependerá de cada plataforma.

IDE

Integrated development environment (IDE) es un entorno de desarrollo que incorpora todas o casi todas las herramientas de desarrollo necesarias, desde herramientas de modelado o diseño hasta herramientas de *debugging*.

Algunos de estos IDE no son multiplataforma, por lo que requieren de un equipo de desarrollo específico (como es el caso de Windows Phone o iOs).

Estos IDE suelen proporcionar todo lo necesario para cubrir el desarrollo completo de la aplicación, de manera que incluyen los emuladores necesarios para probar nuestra aplicación mientras la desarrollamos.

2) Implementación. Todas las implementaciones son distintas. Cada sistema utiliza su propio método y sus propios patrones, pero hay algunos puntos comunes:

a) Existe un emulador con el que podemos probar nuestras aplicaciones. Sin embargo, en ocasiones el emulador no permite emular todas las acciones de usuarios o la emulación no es lo suficientemente ágil, por lo que necesitamos un dispositivo real.
b) Separación de presentación y lógica, de manera que aprovechemos al máximo los componentes.
c) Posibilidad de "debugar" nuestra aplicación para poder tener mayor control.
d) Generalmente existen herramientas que facilitan la construcción de las interfaces gráficas o UI (*user interface*).

3) Pruebas. Para poder hacer pruebas, cada IDE tiene sus herramientas, desde las típicas tecnologías de pruebas unitarias hasta sistemas más complejos, como el *monkeyrunner* de Android. Para realizar pruebas de estrés de las aplicaciones, también existen herramientas para hacer pruebas de aceptación contra la UI, que utilizan lenguajes de alto nivel, como es el caso de UIAutomation sobre iOs.

Monkeyrunner

Las herramientas *monkeyrunner* y *monkey* sirven para lanzar muchos eventos de usuario sobre el emulador. Son eventos aletorios (por eso se empela la metáfora de un mono).

UIAutomation

UIAutomation sirve para definir en Javascript las acciones de usuario que finalmente se ejecutan en el dispositivo o en el emulador.

Sin duda, las posibles pruebas que se pueden realizar sobre las aplicaciones nativas son mucho más extensas y están más controladas que aquellas que se puedan realizar en otro tipo de aplicación, ya que se tienen las herramientas propias de la plataforma.

4) Firma y distribución. Para poder distribuir la aplicación o, incluso, ejecutarla en un terminal para hacer pruebas, puede ser necesario firmar dicha aplicación con un certificado digital que nos identifique como desarrolladores.

Si la distribución se realiza mediante terceros, como sucede en los mercados de aplicaciones (*market places*), es incluso más necesario para poder acreditar que tenemos el derecho de publicar aplicaciones, así como para hacernos responsables de dichas aplicaciones.

Según la plataforma, los modelos de distribución son simplemente sistemas de descarga, bien mediante sistemas *OTA* (*over the air*), bien mediante los mercados de aplicaciones. Este último sistema está teniendo una gran acogida, ya que permite vender fácilmente la aplicación y llegar a un gran número de usuarios potenciales de forma rápida, aunque podemos encontrarnos con unas fuertes restricciones a la hora de conseguir publicar nuestra aplicación.

2.2.3. Entorno de desarrollo multiplataforma

Como habéis visto, las aplicaciones nativas son muy potentes, pero a la vez requieren de un esfuerzo de desarrollo para soportar solamente una plataforma, y así con cada una de las plataformas que queramos soportar. Para poder soportar todas las plataformas, necesitaríamos saber muchos lenguajes, ya que habría que portar dichas aplicaciones entre plataformas. En concreto, en la actualidad existen al menos cinco lenguajes diferentes, que son necesarios para poder realizar aplicaciones sobre las plataformas más actuales: C, C++, Java, C#, Javascript, Objective-C, además de los diferentes IDE necesarios y sus correspondientes librerías específicas.

En cambio, las aplicaciones web nos permiten llegar a muchas plataformas con un mismo código sin necesidad de portar el código, pero sin poder llevar al usuario la misma experiencia que consigue con las aplicaciones nativas.

Por lo tanto, si existiera la posibilidad de realizar aplicaciones nativas desde una misma línea de código, tendríamos lo mejor de ambas aproximaciones. Aquí es donde entran en juego las estrategias de aplicaciones multiplataforma o *cross-platform*, también conocidas como aplicaciones híbridas.

Desde hace tiempo se está persiguiendo esta "multiplataformidad" con intentos como JavaMe, que no han dado los resultados esperados. La llegada de HTML 5 y la explosión de nuevos *smartphones* han creado nuevas alternativas, que se deben tener en cuenta. Muchas de estas alternativas aprovechan HTML 5 como base y construyen a su alrededor maneras de acceder a las capacidades que HTML 5 no da de partida, prácticamente siempre mediante objetos Javascript. Usar elementos 100% estándar (como HTML 5, CCS y Javascript) ofrece un gran punto a favor, pues se trata de tecnologías muy conocidas.

Las hay que simplemente se quedan en un *wrapper* de la aplicación HTML 5 y añaden estos puntos de acceso. En cambio, otras realizan preprocesamiento para acabar generando aplicaciones 100% nativas. Hay otras alternativas que proporcionan su propia arquitectura y sus propios lenguajes, y también mediante un sistema de compilación o ejecución vía máquina virtual consiguen tener aplicaciones nativas.

Hay aspectos que estas aproximaciones no van a poder evitar fácilmente (a no ser que tengan código condicional específico para cada plataforma). Son los siguientes:

- **Pérdida de controles específicos de una plataforma**: Si tenemos un control de la UI o una funcionalidad concreta que solo existe en una plataforma, no podremos generar de manera única, por nuestro desarrollo multiplataforma.

- **Integración en el escritorio del dispositivo**: Según la plataforma, las posibilidades de añadir elementos en el escritorio de cada usuario varían. Por ejemplo, en Android o Symbian se pueden crear *widgets* potentes para mejorar la el uso de nuestra aplicación, mientras que en Windows Phone solo es posible añadir iconos de la misma.

- **Gestión de la multitarea**: Debido a que se trata de conceptos de bajo nivel de cada plataforma, cada una la trata de manera diferente, con restricciones diferentes, por lo que no será fácil hacer código común para todas sin perder mucha potencia.

- **Consumo de la batería**: Estas aproximaciones requieren de una capa de abstracción sobre nuestro dispositivo, que provoca problemas como la multitarea. De la misma forma, el control sobre el consumo de batería se hace más difícil cuando no se tienen las capacidades concretas de la plata-

forma. También afecta el hecho de no tener el control sobre la multitarea, ya que esta es una de las maneras de ahorrar las baterías.

- **Servicios de mensajería asíncrona o *push services***: Sirven para implementar elementos como la mensajería instantánea, pero debido a que cada plataforma los implementa de una manera, se hace complicado atacarlos conjuntamente.

A continuación exponemos algunos de los ejemplos que existen:

- **PhoneGap**: Son aplicaciones realizadas en HTML 5 con objetos Javascript que permiten el acceso, mediante enlaces, a las funciones nativas para las capacidades que HTML 5 no ofrece. Las aplicaciones se ejecutan sobre un componente que contiene un navegador.

- **Rhodes**: Son aplicaciones escritas en Ruby que utilizan el patrón de diseño MVC[7] y que se ejecutan en máquinas virtuales específicas de Ruby para cada plataforma. Por tanto, son aplicaciones nativas. Incluyen sistemas para sincronizar datos de manera sencilla y conseguir, de esa manera, el cambio de "en línea" a "fuera de línea" sin mucho coste.

[(7)]MVC (modelo vista controlador)

- **Appcelerator**: Son aplicaciones escritas en HTML 5 que son compiladas en aplicaciones nativas. También son capaces de generar aplicaciones de sobremesa clásicas, ejecutables sobre Mac, Linux o Windows.

- **Wacapps**: Son aplicaciones escritas en HTML 5 con soporte para la distribución por parte de las operadoras.

- **Flash**: Son aplicaciones que corren sobre un *player* propietario. Si existe el *player* correspondiente, no necesitan ser portadas. Se escriben de igual manera que las aplicaciones de sobremesa y tienen las mismas restricciones.

- **Java ME**: Son aplicaciones escritas en Java, con todas las peculiaridades de los perfiles J2ME, y se ejecutan mediante una máquina virtual, con sus correspondientes restricciones. Actualmente están en casi el ochenta por ciento de los dispositivos móviles del mercado.

- **Unity3D**: Entorno para desarrollar juegos nativos o web para cada plataforma, como Play Station, Wii, PC, Mac, etc. También da soporte para las Android e Iphone.

Estas son solo algunas de las existentes, pero hay muchas más y, actualmente, siguen saliendo alternativas.

Estos desarrollos nos ofrecen beneficios de los desarrollos previos:

- Solo una línea de desarrollo para varias plataformas.

- Aplicaciones que podemos instalar en nuestros dispositivos, que tienen la posibilidad de ser distribuidas en los mercados de aplicaciones.
- Dependiendo del caso, el resultado son aplicaciones nativas que aprovechan en gran medida el potencial del dispositivo y ofrecen un diseño y experiencia de usuario idéntica a las aplicaciones desarrolladas sólo para una plataforma.

Sin embargo, también existen aspectos negativos:

- En diferentes grados, tienen acceso parcial a los recursos del dispositivo. Esto significa que, según el grado, no pueden interaccionar con algunas capacidades y, en mayor o menor medida, serán aplicaciones menos eficientes que las aplicaciones nativas.

- Para poder soportar varios dispositivos, suelen tener que hacer el mínimo común denominador de las capacidades. Así, si una plataforma aporta una nueva funcionalidad (por ejemplo, un sistema de comunicación que el resto no tenga), no se podrá implementar sin añadir código específico.

- Soporte parcial. No tienen soporte absoluto para todas las plataformas y versiones.

- Las nuevas funcionalidades tardan en ser soportadas. Dado que las novedades en las plataformas aparecen en un ritmo muy elevado y en cortos periodos de tiempo, si se utilizan estos entornos para el desarrollo, es necesario esperar un tiempo para poder utilizar dichas herramientas. En el mejor de los casos, si se trata de código libre o ampliable, podemos realizar una implementación para tener acceso a esa funcionalidad.

- Se requiere pagar la licencia, si la tiene, para el entorno de desarrollo, además de la licencia propia de cada plataforma para poder distribuir la aplicación.

Como veis, esta es sin duda una alternativa a tener en cuenta para decidir qué estrategia de desarrollo queréis para vuestro proyecto web, pero no siempre será la mejor.

1) Prerrequisitos. En general, para conseguir las aplicaciones nativas, cada uno de los entornos proporciona su entorno de desarrollo completo.

En muchos casos, las aplicaciones se prueban en los emuladores de las plataformas nativas, de manera que hay que instalar el IDE propio del entorno de desarrollo y los emuladores o, en ocasiones, los SKD.

SDK

SDK o kit de desarrollo de *software* es el conjunto de herramientas necesarias para realizar el desarrollo de aplicaciones en una plataforma.

2) Implementación. Durante la implementación, variará mucho en función de cada plataforma, pues hay algunas que podrán aprovechar todas las herramientas de desarrollo, otras que no lo necesitarán en exceso y algunas en las que el desarrollo será más difícil.

En general, las herramientas facilitan el proceso de desarrollo, aunque sin llegar al nivel de las herramientas de desarrollo nativas.

3) Firma y distribución. A veces la distribución se puede realizar directamente mediante los IDE de las plataformas de desarrollo, pero en la mayoría de los casos es necesario hacerla mediante los canales habituales para las aplicaciones nativas.

2.3. Métodos aplicados al desarrollo de aplicaciones móviles

En el mundo del desarrollo de *software* existen muchos métodos de desarrollo, cada uno con sus puntos fuertes y sus puntos débiles. En el caso del desarrollo de aplicaciones móviles sucede lo mismo, y cuando os planteéis qué método elegir deberéis saber escoger en función de vuestras necesidades.

Algunos de los métodos más conocidos son los siguientes:

- modelo *waterfall*
- desarrollo rápido de aplicaciones
- desarrollo ágil (cualquiera de sus variantes)
- Mobile-D

Una de las características importantes de la gran mayoría de los desarrollos móviles es su corta duración. Esto se debe a factores como la gran competencia en el sector, los cambios en el mismo con la aparición, casi constante, de novedades tanto *software* como *hardware*, el hecho de que muchas aplicaciones nacen con un desarrollo precoz en forma de prototipo (y van evolucionado después) o incluso la simplicidad de las aplicaciones, que no requieren grandes desarrollos. Esta suele ser, salvo algunas excepciones, la norma de los desarrollos de aplicaciones para dispositivos móviles.

"It can take up to nine months to deploy an entertainment (mobile) application, but that's the duration of a cell phone in this market."

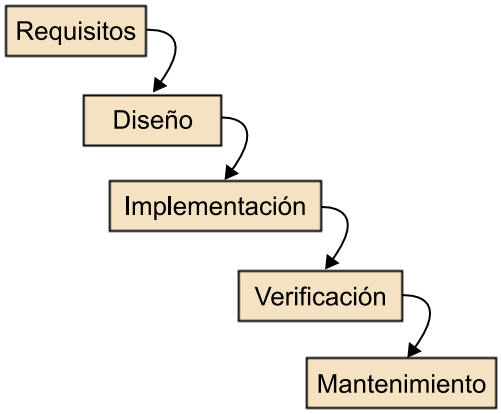
Craig Hayman (IBM)

2.3.1. Modelo *waterfall*

El modelo *waterfall* es el modelo más estático y predictivo. Es aplicable en proyectos en los que los requisitos están fijados y no van a cambiar durante el ciclo de vida del desarrollo. Esta aproximación divide el proyecto en fases

estancas totalmente secuenciales. En este modelo, el desarrollo se interpreta como el agua que va cayendo de un estanque al siguiente. Se le da mucho énfasis a la planificación, a los tiempos, a las fechas límite y al presupuesto.

Ejemplo de fases de un proyecto de *software* en un modelo *waterfall*



En el contexto del desarrollo de aplicaciones móviles, el modelo *waterfall* puede ser aplicable a proyectos realmente controlados y previsibles, en los que no hay mucha incertidumbre por lo que se desea hacer y para los que no son importantes los cambios constantes en la industria.

2.3.2. Desarrollo rápido de aplicaciones

El desarrollo rápido de aplicaciones es un método de desarrollo iterativo cuyo objetivo es conseguir prototipos lo antes posible para mejorarlos después, poco a poco. Se suele priorizar la implementación sobre la planificación, y se utilizan muchos patrones de diseño conocidos para poder adaptarse de la mejor manera a cambios en los requerimientos.

El desarrollo rápido de aplicaciones es un método muy útil para el desarrollo de proyectos realmente urgentes con tiempos de entrega muy cortos.

2.3.3. Desarrollo ágil

El desarrollo ágil es un modelo de desarrollo basado en iteraciones, donde en cada iteración se realizan todas las fases del ciclo de desarrollo.

Manifiesto ágil

El manifiesto ágil fue publicado en el 2001 por diecisiete desarrolladores de *software*, quienes representaban entonces los métodos de desarrollo más populares, que pasarían a conocerse como ágiles (Extreme Programming, Crystal Clear, DSDM o ASD, entre otros). El manifiesto define doce principios y cuatro valores éticos para los desarrolladores. Podéis consultarlo en http://agilemanifesto.org.

El desarrollo ágil se basa en los principios del manifiesto ágil y sus valores éticos, que tratan de dar más valor a algunos conceptos, pero sin dejar de lado los demás. Son los siguientes:

1) Dar más valor a los individuos y a sus interacciones que a los procesos y herramientas.
2) Dar más valor al *software* que funciona que a la documentación exhaustiva.
3) Dar más valor a la colaboración con el cliente que a la negociación contractual.
4) Dar más valor a la respuesta al cambio que al seguimiento de un plan.

Con estos valores se intenta conseguir, entre otras cosas, entregar algo lo más pronto posible y evitar problemas originados por cambios de requisitos. Esto es muy apropiado para proyectos cambiantes, ya sean grandes o pequeños, ya que mediante estos valores se pueden mitigar los riesgos. Para conseguir proyectos que puedan cambiar fácilmente, se pone especial atención en la calidad de los productos conseguidos, cosa que es realmente importante en proyectos de *software* para dispositivos móviles. Para conseguir esto, se basan en las pruebas de la aplicación y, a menudo, las automatizan.

Métodos derivados

Existen varios métodos derivados de este modelo de desarrollo ágil, como Scrum, Kanban, Lean, AUP, Extreme Programming, etc.

Los métodos ágiles suelen ser muy adecuados para el desarrollo de aplicaciones móviles por las siguientes razones:

- **Alta volatilidad del entorno**: Con cambios en entornos de desarrollo, nuevos terminales y nuevas tecnologías a un ritmo mucho más elevado que en otros entornos de desarrollo.

- **Equipos de desarrollo pequeños**: Dado que los desarrollos móviles suelen ser proyectos relativamente pequeños, los equipos no suelen ser muy grandes. Generalmente son llevados a cabo por desarrolladores individuales o por PYME.

- ***Software* no crítico**: No suelen ser aplicaciones de alto nivel de criticidad, dado que suelen ser aplicaciones para entretenimiento o gestión empresarial no crítica.

- **Ciclos de desarrollo cortos**: Dada la evolución constante de la industria, se requieren ciclos de vida realmente cortos para poder dar salida a las aplicaciones a tiempo.

2.3.4. Mobile-D

El método Mobile-D se desarrolló junto con un proyecto finlandés en el 2004. Fue realizado, principalmente, por investigadores de la VTT (Instituto de Investigación Finlandés) y, a pesar de que es un método antiguo, sigue en vigor (se está utilizando en proyectos de éxito y está basado en técnicas que funcionan).

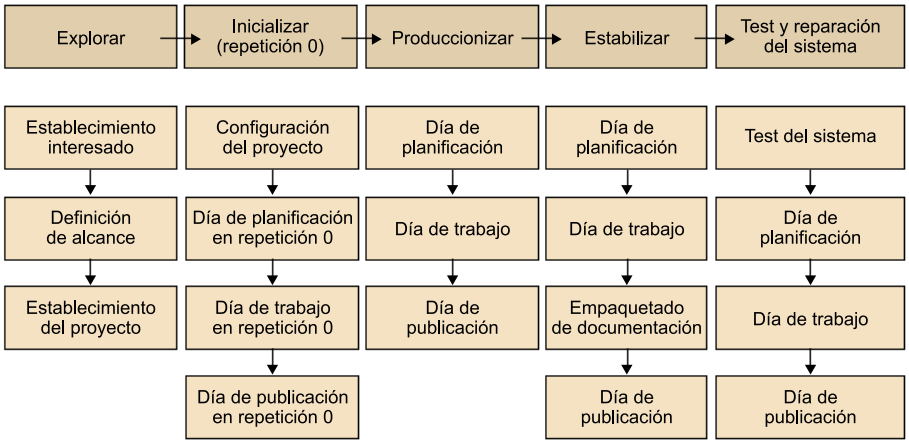
Enlace de interés

Para saber más sobre Mobile-D, podéis visitar la siguiente página web de la VTT: http://agile.vtt.fi/mobiled.html.

El objetivo es conseguir ciclos de desarrollos muy rápidos en equipos muy pequeños (de no más de diez desarrolladores) trabajando en un mismo espacio físico. Según este método, trabajando de esa manera se deben conseguir productos totalmente funcionales en menos de diez semanas.

Se trata de método basado en soluciones conocidas y consolidadas: Extreme Programming (XP), Crystal Methodologies y Rational Unified Process (RUP), XP para las prácticas de desarrollo, Crystal para escalar los métodos y RUP como base en el diseño del ciclo de vida.

Ciclo de desarrollo de Mobile-D



Cada fase (excepto la inicial) tiene siempre un día de planificación y otro de entrega. Las fases son:

- **Exploración**. Se dedica a la planificación y a los conceptos básicos del proyecto. Es diferente del resto de fases.

- **Inicialización**. Se preparan e identifican todos los recursos necesarios. Se establece el entorno técnico.

- **Productización o fase de producto**. Se repiten iterativamente las subfases, con un día de planificación, uno de trabajo y uno de entrega. Aquí se intentan utilizar técnicas como la del *test driven development* para conseguir la mayor calidad.

Test driven development

El *test driven development* (TDD) o desarrollo dirigido por las pruebas, indica que antes de realizar una funcionalidad debe existir una prueba que verifique su funcionamiento.

- **Fase de estabilización**. Se llevan a cabo las acciones de integración para asegurar que el sistema completo funciona correctamente.

- **Fase de pruebas y reparación**. Tiene como meta la disponibilidad de una versión estable y plenamente funcional del sistema según los requisitos del cliente.

2.4. Fases de los proyectos de desarrollo de aplicaciones móviles

Hemos visto que hay varios métodos existentes en el mercado para el desarrollo de aplicaciones móviles y que todos ellos se dividen en diferentes fases. Cada uno de estos métodos especifica lo que se debe hacer en cada fase, así como el nivel o los resultados que se requieren. Estas fases del desarrollo de aplicaciones móviles tendrán problemas comunes y soluciones comunes, de modo que vamos a repasarlas.

2.4.1. Planificación

En la fase de planificación se intenta distribuir el tiempo y los recursos necesarios para poder llevar a cabo el proyecto, ya sea en una única planificación completa o en planificaciones más divididas.

Durante las fases de planificación siempre se intenta conseguir la máxima precisión de las estimaciones, para minimizar los riesgos asociados al proyecto. En el caso de los proyectos de aplicaciones móviles, es necesario que tengáis cuenta los siguientes riesgos implícitos:

- **Dificultades por el desconocimiento de la tecnología**. Suele tratarse de tecnologías nuevas, en ocasiones desconocidas para los desarrolladores. Eso repercute en el tiempo de aprendizaje y puede ocasionar desviaciones por contratiempos no conocidos

- **Disponer de dispositivos reales**. Para poder realizar un buen proyecto de desarrollo de aplicaciones móviles, es necesario poder probarlo en dispositivos reales. Para ello, se debe planificar una fase de pruebas reales.

- ***Time to market***. A la hora de planificar hay que tener muy en cuenta el que suele tardar una idea en llegar al mercado e intentar reducirlo al máximo, pues la competencia es muy grande.

- **Prototipado**. Es importante planificar cuando se va a conseguir el primer prototipo (y, tal vez, la primera salida en beta), pues el prototipado rápido puede ser muy útil para este tipo de aplicaciones.

2.4.2. Toma de requisitos

Como en todo proyecto de *software*, es necesario que conozcáis los requisitos (tanto los funcionales como los no funcionales). En cuanto a los requerimientos no funcionales, deberéis tener muy presente aspectos relacionados con el uso como, por ejemplo:

- ¿Quien va a ser nuestro usuario? ¿En qué momento va a utilizar nuestra aplicación?
- ¿Qué requerimientos mínimos de *hardware* son necesarios?
- ¿Necesitamos gestionar el modo "en línea" y "fuera de línea"?
- ¿Deben existir datos de terceros? (desde un mapa, hasta datos del propio dispositivo).

Una vez tengáis todos los requisitos, es importante que dediquéis un tiempo a intentar ordenarlos, lo que os ayudará a tomar decisiones.

Un objetivo importante de esta fase es conseguir el plan de dispositivos o *device plan*.

Plan de dispositivos (*device plan*)

El plan de dispositivos o *device plan* es un listado ordenado de todos los dispositivos o grupos de dispositivos que se desean soportar.

Para elaborar el plan de dispositivos, agrupamos los dispositivos cuyo desarrollo se pueda atacar conjuntamente (por ejemplo, todos los que tengan Android 2.1 o superior o todos los que tengan GPS). A partir de ahí, se le da un valor de negocio a cada clase de dispositivo, así como el coste asociado al desarrollo de la aplicación para ese grupo. Finalmente, se consiguen los beneficios esperados de cada clase, tal como os mostramos a continuación:

	Coste estimado	Ingresos estimados	Beneficios previstos
Dispositivos de clase A	1	5	4
Dispositivos de clase B	2	4	2
Dispositivos de clase C	3	3	0
Dispositivos de clase D	4	2	-2
Dispositivos de clase E	5	1	-4

A la hora de decidir el coste, debéis añadir todos los posibles costes técnicos o no técnicos que puedan deberse a los requerimientos asociados.

Ejemplos

Si realizáis un juego 3D, debéis añadir el coste asociado al desarrollo en el caso de alguna plataforma no tenga librerías conocidas para ese tipo de modelado gráfico. Si queréis realizar una aplicación para gestión empresarial de inventario, por ejemplo, y realizáis la aplicación para Android y iOs, en caso de querer soportar los *tables PC* correspondientes, debéis tener en cuenta el coste que conlleva adaptar la interfaz de usuario.

Definición de la arquitectura

Siempre que se tiene un aplicación (móvil o no), existen varias opciones de arquitecturas. En el caso de los dispositivos móviles, hay aún más alternativas. Ya hemos visto que existe la posibilidad de tener sitios web móviles o aplicaciones web móviles o, cómo no, aplicaciones nativas, pero esto es solo una parte del problema, pues existen muchas arquitecturas de aplicación posibles en lo que respecta a las aplicaciones móviles.

En ocasiones, para poder tomar una buena decisión sobre la arquitectura, es necesario realizar pequeños prototipos. Esto dependerá del tamaño del proyecto y del conocimiento de la tecnología.

Sin duda, decidir la arquitectura de vuestra aplicación os ayudará a saber qué tipo de desarrollo vais a realizar (una aplicación web, una aplicación nativa o, tal vez, una híbrida). Por tanto, si no tenéis clara la arquitectura, muchas de las fases siguientes del proyecto pueden verse afectadas.

A continuación repasamos las arquitecturas más habituales en el desarrollo de aplicaciones para dispositivos móviles, como las aplicaciones "en línea", "fuera de línea" o las aplicaciones de sincronización. Finalmente repasaremos las aplicaciones que necesitan conexión entre dispositivos.

1) Aplicación "fuera de línea"

La aplicaciones "fuera de línea" son aplicaciones que, una vez descargadas, no requieren en absoluto de conexión (a excepción de las actualizaciones) para poder funcionar. Estas aplicaciones solo necesitan desarrollar la aplicación del dispositivo móvil (no son necesarios más componentes).

Tienen como ventaja que se pueden utilizar tanto con conexión como sin ella, pero suelen ser aplicaciones a las que, una vez instaladas, se les pierde el rastro.

Ejemplos de aplicaciones "fuera de línea"

Os mostramos algunos ejemplos de aplicaciones "fuera de línea":

- Muchos tipos de juegos que no comparten ningún tipo de información.
- Aplicaciones de productividad (como el gestor de tareas o las alarmas).

2) Aplicación totalmente "en línea"

Las aplicaciones totalmente "en línea" son aplicaciones que no pueden funcionar sin conexión a Internet. Estas arquitecturas requieren, sin lugar a dudas, de una parte de servidor, y están pensadas para mantener una comunicación constante con dicha parte servidora.

Tienen como desventaja que el usuario no puede utilizar la aplicación cuando no tiene conexión, pero disponen de información constante de las interacciones del usuario. Es necesario desarrollar, al menos, la parte servidora, tal vez una parte de desarrollo en el cliente y, en ocasiones, la comunicación entre ambos. Al necesitar estar siempre conectados, tienen un consumo extra de batería.

En ocasiones no es necesario realizar la parte servidora, ya que se trata de aplicaciones llamadas *mash-ups*, que son aplicaciones que aprovechan API existentes en la red para interaccionar con datos, (como, por ejemplo, la API de Twitter o datos públicos del estado).

Ejemplos de aplicaciones totalmente "en línea"

Os mostramos algunos ejemplos de aplicaciones totalmente "en línea":

- Todas las aplicaciones que se pueden utilizar en el móvil mediante el navegador web correspondiente (redes sociales, accesos a *webmails*, etc.).
- Aplicaciones web móviles nativas, que son aplicaciones web pero tienen apariencia y características de una aplicación nativa.
- Aplicaciones nativas que requieren de la conexión para poder estar autentificados o para obtener los datos.
- Aplicaciones de redes sociales o de tiempo real en las que la conexión es prácticamente imprescindible para tener la información actualizada.
- Cualquier tipo de *chat*, aplicación de llamadas o videoconferencias.

3) Aplicaciones de sincronización

Las aplicaciones de sincronización son aplicaciones que pueden funcionar en ambos modos, "en línea" y "fuera de línea", y permiten realizar las mismas acciones o acciones muy parecidas en ambos casos. La aplicación debe sincronizar los datos de la situación "fuera de línea" cuando se encuentre "en línea" y gestionar los posibles conflictos. Esto supone un beneficio para el usuario, ya que le permite trabajar en cualquier lugar y tener la información lo más actualizada posible.

La sincronización puede darse en un servidor propio, o bien con objetos o API para la sincronización en la nube, de modo que facilita su desarrollo y reduce costes. Hay diferentes tipos de sincronización:

- **Unidireccional**. Son aplicaciones que pueden sincronizar los cambios con algún servidor externo, bien porque en uno de los dos extremos solo es posible leer información, bien porque sirven como *backup* de información.

- **Bidireccional**. Son aplicaciones que pueden sufrir modificaciones tanto en el servidor como en el cliente, y la parte servidora y cliente se deben comunicar para realizar la sincronización.

Ejemplos de sincronización bidireccional

Os mostramos algunos ejemplos de sincronización bidireccional:

- Envío de estadísticas de juegos o de aplicaciones deportivas.
- Una información de visualización de mapas o de hojas de ruta para repartidores.
- Aplicaciones de envío y recepción de correo electrónico.
- Calendarios que se puedan modificar externamente a la aplicación móvil.
- Aplicaciones de suscripción y lectura de agregadores de contenidos (RSS)

4) Aplicaciones para la comunicación entre dispositivos

Las aplicaciones para la comunicación entre dispositivos son aplicaciones que interconectan dos (*unicast*) o más (*multicast*) dispositivos e intercambian información.

Este tipo de aplicaciones requieren desarrollar la parte del cliente además de la comunicación y la recepción de la información. También es necesario desarrollar la fase de búsqueda y enlace entre los dispositivos. Este tipo de aplicaciones suelen utilizar comunicaciones WPAN, como Bluetooth o NFC.

Ejemplos de aplicaciones para la comunicación entre dispositivos

Os mostramos algunos ejemplos de aplicaciones para comunicación entre dispositivos:

- aplicaciones de intercambio de contactos
- juegos entre dos o varios jugadores

2.4.3. Especificación y diseño

A diferencia de lo que sucede con otras fases del desarrollo de aplicaciones, la especificación no tiene grandes diferencias con respecto a las aplicaciones de sobremesa normales. Únicamente hay que tener en cuenta que, en muchas ocasiones, la fase de especificación se solapa con la del diseño.

En cuanto al diseño, existen algunos patrones de diseño ampliamente conocidos en el desarrollo de aplicaciones, que suelen ser implementados en las aplicaciones para dispositivos móviles:

- ***Model-View-Controler*** **(MVC)**. Se utiliza para poder separar al máximo la lógica de la visualización e interacción, y así poder dar soporte a más escenarios, como puede ser el caso de una aplicación para *smartphone* y la misma para *tablet PC*.

- ***Threading***. Se refiere al uso de hilos en segundo plano para realizar tareas largas que bloqueen al usuario.

- ***Delegation***. Se trata de delegar una parte del trabajo hacia otro objeto sin que este tenga que ser una subclase del primero. En el caso de los desarrollos móviles, es muy útil para delegar trabajos relacionados con la interfaz, de manera que pueda trabajar con eventos de manera muy sencilla y sostenible.

- **Modelo de memoria gestionada**. En general, las aplicaciones no se ejecutan directamente sobre la plataforma, sino que suele haber una capa intermedia o *middleware*, y esta suele gestionar la memoria. Sin embargo, para poder hacerlo de manera eficiente, es necesario realizar algunas acciones o convenciones.

Además de los patrones orientados a conseguir un *software* de mejor calidad, también existen patrones de diseño para maximizar el uso de nuestra aplicación. Estos patrones deben ser interpretados de manera diferente en función de la plataforma para la que desarrollemos, pues al final intentan utilizar los comportamientos comunes al resto de aplicaciones, y puede que estos comportamientos varíen según la plataforma.

Ejemplo

En Android se recomienda hacer que las navegaciones mediante pestañas se sitúen en la parte superior de la pantalla, mientras que en iOs se recomienda que estén en la parte inferior.

En general, existen unos principios básicos aplicables a todo diseño; por ejemplo, dar más valor a lo claro que a lo simple y al contenido que al continente. Para ello, existen algunos patrones de diseño de la UI (interfaz de usuario) que ayudan a resolver problemas conocidos y que han sido probados en numerosas ocasiones.

Entre los más conocidos se encuentran Dashboard, ActionBar, Dynamic List, Pager, *popups*, *alerts*, *etc*.

A continuación os explicamos algunos de ellos. Exponemos el problema que solucionan y cómo lo solucionan, así como algunas recomendaciones a tener en cuenta.

Cuadro de control (*dashboard*)

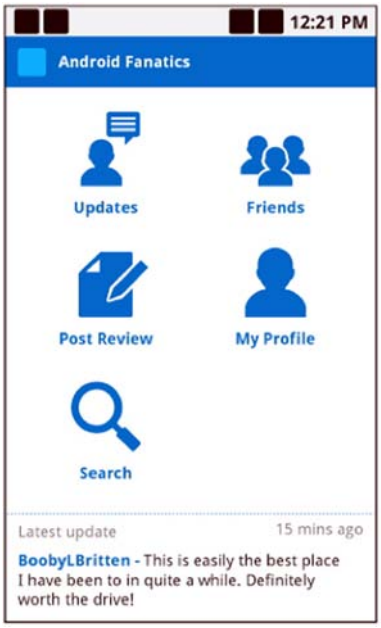
Problema: Acceder de manera rápida, clara y sencilla a las funcionalidades principales de la aplicación. En el caso del móvil, es importante por la responsabilidad que el usuario requiere.

Solución: Tener una página de llegada con la información clara de la última información de la aplicación y las acciones más importantes.

Se recomienda:

- Resaltar lo que es nuevo.
- Enfocar de tres a seis características importantes.

Ejemplo de patrón de interfaz de cuadro de control



Barra de acción (*action bar*)

Problema: La limitación de espacio de las pantallas hace pueda ser muy complicado mostrar las acciones que el usuario puede hacer en una pantalla, y puede provocar mucha pérdida de espacio útil si se introduce un objeto visual botón por cada acción.

Solución: Se agrupan todas las acciones que se pueden hacer en la pantalla actual en una zona, en la parte superior o inferior (dependiendo de la plataforma), para aprovechar mejor el espacio y tener una mayor cohesión entre aplicaciones. Además, se puede aprovechar este espacio para indicar el lugar de navegación donde nos encontramos.

Ejemplo de patrón de barra de acción

Menús contextuales (*quick action menu*)

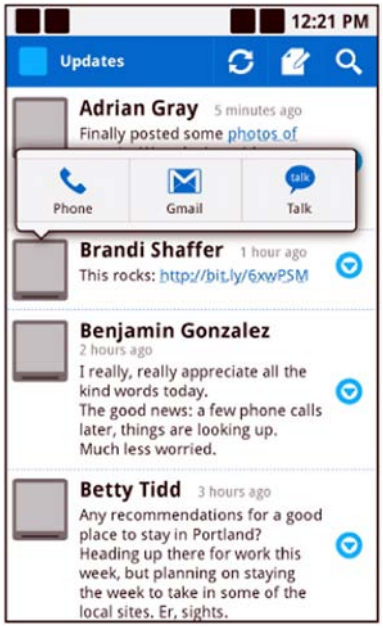
Problema: Las limitaciones de espacio para las acciones propias de un elemento de la pantalla ocuparían, potencialmente, mucho espacio.

Solución: Se muestra un menú contextual al pulsar el objeto, generalmente en la imagen (por ejemplo, en las imágenes de los contactos), y así se evita tener que añadir más ruido a la interfaz. Dentro de este menú se pueden agrupar todas las acciones correspondientes al objeto en cuestión.

Se recomienda:

- Usarlo solo para las acciones más obvias e importantes
- Usarlo cuando el objeto no tenga una vista especialmente diseñada para él. En ese caso, hay que ir a esa vista y mostrar allí la información.
- No usar en contextos donde se soporten múltiples selecciones.

Ejemplo de uso de patrón de menú contextual

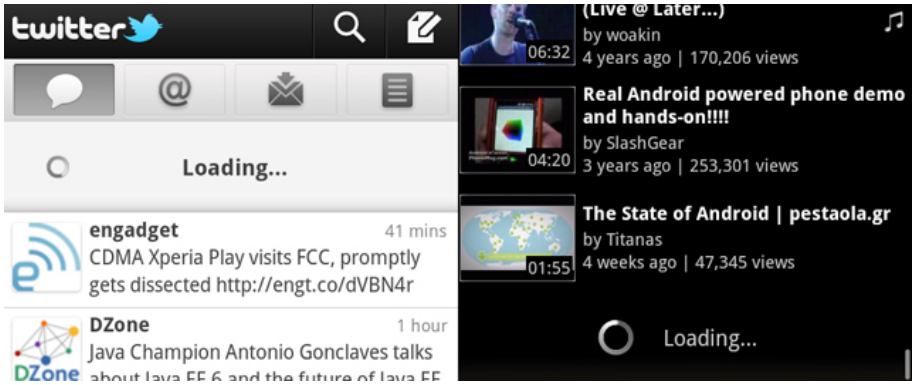


Lista dinámica (*dynamic list*)

Problema: Cargar muchos datos en una lista puede ser muy lento, especialmente si hay problemas de red. Para el usuario puede suponer una mala experiencia si tiene que esperar a que se cargue toda una lista (que en ocasiones no acaba o es muy grande).

Solución: En lugar de esperar a que la lista se cargue, se pueden mostrar los datos relevantes y cargar la lista inmediatamente. Los datos que faltan se pueden cargar cuando el usuario lo pida, o bien cuando la aplicación prevea que va a ser así (por ejemplo, cuando se desplace hacía el final de la lista).

Ejemplo de uso de patrón de lista dinámica



Mensajes de alerta

Problema: Sucede un evento en el sistema o la aplicación que es relevante para el usuario.

Solución: Mostrar un mensaje de alerta, que llama la atención del usuario, para advertirle de la situación. Por ejemplo, cuando hay problemas de batería o cuando nuestra aplicación ha detectado una perdida de conexión.

Ejemplo de patrón de mensaje de alerta

Se recomienda:

- Usarlo solo para mensajes importantes y necesarios. Este tipo de mensajes son muy invasivos, y el usuario, en lugar de sentirse agradecido por recibirlos, puede acabar ignorándolos.

- Realizar alguna acción cuando sea necesario; mostrar una de las opciones como atajo para dicha acción. Por ejemplo, si hay un problema de espacio en el disco, una de las acciones irá al gestor de aplicaciones instaladas para eliminar las que no necesitemos.

2.4.4. Implementación y pruebas

La implementación de aplicaciones para dispositivos móviles se asemeja mucho a la del resto de aplicaciones, aunque, generalmente, se trata de aplicaciones más pequeñas, o bien que tienen ciclos de desarrollo más cortos que las aplicaciones tradicionales. Esto se debe tanto a la propia naturaleza de la aplicación como a las necesidades del mercado que demanda conseguir prototipos o pruebas de concepto rápidas.

Una particularidad de las pruebas es la necesidad de tener un emulador o entorno de pruebas para poder probar aquello que estamos desarrollando. Esta necesidad provoca dificultades (no se puede realizar la prueba si no se dispone de un dispositivo) y ser lentas (por la lentitud de ejecución sobre los dispositivos o emuladores). Por este motivo, se acude a las pruebas unitarias que permiten dividir el desarrollo, con lo que se puede probar de manera desacoplada y desarrollar partes de nuestra aplicación sin la necesidad de pruebas en el emulador, como pueden ser los accesos a bases de datos o bien la lógica de negocio.

De cara a la implementación, hay una serie de factores que toman especial importancia en el caso de las aplicaciones para dispositivos móviles, y serán primordiales para el éxito del proyecto:

- **Usabilidad**. La usabilidad de la aplicación se debe tener muy en cuenta (debemos tener presente que la mayoría de usuarios no tendrán tiempo ni ganas de leer los manuales). Una buena práctica para mejorar la usabilidad es adaptar las aplicaciones a los comportamientos estándares de la plataforma, de manera que el usuario pueda aprovechar reglas mnemotécnicas o hábitos adquiridos.

- **Responsividad**. La aplicación debe responder a las acciones de usuario lo mejor posible y de manera ágil. Es un punto muy importante pues el usuario de dispositivos móviles suele ser más exigente que el de aplicaciones tradicionales en situaciones parecidas. Algunas plataformas, como Android, son muy estrictas a la hora conseguir que la aplicación sea suficientemente responsiva. Si la aplicación no es suficientemente ágil a la hora de responder al usuario, el sistema debe avisarle de esta circunstancia y permitir que la cierre inmediatamente.

Ejemplo de no responsividad

Una aplicación que, mientras lleva a cabo una tarea costosa, bloquea la interfaz de usuario, y además, no le informa del hecho.

- **Optimización de recursos**. Los dispositivos móviles, incluso los actuales, cuentan con unos recursos mucho más reducidos que las aplicaciones de sobremesa o las páginas web. Esto quiere decir que tenemos que hacer un buen uso de la memoria y del procesador del dispositivo, por lo que es conveniente cerrar los recursos que no se necesiten para evitar los problemas asociados.
  También es importante prestar mucha atención al consumo de batería de la aplicación. Por ello, es recomendable evitar cálculos excesivos (como los cálculos de coma flotante), el uso de las funciones de vibración o el uso de las conexiones inalámbricas. Cada una de estas recomendaciones se debe adaptar a la plataforma específica donde trabajamos, pues los fabricantes nos darán las recomendaciones concretas.
  Otra buena práctica, en este sentido, es perfilar la aplicación con las herramientas propias de la plataforma para encontrar problemas.

- **Accesibilidad de la aplicación**. Debemos tener en cuenta al diseñar la aplicación que los usuarios pueden necesitar acceder a ella de diferentes maneras. Por ejemplo, si tenemos una aplicación con formularios, es ideal que sea accesible tanto a través de las pantallas táctiles como de los *trackballs*.

Nota

Existen herramientas que traducen la aplicación a texto, haciéndola accesible a personas ciegas. Para ello, se debe desarrollar de manera correcta.

Importancia de las pruebas unitarias

Las pruebas unitarias sirven, como ya hemos comentado, para probar el correcto funcionamiento de una parte del código. Estas pruebas tienen como características más destacadas, que han de ser automatizables (no es obligatorio, pero sí muy recomendable), completas, reutilizables o repetibles a lo largo del tiempo, independientes entre si y tan profesionales como el propio código.

Si la prueba se centra en una parte de la aplicación que depende del dispositivo en qué se ejecuta, se la denomina prueba de integración.

Las pruebas unitarias agilizan el desarrollo porqué se centran en una parte del desarrollo y, por tanto, no será necesario probar dicha unidad dentro del emulador o emuladores, siempre que se trate de una parte realmente unitaria. Además, refuerzan la fiabilidad del desarrollo porqué se realiza al mismo tiempo que la prueba que lo verifica. Y las ventajas se multiplican cuando la prueba es automatizada pues evita la aparición de errores en un futuro al probar la aplicación de manera más exhaustiva.

Pruebas de integración contextualizadas

Las pruebas que realicéis deben estar contextualizadas; es decir, deben reproducir lo que realmente le está pasando al usuario cuando utiliza nuestra aplicación. Así, no es lo mismo probar un gestor de rutas para el coche que probar una aplicación para controlar los gastos de la empresa, pues se usan en momentos y con objetivos muy distintos.

Hay una serie de preguntas que se debe hacer la persona que pruebe la aplicación para poder realizar correctamente estas pruebas. Por ejemplo:

- ¿Cuál es la experiencia del usuario con la aplicación?
- ¿Se carga rápidamente la aplicación? ¿Se necesita alguna barra de progreso de la aplicación? ¿Y con conectividad reducida?
- ¿Cambia la aplicación al mover el dispositivo? Es decir, ¿se deben tener en cuenta los sensores de acelerómetros? ¿Reacciona con agilidad a estos cambios?
- ¿Acepta correctamente la aplicación las interrupciones como, por ejemplo, las llamadas telefónicas? ¿Acaba correctamente la aplicación? Es decir, ¿cierra correctamente todos los recursos utilizados?
- En caso de utilizar conectividad, geolocalización o cualquier otro servicio, ¿cuál es el comportamiento esperado de la aplicación ante la falta de dicha capacidad?

Para poder responder a este tipo de preguntas se deberán generar una serie de pruebas que llevaran a contextualizarla. Sin estas pruebas, la aplicación puede no funcionar bien en muchos casos. La mejor manera de llevar a cabo estas pruebas es, sin duda, sobre los propios dispositivos. El tipo, número y diversidad de las pruebas dependerá de los requisitos de la aplicación, pero se hace totalmente imprescindible haber realizado pruebas con dichos dispositivos antes de poder distribuir la aplicación.

Ejemplo

http://www.perfectomobile.com/ es un ejemplo de servicio que permite probar aplicaciones sobre dispositivos reales, a través de una granja de los mismos. Además, permite automatizarlas, lo que redunda en una mejora de la calidad.

Continuidad de las pruebas

Como en cualquier desarrollo, es importante que realicéis pruebas en cada nuevo desarrollo, pero en este caso lo es más que en otros entornos con mayor facilidad de cambio, pues generalmente (y en especial con aplicaciones nativas) los despliegues de la aplicación y de sus actualizaciones no suelen estar controlados por los desarrolladores. Esto quiere decir que cualquier cambio, *bug* o mal funcionamiento tiene mayor repercusión y se debe ir con mucho cuidado para evitar la reaparición de problemas ya solucionados. Una manera de combatir este problema es mantener un plan de pruebas y ejecutarlo en cada nueva versión.

Si se trata de una aplicación web para móviles, es mucho más fácil hacer cambios. En una aplicación nativa, las nuevas versiones pueden ser actualizadas mediante el propio sistema, dependiendo del entorno operativo (OC).

Es importante tener sistemas que aseguren la fiabilidad y la no reaparición de errores antes de lanzar una nueva versión, ya sean automatizados (como los sistemas de integración continua) o manuales (como planes de pruebas completos y reproducibles).

3. Negocio

Uno de los aspectos que están favoreciendo el crecimiento del desarrollo de aplicaciones móviles es la facilidad para poder hacer negocio con ellas. Rentabilizar una idea es, actualmente, más fácil que hace unos años.

Esto se ha conseguido gracias a la mejora del canal de distribución, principalmente con la aparición de los mercados de aplicaciones, que facilitan su distribución y compra.

Así, anteriormente los negocios en dispositivos móviles se basaban en la venta directa de aplicaciones a medida o en servicios de micropagos con SMS. En cambio, hoy en día han aparecido nuevos modelos de negocio que han sabido aprovechar la presencia de nuestro dispositivo móvil en nuestro día. Por ejemplo el pago en aplicaciones móvil por bienes virtuales o el pago por suscripción.

Con el fin de que entendáis las posibilidades existentes, en este apartado os mostramos una clasificación de los posibles negocios que se pueden realizar con las aplicaciones para dispositivos móviles. Después veremos con más detalle cómo son algunos de estos negocios, y pondremos especial atención en aquellos de reciente aparición. También veremos los pasos necesarios para conseguir publicar aplicaciones que se venden y distribuyen a través de mercados de aplicaciones, y las recomendaciones que son importantes para llegar a más usuarios.

3.1. Posibilidades de negocio

Existen muchas maneras de hacer negocio con las aplicaciones móviles. Entre los modelos clásicos de negocio con el móvil, destacaremos los siguientes, aunque hay que tener en cuenta que alguno de ellos ha sufrido una caída en las ventas, en parte por el abuso (principalmente mensajería) y la saturación del mercado:

- SMS premium. Mensajes de texto con un coste extra, también conocidos como servicios de tarificación adicional (STA).
- SMS para descargar aplicaciones o melodías.
- Desarrollo de aplicaciones a medida.

Nota

En el 2009 entra en vigor el código de conducta de los STA, que, entre otras medidas para regular el sector, obliga a todos los servicios basados en mensajes premium a avisar al usuario del coste (siempre que sea mayor de 1,2 euros). Este hecho se conoce como *opt-in*.

En contrapartida, han aparecido nuevos modelos de negocio, entre los cuales destacamos los siguientes:

- Vender la aplicación a través de una tienda de aplicaciones. Es el modelo más clásico, aunque no siempre es el más eficaz.

Terminología

A las aplicaciones gratuitas que tienen una versión de pago se las suele conocer como versión *lite*, y en ocasiones, a la versión de pago, como versión *pro*.

- Ofrecer versión gratuita, y con su distribución conseguir marketing de la aplicación o de los servicios relacionados, como puede ser una web inmobiliaria.

- Ofrecer una versión gratuita y una versión de pago. La versión gratuita no contiene todas las funcionalidades de la aplicación, de esta manera, el usuario puede probarla y, si le convence y quiere obtener las funcionalidades extras, deberá pagar por ellas.

- Vender publicidad. Diversas tecnologías permiten incluir en las aplicaciones, tanto en las basadas en tecnología web como en las nativas, anuncios contextualizados. En ocasiones la publicidad sólo aparece en la versión *lite*.

- Vender packs de contenidos extras. Se trata de vender la aplicación por partes, cada una con un coste. Esto es muy habitual en los juegos, por ejemplo, que ofrecen packs con distintos niveles.

- Llevar un negocio existente al mundo móvil:
  - Mejorar una línea de negocio, por ejemplo se hace añadiendo la localización geográfica a nuestro servicio.
  - Intentar conseguir nuevos clientes. Por ejemplo, conseguir clientes compulsivos; es decir, aquellos que si tuvieran que ir al ordenador no comprarían, pero teniéndolo en el bolsillo sí.

Ejemplo

Pensemos, por ejemplo, en una aplicación de información meteorológica o de tráfico, ofrecida vía web, que concrete la información en función de la localización del usuario.

- Construir una aplicación como un servicio (como, por ejemplo, Gootaxi, que permite acceder al servicio de taxis a través del móvil).

Ejemplo

Pensemos, por ejemplo, en información turística durante un viaje, o en información de descuentos del comercio donde está el usuario.

- Construir nuestra aplicación como una suscripción. Por ejemplo, el *streaming* de partidos de la NBA que ofrece el portal http://www.nba.com.

- Movilizar una tecnología existente. Por ejemplo, un CRM.

- Crear una aplicación que extienda un negocio "en línea", aprovechando principalmente las API públicas de Google, Twitter, Idealista, etc.

- Aplicaciones con sistemas *pay-in*. Esto significa que la aplicación puede pedirte pagos a cambio de opciones de la aplicación mientras está ejecutándose. Esto puede funcionar para aplicaciones como juegos para comprar elementos que ayuden al juego, o aplicaciones de modelado para comprar elementos especiales, o aplicaciones normales que piden pagos para servicios adicionales.

Según el público objetivo de la aplicación, la aceptación de que sea de pago será muy diferente: hay plataformas donde es probable que el usuario esté dispuesto a pagar y otras en que la publicidad es el mejor medio para conseguir ingresos por la aplicación.

A continuación veremos con más detalle dos de los modelos apuntados en este apartado:

- El modelo de aplicación gratuita
- El modelo de pago directo o indirecto

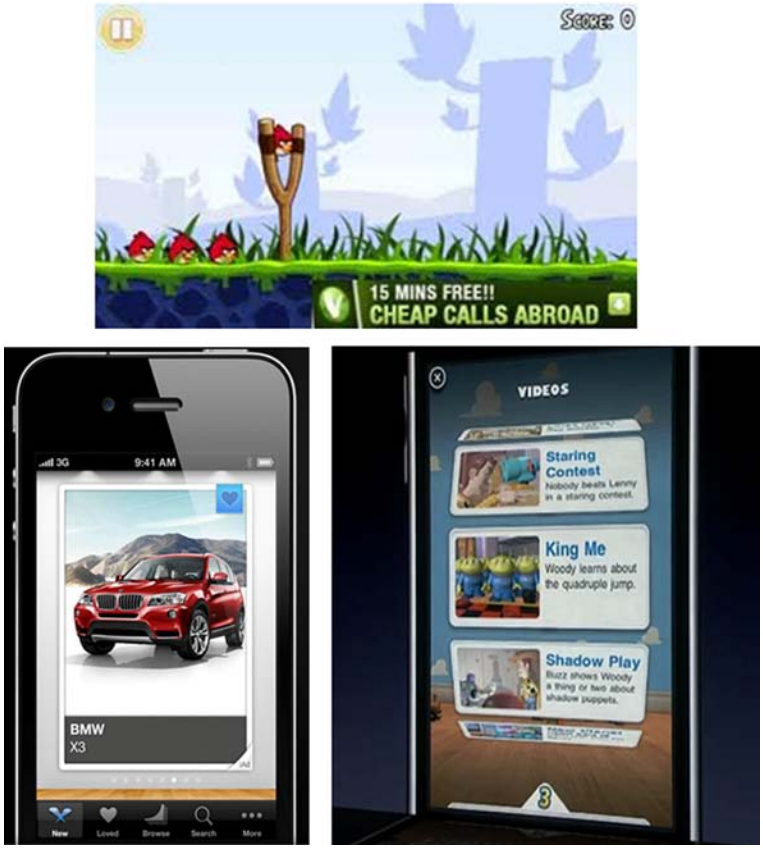
3.1.1. Modelo de aplicación gratuita

Hay muchos tipos de aplicaciones gratuitas, la mayoría sencillamente pretende atraer usuarios para que acaben comprando la versión de pago de la aplicación o productos relacionados con ellas.

Terminología

Este modelo de aplicaciones recibe el nombre de *freemium*.

Otra tipología de aplicación gratuita, a la que también nos hemos referido antes, es la que obtiene sus beneficios a partir de la publicidad, generalmente contextualizada. Este modelo de negocio es muy cercano al de las páginas web que obtienen beneficios por el número de visitas o por el número de clics en los anuncios que incorporan. Hay muchas empresas que ofrecen dicha publicidad y las propias plataformas incorporan soporte directo para ellas, como por ejemplo iAd para iOs, adMob para Android, Microsoft Advertising Exchange para Windows Phone 7 o Blackberry Advertising Service para Blackberry, entre otros. Y muchas de ellas se pueden usar en otras plataformas, lo cual hace algo más fácil portar las aplicaciones. También existen plataformas de publicidad específica para dispositivos móviles que ofrecen la publicidad en el formato correcto y con mayor contextualización.

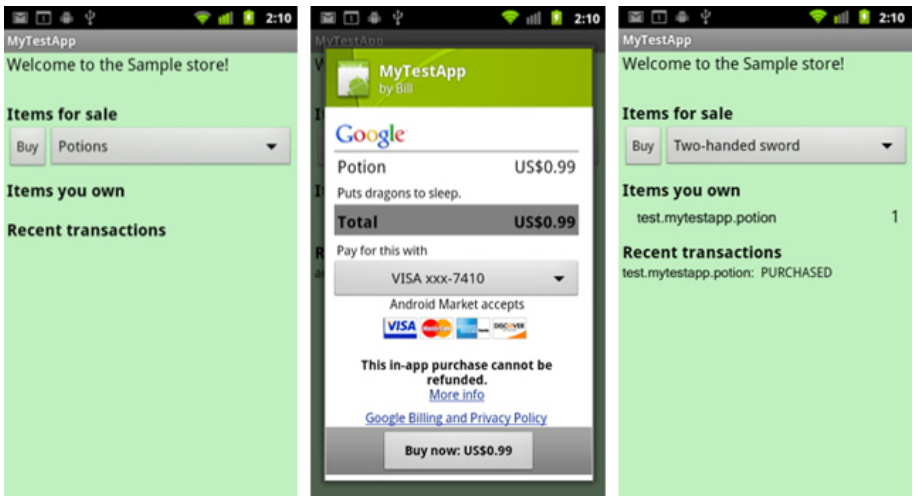


Ejemplo de publicidad en aplicaciones para dispositivos móviles

Este tipo de negocio tiene un beneficio oculto, que es el seguimiento (*tracking*) de los usuarios de la aplicación. Esta información variará según la plataforma de publicidad utilizada, pero siempre ayuda a definir el perfil de los usuarios, lo que redunda inmediatamente en posibilidades de mejora de la aplicación.

3.1.2. Pago directo o indirecto

Las aplicaciones que se deben pagar, ya sea en el momento de la descarga o bien en el momento de utilizar algún servicio concreto, conformarían este grupo, que también engloba las aplicaciones que tienen una versión gratuita y otra de pago.



Ejemplo de aplicación de tipo *pay-in*

El primer paso para conseguir que una aplicación pueda ser vendida una vez desarrollada es incluirla en una tienda de aplicaciones. Actualmente existen muchas tiendas oficiales, tanto de las plataformas como de los fabricantes (si son diferentes) y también tiendas alternativas o extraoficiales.

Las exigencias para poder incluir una aplicación en una tienda pueden variar mucho según la tienda de que se trate, lo que parece indudable es que el número de compradores potenciales suele ser mayor, cuanto mayores son estas exigencias: en una tienda oficial la aplicación deberá pasar unos controles de calidad que, seguramente, no existirán en una tienda alternativa. Evidentemente, el número de compradores potenciales de una tienda oficial es mucho mayor que el de una tienda alternativa.

En resumen, si pretendemos que nuestra aplicación tenga una perspectiva de descargas aceptable, deberemos incluirla en alguna de las tiendas oficiales antes mencionadas, para lo cual habrá que cumplir unos requisitos:

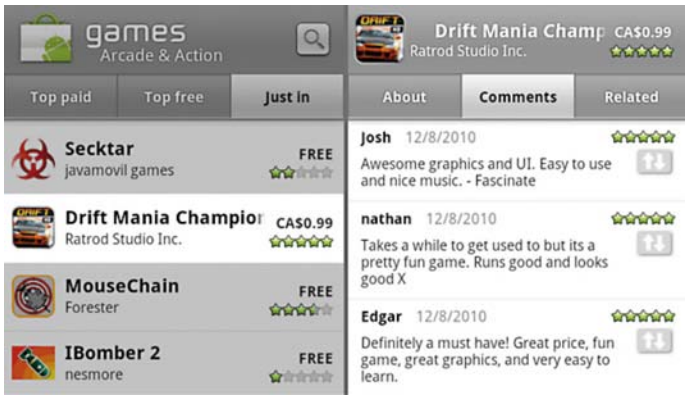
- Registrarnos y, generalmente, pagar para conseguir una cuenta de desarrollador que nos acredite como responsables de las aplicaciones.

- Subir la aplicación a la tienda a través de su zona de desarrollador.

- Superar el proceso de aprobación, lo que puede tardar días o semanas. En todas las tiendas es posible que nuestra aplicación sea eliminada en caso de causar problemas o bien de incumplir alguna de las normas de dicha tienda.

Una vez la aplicación está en la tienda, podremos controlar la información relativa a la misma, como el número de descargas, comentarios, valoraciones, etc.

Debemos tener presente que el éxito de las aplicaciones depende de que los usuarios las puedan encontrar fácilmente, por lo que es primordial situarse en la parte alta de la lista de aplicaciones de la tienda, ya que ocupar esas posiciones suele estar asociado a las descargas y a la valoración de los usuarios, por lo que es muy importante tener en cuenta sus opiniones.



Ejemplo de opiniones sobre las aplicaciones

Resumen

El objetivo principal de este módulo didáctico ha sido ver y aprender sobre las peculiaridades del desarrollo de aplicaciones para dispositivos móviles y, para ello, hemos presentado tanto las principales dificultades como los posibles beneficios derivados del tipo de dispositivos sobre el cual se van a ejecutar dichas aplicaciones y se ha profundizado en las estrategias de desarrollo posibles.

Por lo que respecta a las dificultades hemos destacado especialmente la fragmentación, y se ha estudiado su origen y cómo combatirla. En cuanto a los beneficios, hemos visto que la contextualización de la aplicación y su ubicuidad son los dos grandes puntos a tener en cuenta.

También hemos visto algunas pinceladas sobre los métodos utilizados en proyectos orientados a conseguir este tipo de aplicaciones, con sus puntos fuertes y puntos débiles. Prestando atención a fases que son especialmente delicadas en este tipo de proyectos, como el diseño de la interfaz y la selección del tipo de aplicación (web, nativa, etc.), o incluso únicas para este tipo de aplicaciones como son la distribución basada en *market places*.

Finalmente hemos presentado las posibilidades de negocio para este tipo de aplicaciones.

Actividades

1. Evaluad el estado del arte en las aplicaciones móviles en 3D, investigad las opciones de desarrollo y las API que existen para dos de las siguientes plataformas:

a) Android
b) IPhone
c) Java ME
d) BREW
e) Symbian

2. Instalad y desarrollad una aplicación cualquiera con, al menos, dos herramientas de desarrollo multiplataforma: Phonegap, Rhomobile, Wapps, Appcelerator, etc.

3. Planificad un proyecto de una aplicación sencilla para gestionar tareas compartidas. La planificación debe incluir temporalización, recursos humanos y materiales (dispositivos, por ejemplo).

4. Desarrollad un plan de implantación de dispositivos para los siguientes casos:

a) Una aplicación para leer firmas de documentos oficiales. Las firmas se realizan con el dedo o un bolígrafo sin punta sobre la pantalla del dispositivo.
b) Una aplicación para mostrar la situación de nuestros amigos de una red social y su último comentario.

5. Identificad entre las aplicaciones que usáis diariamente cuáles podrían ser móviles y cuáles no. Explicad los motivos.

6. Averiguad los costes asociados al pago de las aplicaciones.

7. Contabilizad el número total de aplicaciones vendidas y los beneficios generados para la plataforma y para los desarrolladores de tres de las principales plataformas del mercado con tiendas de aplicaciones.

8. Indicad cómo se puede evitar la fragmentación con iOs.

9. Explicad cómo funciona la gestión de la caché del navegador en HTML5.

10. Explicad la diferencia que hay en HTML5 entre base de datos local y gestión fuera de línea de la aplicación.

11. Especificad los componentes que se utilizan para conseguir aplicaciones web móviles nativas. Escribid un ejemplo para cargar la página principal de la UOC.

12. Indicad cuáles son las posibles fuentes de fragmentación actuales.

13. Enumerad las fuentes de fragmentación que se utilizan en una aplicación de visualización de fotografías para dispositivos Android.

14. Poned un ejemplo de aplicación LBS en el contexto de la restauración.

Glosario

accesibilidad *f* Grado en el que las personas pueden utilizar un objeto, visitar un lugar o acceder a un servicio, independientemente de sus capacidades técnicas, cognitivas o físicas.

API *f* Sigla de *application program interface*. Interfaz expuesta para ser utilizada dentro de una aplicación con el objetivo de dar acceso a librerías o funciones externas a la aplicación.

CRM *f* Sigla de *customer relationship management*. Software para gestionar la relación con los clientes.

CSS *m* Sigla de *cascade stylesheet*. Lenguaje usado para definir la presentación de un documento estructurado con XML o HTML.

derivación de software *m* Modificación sobre el software original para poder adaptarse a los cambios de la fragmentación.

device plan *m* Lista ordenada de todos los dispositivos o grupo de dispositivos que se desea soportar.

ecosistema móvil *m* Conjunto de actores necesarios para poder tener los dispositivos móviles y finalmente las aplicaciones para estos dispositivos. En concreto, en este ecosistema se incluyen las operadoras de telecomunicaciones, los fabricantes de hardware y los elementos de software que intervienen en la ejecución de la aplicación

escenario de una aplicación *m* Caso de fragmentación debido a una o varias de las posibles causas de fragmentación.

fragmentación *f* Situación, o condicionantes de la situación, que no permite compartir una misma aplicación entre diferentes ecosistemas sin hacerle las adaptaciones pertinentes.

***framework* de aplicaciones** *m* Marco de desarrollo que permite realizar aplicaciones de manera más sencilla, ordenada y mantenible.

fremium *m* Modelo de negocio gratuito en contraposición al modelo premium.

ide *m* Sigla de *integrated development environment*. Entorno de desarrollo que incorpora todas, o casi todas, las herramientas necesarias (herramientas de modelado o diseño, herramientas de *debugging*, etc.).

inverse of control *m* Patrón de diseño utilizado para definir las dependencias desde un contenedor externo.

iOs *m* Sistema operativo para dispositivos móviles de Iphone.

itinerancia *m* Capacidad de un dispositivo de moverse de una zona de cobertura a otra.

Javascript *m* Dialecto del estándar ECMA Script, utilizado para dar dinamismo a las aplicaciones web.

LBS *m* Sigla de *location based service*. Servicio que intenta ofrecer un valor añadido gracias al conocimiento de la ubicación geográfica del usuario.

lenguaje de marcado o lenguaje de marcas *m* Lenguaje que estructura la información a través de marcas o etiquetas (*tags*).

markup language *m* Lenguaje de marcado o lenguaje de marcas.

mash-up *f* Aplicación que aprovecha las API existentes en la red para interactuar con datos, como puede ser la API de Twitter, o datos públicos del Estado.

método *m* Definición de los pasos a seguir para conseguir un objetivo.

MMS *m* Sigla de *multimedia message system*

MoSoSo *m* Sigla de *mobile social software*. Software con una capa social añadida para aprovechar las conexiones sociales del usuario.

OTA *m* Sigla de *over-the-air*. Método para distribuir actualizaciones u otras funciones accesibles a través de Internet.

plugin *f* Aplicación añadida a una aplicación principal para ampliar y/o cambiar su funcionalidad.

preprocesador *m* Herramienta que permite realizar cambios sobre el código con el objetivo de adaptarlo a unas necesidades específicas.

prueba unitaria *f* Prueba que sirve para comprobar el correcto funcionamiento de una parte del código. Es recomendable que este tipo de pruebas sean automatizables, completas, reutilizables (repetibles a lo largo del tiempo), independientes entre sí y tan profesionales como el propio código.

responsividad *f* Capacidad de respuesta de una aplicación ante las acciones de usuario.

roaming *m* Itinerancia.

SDK *m* Sigla de *software development kit*. Herramientas necesarias para poder realizar el desarrollo de aplicaciones en una plataforma.

smart card *f* Tarjeta inteligente.

tarjeta inteligente *f* Tarjeta con un circuito integrado de tamaño bolsillo que se puede programar con algún tipo de lógica. Un ejemplo de ello son las tarjetas de crédito con microchip.

TDD *m* Sigla de *test driven development*. Desarrollo dirigido por las pruebas: antes de realizar una funcionalidad debe existir una prueba que verifique su funcionamiento.

ubicuidad *f* Capacidad de acceder a toda la información o servicios que necesita el usuario en cualquier momento y circunstancia, a través del dispositivo que tenga actualmente.

usabilidad *f* Facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto.

w3c *m* Sigla de World Wide Web Consorcium. Consorcio internacional que produce recomendaciones, que no estándares, para la World Wide Web (www).

WHATGW *m* Sigla de Web Hypertex Application Techonolgoy Working Group. Grupo formado por las empresas responsables de los principales navegadores web: Apple, Opera, Mozilla, Microsoft y Google.

WML *m* Sigla de *wireless markup languaje*

WURLF *m* Repositorio de información que permite identificar las capacidades y limitaciones de un dispositivo móvil a través de la metainformación de una petición.

Bibliografía

Ahonen, Tomi (2007). *Mobile the 7th Mass Media is to internet like TV is to radio*.

Allen, Sarah; Graupera, Vidal; Lundrigan, Lee (2010). *Pro Smartphone Cross-Platform Development*. Apress.

Blanc, Pablo; Camarero, Julio; Fumero, Antoni; Werterski, Adam; Rodriguez, Pedro (2009). *Metodología de desarrollo ágil para sistemas móviles*. Universidad de Madrid

Fling, Brian *et al.* (2009). *Mobile Design and Development*. O'Reilly.

Lehtimaki, Juhani. "Android UI Design Patterns".

Rajapakse, Damith C. (2008). *Fragmentation of mobile applications*

Spataru, Andrei Cristian (2010). *Agile Development Methods for Mobile Applications*. University of Edinbugh.

Virkus, R.; Gülle, R.; Rouffineau, T, y otros (2011) *Don't panic Mobile Developer's guide to Galaxy*. Enought Software Gmb H + Co. KG.

Wong, Richard (2010). *In Mobile, Fragmentation is Forever. Deal With*.

Woodbridge, Rob (2010). 9 Mobile Business Modesl taht yo ucan use right now to generate revenue.

Enlaces de Internet

http://www.comp.nus.edu.sg/~damithch/df/device-fragmentation.htm

http://techcrunch.com/2010/03/04/mobile-fragmentation-forever/

http://communities-dominate.blogs.com/brands/2007/02/mobile_the_7th_.html

http://www.versionone.com/pdf/mobiledevelopment.pdf

http://www.adamwesterski.com/wp-content/files/docsCursos/Agile_doc_TemasAnv.pdf

http://www.inf.ed.ac.uk/publications/thesis/online/IM100767.pdf

http://developer.smartface.biz/documents/Application_Development_Methodology.pdf

http://untether.tv/ellb/blog/8-mobile-business-models-that-you-can-use-right-now-to-generate-revenue/

http://en.wikipedia.org/

http://www.mit.jyu.fi/opetus/kurssit/jot/2005/kalvot/agile%20sw%20development.pdf

http://developer.android.com

http://developer.apple.com