**Buenas Prácticas para desarrollar una App**

En la vida del desarrollador, puede que comiences un proyecto de cero o te encuentres con uno ya en proceso de **desarrollo**.  
Para el primer caso, eres tu el que comienzas a desarrollarlo, por lo que tienes una libertad en cuanto a la arquitectura y el desarrollo de la misma. La barrera que obstaculiza el entendimiento del proyecto y las horas de depuración para entender el funcionamiento del mismo, desaparecen.

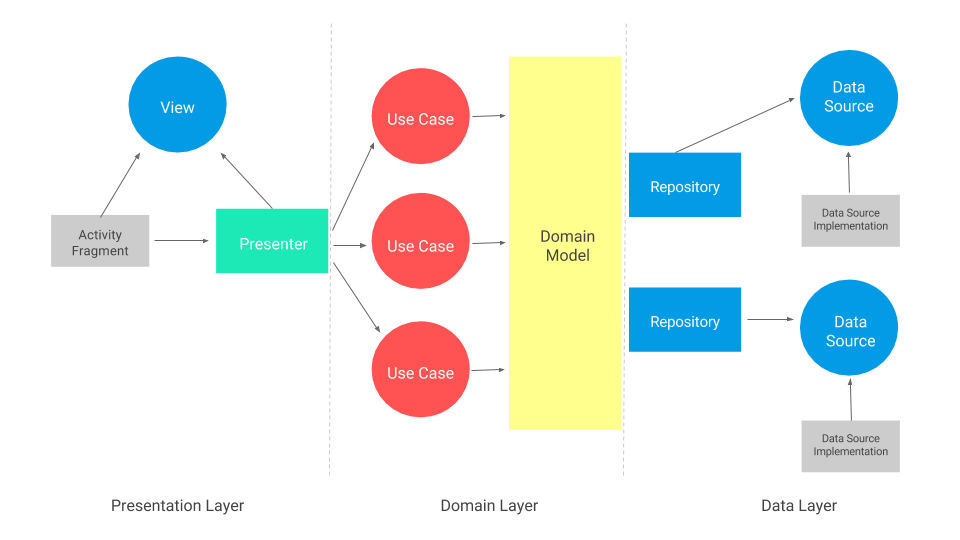
Seguramente te encuentres en el segundo caso, por lo que si el proyecto se realizó con buenas prácticas, no te será muy complicado realizar las modificaciones oportunas para incorporar nuevas funcionalidades, ajustes visuales, optimizaciones…

Lamentablemente, eso no es una realidad para la mayoría de los casos, por lo que se pueden encontrar con proyectos, que si bien, son plenamente funcionales, su escalabilidad es un autentico laberinto, obligando a posteriores desarrolladores a que se adentren dentro de un misterioso y ominoso mundo, algo así como **“Alicia, en el país de la maravillas”** pero con mucho café de por medio.

Se pierde mucho tiempo para adaptar y escalar dichos proyectos. Es por ello, que entran en juego las llamadas **“Buenas practicas para el desarrollo”**. El tiempo tiene un valor incalculable, valor que determina el resultado final del proyecto. Hay unas estimaciones y unos costes. Lo que interesa es reducir el tiempo de entendimiento y de escalabilidad del proyecto, logrando unos resultados.

Como desarrolladores, debemos hacer aplicaciones funcionales, útiles y atractivas, pero también seguras, o al menos hacer todo lo posible para proteger la información privada. Se van a definir una serie de buenas practicas y algunos tips que servirán como primera toma de contacto con el tema:

1.Hay que tener en cuenta la complejidad del proyecto.

Si sabemos de primera mano que va a disponer de una gran cantidad de funcionalidades, y que a posteriori, se van a ir incorporando cada vez más, la arquitectura MVP es una buena solución en cuento a escalabilidad y seguridad.  
Aplicar una arquitectura limpia consumirá más tiempo a la hora de empezar a diseñar la base del proyecto, pero a la larga, reducirá tiempos de cara a posteriores modificaciones.  
Esto es debido al hecho de dividir la parte visual de la lógica, eliminando la dependencia de la base del proyecto con elementos externos y garantizando su funcionamiento integro, acceso de repositorios, reutilización de código, y librerías de terceros para simplificar las arduas tareas [](https://binima.com/wp-content/uploads/2017/12/android_archi.png)(**Butterknife, Dagger. Glide, FlowUp, Calligraphy**…)

2. Documentar el proyecto.

Algo tan cotidiano como ir comentando cada una de las funciones que estamos incorporando al proyecto y que termina en el olvido en muchas ocasiones. El objetivo de esto es tener una hoja de ruta para los actuales y futuros desarrolladores, accediendo a la información desde un documento generado y reducir así el tiempo para entender el funcionamiento del mismo. Piensa en un mapa para turistas en el que consultar cada elemento del lugar… pues la documentación es exactamente igual.

3. Ofuscar el código del proyecto compilado y cifrar los datos comprometedores.

No emplear un mecanismo criptográfico pone en riesgo la integridad de los datos privados del usuario.

Pero utilizar dichos mecanismos de manera inadecuada, garantizando una inexistente seguridad para con el usuario, lo deja en una situación de riesgo incluso peor.  
Prevenir siempre sera la mejor contramedida. Es por ello, que se suelen utilizar soluciones como Proguard, que es gratuita, para ofuscar los métodos del proyecto y realiza optimizaciones del mismo. Como debilidad, no cifra los Strings del proyecto, por lo que si las claves son definidas en dichas variables, pues serán claramente legibles.  
Por ello, hay otras soluciones, algunas de pago como **DexGuard y DexProtector**, que impiden por completo la visualización de la información del mismo mediante algoritmos de cifrado más robustos que los anteriores.  
Es importante mantener las comunicaciones y llamadas a servidores (**Retrofit, Volley**), bases de datos (**Realm**) de manera segura. Esto es fundamental para pasarelas de pago, tarjetas de crédito y transacciones varias (**BrainTree**).  
Existen diversas formas de cifrar la información, recomiendo las soluciones de **SpongyCastle** (Version recortada de Bouncycastle) e **IQrypt**. Crypto ha dado algunos problemas de vulnerabilidad en estos últimos años, y teniendo en cuenta que dispone de unas librerías más básicas que las de SpongyCastle, es un paso atrás.

4. Mantener las librerías en la misma versión.

Cuando añadimos dependencias en gradle, en muchas ocasiones, incluimos referencias a librerías con versiones diferentes. Esto da problemas a la hora de firmar y/o compilar el proyecto en un .apk de producción. Se recomienda introducir variables que gestionen dichas versiones e incluir las referencias necesarias.

5. Pruebas y verificación.

Recomiendo encarecidamente Firebase y su crash-reporting.  
Es fundamental para poder detectar problemas en diferentes terminales y casuísticas. Ademas que es muy fácil de integrar dado que el propio **IDE** viene con esa opción.También se dispone de **Hockeyapp**, que permite la subida de versiones en depuración de una app por cada versión para que podamos probar la **app** en diferentes terminales móviles, pudiendo gestionar y tratar los posibles errores.

6. Tips varios:

**☛ Ralentizaciones con imágenes de alta densidad de pixeles.**  
En muchas ocasiones solo disponemos de un recursos gráfico de alta densidad, y no podemos adaptarlo de diferentes versiones.  
Para evitar re-escalados y ralentizaciones en terminales móviles, se recomienda introducir dichos drawables en un directorio inexistente llamado “**drawable-nodpi**”, para evitar que el dispositivo tenga que re-escalar y calcular los mipmaps de la imagen, generando ralentizaciones.

**☛ Problemas de carga de librerías ya definidas**.  
En el editor Android Studio, con acceder a **File/Invalidate caches/restart** se suele solucionar el problema. Comprobar si la versión del SDK es la adecuada.  
También consultar si en **build.gradle** del proyecto tiene la dependencia.

|  |
| --- |
| maven { url **“https://maven.google.com”**} , |

Todas las librerías vienen definidas en un **.iml** con el nombre del proyecto.

**Mejores prácticas para Desarrollo de Android Apps**

En este artículo se tratan cuestiones relacionadas con el diseño de aplicaciones para tabletas: el diseño de la interfaz de usuario, la duración de las sesiones, las adaptaciones a las diferencias entre usuarios y la localización. Se analizan procedimientos de depuración y optimización. También nos ocupamos del impacto que tienen diferentes métodos de despliegue en las funcionalidades disponibles de las aplicaciones. Con los consejos de este artículo, estará preparado para crear una aplicación que se distinga, dirigida a sus clientes que usen tabletas Intel® Android\*.

Introducción

Compartimos en este artículo un resumen de lo que hemos visto y aprendido, con la esperanza de que ayude también a los lectores a lograr el éxito.

Desde el concepto hasta el diseño y la implementación, los desarrolladores tienen la oportunidad de que sus aplicaciones sobresalgan. La integridad conceptual, la eficiencia de diseño, la elección del entorno y la selección de herramientas son algunos de los muchos aspectos fundamentales con los que los desarrolladores deben hacer malabares. En este artículo se ofrecen sugerencias que conviene tener presentes para aprovechar el potencial de los dispositivos.

Mantenerse enfocados

Los mejores juegos y aplicaciones tienen una idea básica que respetan a la perfección. Si ya tienes la idea central para tu juego o app reducida a su esencia, esto te facilitará tomar la mayoría de las demás decisiones relacionadas con el desarrollo. Si la finalidad no es clara, tendrás un producto con características que no vienen al caso, y los usuarios no la utilizarán porque prefieren inclinarse por aplicaciones que hagan una sola cosa realmente bien. Con la integridad conceptual apropiada, detectarás oportunidades de mejorar el diseño y evitar perder tiempo en desarrollar funcionalidades que los usuarios ya reciben de otras aplicaciones.

Diseñar para la mayor cantidad posible de dispositivos

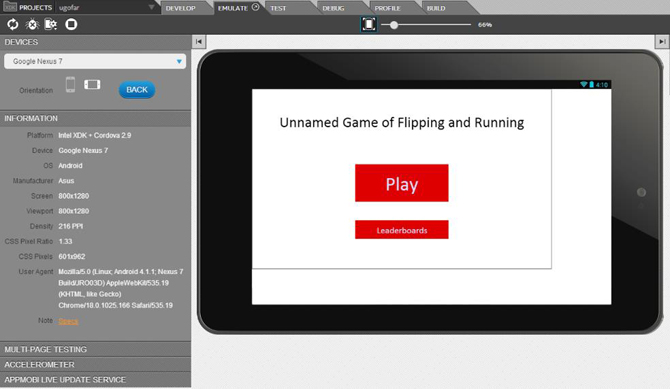
[](http://phoboslab.org/log/2012/09/drawing-pixels-is-hard)Si bien es conviene tener en mente un dispositivo objetivo, es importante que tu aplicación se adapte bien a una amplia variedad de plataformas. Para que las pantallas pequeñas muestren fielmente los aspectos visuales, diseña para la pantalla más grande en la que se vaya a ejecutar tu aplicación y luego reduce el tamaño para que se ajuste al área visible. Hay varios métodos para lograr[compatibilidad con diferentes tamaños de pantallas](http://developer.android.com/guide/practices/screens_support.html).

Figura 1: Los elementos de tamaño fijo de la interfaz de usuario pueden verse demasiado pequeños en pantallas de tabletas grandes.

Otra cuestión que se debe considerar es que todos los controles y el texto deben ser útiles en las pantallas más pequeñas. <https://developer.android.com/design/index.html>

Diseñar para movilidad

El usuario promedio consulta el teléfono entre 100 y 900 veces al día, las sesiones de juego informales duran unos 3 minutos y los lugares de uso son muy distintos (de la propia casa al trabajo, y todo lo que haya en el medio, como transporte público, restaurantes, etc.); por ello, la variación de entornos de las aplicaciones y los juegos móviles repercuten de manera considerable en su facilidad de uso.

Si estás creando un juego, la duración general de las sesiones se reduce a pocos minutos. Esto no quiere decir que estén prohibidas las aventuras épicas inacabables, pero sí significa que debe haber un mecanismo para que los usuarios hagan una pausa casi en cualquier momento y no tengan problemas para regresar al juego. Más allá de si el mecanismo es incorporar batallas cortas o separar rompecabezas con pantallas de selección de nivel, o un sencillo menú permanente para hacer una pausa o continuar jugando que incluya en resumen de lo que está ocurriendo en el juego, la mayoría de los usuarios necesitan tener la posibilidad de abandonar sus dispositivos.

De manera similar, si la aplicación exige silencio absoluto y una iluminación perfecta, muy pocos podrán usarla fuera de casa, y quizá ni siquiera allí. Todo, desde el[color](http://www.uxmatters.com/mt/archives/2007/01/applying-color-theory-to-digital-displays.php) que se elija hasta la[modulación del sonido](http://www.gamasutra.com/view/feature/134761/aaalite_audio_design_challenges_.php?print=1), es importante para asegurarse de que la experiencia del usuario sea buena.

Diseña para personas

Si está creando aplicaciones para un conjunto muy reducido de la población, puede ajustar el diseño a sus necesidades, pero esta es una situación poco común. La mayoría de las aplicaciones no puede darse ese lujo. Deben poder usarlas la mayor cantidad de gente posible.

Además de lo que ya mencionamos sobre los entornos de uso, como el transporte público, hay más de[360 millones de personas con problemas de audición](http://www.who.int/mediacentre/factsheets/fs300/en/),[285 millones con discapacidad visual](http://www.who.int/mediacentre/factsheets/fs282/en/) y[280 millones con daltonismo](http://www.vischeck.com/faq/) en todo el mundo. La mayoría de los usuarios que necesitan algún tipo de asistencia ya tendrán alguna solución preparada, pero de todos modos este es un aspecto del diseño y el desarrollo que requiere de atención. Si existe alguna manera de agregar indicaciones sonoras a las aplicaciones con muchas imágenes, subtítulos a las partes habladas de los juegos y símbolos identificables para diferenciar entre objetos de colores, o una paleta distinguible para daltónicos en la cual todos los colores importantes tengan distinta luminosidad, estará mucho más cerca de lograr que todo aquel que descargue su producto pueda usarlo.

La localización es otro terreno al que muchos juegos y aplicaciones terminan por dedicar tiempo de desarrollo y dinero con el único fin de llegar a otros países. Como el costo puede llegar a ser de[10.000 a 15.000 dólares estadounidenses por idioma](http://books.google.com/books?id=sN5Glp9sLj0C&amp;amp;printsec=frontcover&amp;amp;quot;%20%5Cl%20&amp;amp;quot;v=onepage&amp;amp;q&amp;amp;f=false), conviene reducir al mínimo el texto que se necesite localizar. ¿Puede su aplicación hacerse entender por el usuario mediante símbolos comprensibles en todo el mundo? ¿Pueden los personajes del juego usar emojis universales para comunicarse, en lugar de palabras? Cada paso que se acerque a interactuar directamente con la mente del usuario en lugar de con su vocabulario aumentará el público potencial, sin costo adicional.

Elegir el tipo de ejecutable adecuado

Si tu aplicación no va a utilizar un[motor de juego ya existente](https://software.intel.com/en-us/android/articles/choosing-the-right-engine-for-your-x86-based-android-game), una de las primeras decisiones que se deben tomar en cuanto a la implementación es qué tipo de ejecutable tendrá: nativo, web o híbrido.

Nativo. Las aplicaciones nativas son de bajo nivel, lo cual permite acceder a recursos de hardware más potentes, pero impone más restricciones al desarrollo y la portación a otras plataformas. El lenguaje más común para el desarrollo Android nativo es Java, generalmente por medio del[SDK (Kit de Desarrollo de Software) para Android\*](http://developer.android.com/sdk/index.html) básico o el[NDK (Kit de Desarrollo Nativo)](http://developer.android.com/tools/sdk/ndk/index.html). Herramientas tales como[Intel® INDE (Experiencia Integrada de Desarrollo Nativo de Intel®)](https://software.intel.com/es-es/intel-inde) se centran tanto en Java como en C++. Intel INDE no solo contiene compiladores, sino también herramientas de análisis que ayudan a identificar y corregir problemas de rendimiento de las aplicaciones.

Web. Las aplicaciones web, desde los juegos para navegadores hasta las herramientas online, son versátiles y de fácil distribución, pero a menudo les falta potencia y, como lo implica su nombre, por lo general necesitan conectividad a internet para funcionar. HTML5 y JavaScript\* son la norma para estas aplicaciones. Si la aplicación funciona simplemente como una interfaz a alguna herramienta del lado del servidor, la opción web es excelente; la parte negativa es que cada uno de los usuarios accede a los servidores, lo que eleva los costos de alojamiento para satisfacer la demanda.

Híbrido. Para aprovechar la flexibilidad natural de las aplicaciones web y la potencia del hardware de los dispositivos móviles, se puede optar por aplicaciones híbridas: tecnología web integrada en una envoltura nativa.[Intel XDK (Cross-platform Development Kit)](http://xdk.intel.com/) y otras herramientas de este tipo son ideales para elaborar aplicaciones híbridas. Las mismas aplicaciones web en JavaScript que mencionamos antes se pueden importar, potenciadas con llamadas a API que saquen provecho del hardware, y darles un toque final con código “listo para el dispositivo”, para crear paquetes nativos en plataformas móviles.

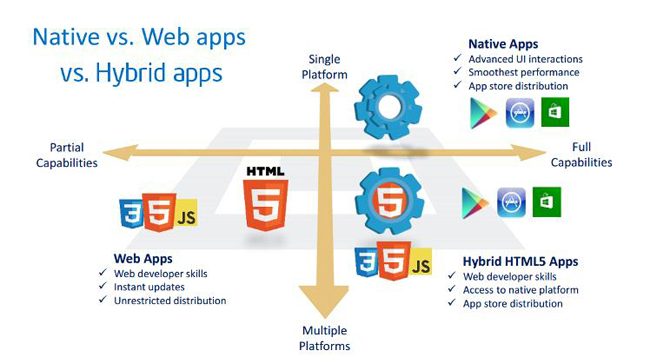


Figura 2: La elección de un ejecutable es algo que lo afectará a usted y a sus clientes.

La decisión de optar por una aplicación nativa, web o híbrida depende en gran medida del uso que se le pretenda dar, los recursos y la funcionalidad requeridos, y la familiaridad del desarrollador con los distintos lenguajes. A la vez, esta decisión determina la manera en que se puede distribuir, usar y monetizar la aplicación.

Hacer lo más ligera posible

Una interfaz fácil de usar brinda muchos beneficios, pero para aprovechar al máximo la aplicación desarrollada, es necesario que no exija demasiado al dispositivo. Las descargas de tamaños inmanejables, el uso exagerado de recursos y el pedido de permisos innecesarios son barreras que dificultan la adopción o la continuidad de uso (y obstaculizan las ventas por el boca en boca).

Una manera de reducir el tamaño de la descarga inicial es con descargas posteriores de “archivos adicionales”. Lamentablemente, esto quizá moleste a los usuarios y trae consigo un problema más importante que también puede afectar el rendimiento: el manejo subóptimo de recursos redundantes. Las bibliotecas de texturas, los modelos y los sonidos que constituyen la mayor parte del tamaño del archivo tendrán que cargarse en la memoria (y si no es así, hay otros problemas), lo que hará más lenta la aplicación. Si se resuelve la causa de origen, no solo la aplicación funcionará mejor, sino que la descarga inicial y las siguientes se reducirán sustancialmente.

La solución para muchas aplicaciones es un procedimiento que apunta a la reutilización y que se ideó en las viejas épocas del desarrollo con limitaciones de memoria. Usar las mismas imágenes y sonidos con diferentes paletas y máscaras puede proporcionar la variedad necesaria y a la vez reducir al mínimo el espacio y los recursos requeridos. Incluso si no hay posibilidad de recurrir a la reutilización en el sentido estricto, reducir al mínimo los archivos a los que sea necesario acceder aumentará el rendimiento de manera demostrable. Por ejemplo,[World of Goo crea atlas de texturas para fines específicos](https://software.intel.com/en-us/android/articles/world-of-goo-stretches-to-android-on-x86) durante la carga de niveles, de esta manera incluso los recursos de imagen que se usan en otros lugares se combinan en un solo recurso. Este incremento del rendimiento dará réditos en el paso siguiente: análisis y optimización.

Eliminar errores y simplificar la ejecución

Lo ideal sería que la aplicación se ejecutara sin ningún inconveniente la primera vez y que no fuera necesario hacer mejoras. Pero el mundo real no es así. En toda aplicación más compleja que un ejemplo para recién iniciados, invariablemente habrá errores. Los peores errores no impiden que funcione la aplicación, sino que la hacen más lenta, en cuanto al rendimiento o la latencia, y la experiencia del usuario se ve perjudicada. Asimismo, el[sobredibujo](about:blank) y los[búferes dobles sin utilizar](https://udn.unrealengine.com/questions/171391/turbocharging-xcom-an-unreal-optimization.html) son dos ejemplos de problemas que, aunque no son errores, reducen el rendimiento y la duración de la batería. Todas estas son oportunidades de mejorar aún más su aplicación. La mejor manera de encontrar oportunidades como estas es con herramientas de análisis de perfiles y analizadores de rendimiento.

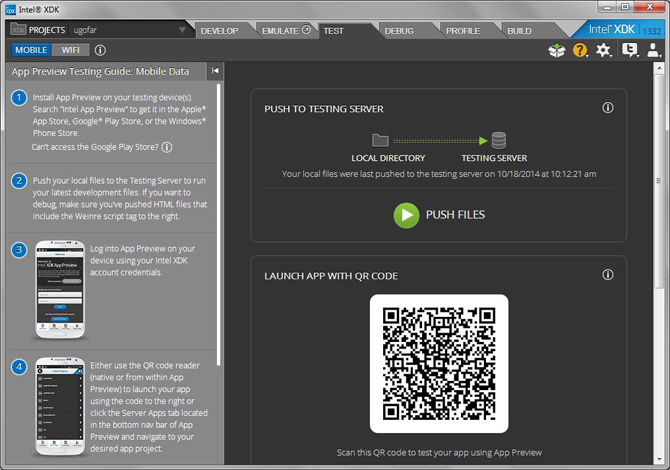
Intel® XDK ofrece depuración en el[emulador](https://software.intel.com/en-us/xdkbook/devemulator), depuración remota en dispositivos Android conectados en la[pestaña Debug](https://software.intel.com/en-us/html5/articles/using-the-debug-tab) e información de análisis de rendimiento en la[pestaña Profile](https://software.intel.com/en-us/html5/articles/using-the-profile-tab).

Figura 3: Intel® XDK facilita el desarrollo de aplicaciones híbridas

¿No usa Intel XDK? Los[Analizadores de Rendimiento de Gráficos Intel® (Intel® GPA)](about:blank) son un conjunto de herramientas de análisis de rendimiento ideadas con el fin de ayudar a identificar áreas subóptimas para mejorarlas. Aunque estas herramientas son perfectamente aptas para aplicaciones que usan todo el poder ofrecido por las PC, también[funcionan con aplicaciones Android](https://software.intel.com/en-us/android/articles/using-the-intel-gpa-system-analyzer-to-improve-android-apps), y ganan su lugar por derecho propio entre las otras herramientas nativas de Intel® INDE.

Resumen

Hay muchísimos factores que se deben tener en cuenta cuando se desarrollan aplicaciones o juegos móviles. La integridad conceptual es clave para decidir entre la enormidad de opciones y acotar el infinito espacio creativo. Cada aspecto del desarrollo puede repercutir en los anteriores y los que le siguen, por ello son fundamentales la planificación y la reiteración. Las convenciones sobre interfaces táctiles, globalización y diseño de imagen y sonido para dispositivos móviles constituyen puntos de partida valiosos y materia para reflexionar en este ejercicio.

Intel® INDE ofrece un enorme conjunto de herramientas para desarrolladores de aplicaciones nativas. Intel® XDK ayuda a los desarrolladores web en todo lo que puedan necesitar: desarrollo, emulación, pruebas, depuración y empaquetado. Ambos tienen abundante documentación y asistencia en foros que continuarán creciendo.

Los métodos y herramientas que se incluyen en este artículo se tomaron de lecciones que los ingenieros de Intel han aprendido de ayudar a muchos desarrolladores de aplicaciones. Son de suma utilidad para maximizar la probabilidad de que su aplicación logre popularidad y pase a formar parte de la vida de sus clientes. Buscar un concepto, un diseño y una implementación óptima es una empresa exigente en extremo, pero que termina siendo provechosa.

*Referencias de la información obtenida:*

<https://sg.com.mx/buzz/mejores-practicas-para-desarrollo-android-apps>

<https://binima.com/buenas-practicas-para-desarrollar-app/>

<https://material.io/develop/android/>

<https://developer.android.com/guide>