EGC

# Gestión de la Configuración de AOSP

# Gestión de la Configuración de AOSP

Sergio García Alonso
Sergio Rodríguez Calvo
Juan Manuel López Pazos
David González Belda
Rafael Vázquez Sánchez

Fecha publicación: 24/12/2013

## Licencia

# 1. Índice

# 2. Índice de Figuras

# 3. Índice de Tablas

# 4. Resumen

Este proyecto tiene como objetivo comprender los distintos aspectos de la gestión de la configuración de AOSP (*Android Open Source Project*), cuyo fin es el de controlar los cambios acaecidos en el proyecto, así como definir el método de aprobación de dichos cambios y asegurar que los integrantes/participantes del proyecto disponen de las versiones adecuadas de los productos generados durante el proyecto.

Para ello vamos a estudiar cómo se realizan y gestionan las siguientes fases de la gestión de la configuración en el proyecto AOSP:

- **Gestión del código fuente**: Analizaremos la estructura del código fuente del proyecto, los roles, la plataforma donde es alojado el código, etc. Se realiza mayormente mediante una serie de repositorios Git, gestionados gracias a Gerrit, en el que compañías y desarrolladores hacen sus aportaciones al proyecto de forma pública.

- **Gestión de la construcción**: Se indicarán las herramientas utilizadas para la construcción del proyecto, cómo se utilizan, etc. En el caso de AOSP la construcción se realiza con un dispositivo concreto como objetivo, e incluye componentes privativos externos al proyecto, pero fácilmente integrables con las herramientas proporcionadas.

- **Gestión de los entregables**: Qué entregables se contemplan en AOSP y cómo se realiza su entrega. Estos son el propio código fuente, liberado como entregable con cada nueva versión, y la documentación que no sigue una política establecida más allá de su dependencia con el código fuente.

- **Gestión del despliegue**: Estudiaremos cómo realiza AOSP la instalación de las construcciones en los dispositivos.

- **Gestión de incidencias y depuración**: Se explica el proceso de reporte de incidencias y cómo se realiza la depuración. Para AOSP se utiliza Google Code como *Issue Tracker* oficial, tanto para sugerencias de mejora como para reportes de errores de la plataforma.

- **Gestión de la variabilidad**: Se realizará una aproximación a lo que sería la línea de producto software en AOSP para, posteriormente, analizar las características del producto resultante mediante un modelo de características. Es un apartado importante de AOSP por la naturaleza del proyecto, ya que éste forma una base de software sobre la que los fabricantes puedan construir su producto a medida.

- **Integración y despliegue continuos**: Como AOSP no contempla la integración ni despliegue continuos, explicaremos una forma de agregarlo por nuestra cuenta mediante Jenkins.

- **Las pruebas**: Se describen los tipos de pruebas que se realizan en AOSP y el procedimiento para ejecutarlas, haciendo hincapié en el CTS o *Compatibility Test Suite* que deben pasar los dispositivos para demostrar su correcto funcionamiento y compatibilidad con la plataforma,

- **Mapa de herramientas**: Se describen las herramientas necesarias para trabajar con AOSP en las diferentes actividades de la Gestión de la Configuración, y la relación que tienen entre ellas. Las principales con las que el desarrollador tendrá que estar familiarizado son Git, Gerrit, Repo, Java, Make, Linux/OSX.

# 5. Introducción

Este proyecto tiene como objetivo comprender los distintos aspectos de la gestión de la configuración de AOSP, pero primero necesitamos comprender qué es la gestión de la configuración. Para ello vamos a ver su definición según varias referencias importantes, y luego intentaremos dar una propia lo más correcta posible.

Según Wikipedia[1]:

> *"se denomina Gestión de la Configuración al conjunto de procesos destinados a asegurar la validez de todo producto obtenido durante cualquiera de las etapas del desarrollo de un Sistema de Información, a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo"*

Sin embargo, si consultamos dicha definición en ITIL[2] vemos que:

> *"es un proceso organizacional que trata de proporcionar un modelo lógico de la Infraestructura IT Gestión de la configuración del software Introducción Conceptos por la identificación, mantenimiento y verificación de TODOS los elementos de la configuración existentes"*

También disponemos de la definición aportada por Métrica v3.0, la cual nos dice que[3]:

> *"es una interfaz cuyo objetivo es mantener la integridad de los productos que se obtienen a lo largo del desarrollo de los sistemas de información, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema disponen de la versión adecuada de los productos que manejan"*

Recientemente hemos aprendido otra aportada por PMBOK, el cual define la GC como[4]:

> *"un subsistema del sistema de información de la gestión de proyectos en general. El sistema incluye el proceso para presentar los cambios propuestos, realizar el seguimiento de sistemas para la revisión y aprobación de los cambios propuestos, definir los niveles de aprobación para autorizar los cambios y proporcionar un método para validar los cambios aprobados"*

[1] http://es.wikipedia.org/wiki/Gesti%C3%B3n_de_la_configuraci%C3%B3n
[2] Fundamentos de la gestión de servicios de TI basada en ITIL V3. ITSMF-International, 2008.
[3] Métrica v3
[4] PMBOK

Nuestra propuesta de definición de Gestión de la Configuración es:

> *proceso que tendrá lugar a lo largo del proyecto a desarrollar - esto es desde que se inicia el proyecto hasta que el software se deja de comercializar -, en el que se realizará un estudio con el fin de controlar los cambios acaecidos, así como aprobar dichos cambios y asegurar que los integrantes/participantes del proyecto disponen de las versiones adecuadas de los productos generados durante el proyecto.*

Para comprender cada apartado de la gestión de la configuración vamos a realizar esta memoria, donde en base al proyecto de software libre *Android Open Source Project* ( AOSP en adelante) estudiaremos y analizaremos cada una de la fase de la gestión de la configuración.

Las fases que se van a estudiar son:

- **Gestión del código fuente**: Analizaremos la estructura del código fuente del proyecto, los roles, la plataforma donde es alojado el código, etc.
- **Gestión de la construcción**: Se indicarán las herramientas utilizadas para la construcción del proyecto, cómo se utilizan, etc.
- **Gestión de los entregables**: Qué entregables se contemplan en AOSP y cómo se realiza su entrega.
- **Gestión del despliegue**: Estudiaremos cómo realiza AOSP la instalación de las construcciones en los dispositivos.
- **Gestión de incidencias y depuración**: Se explica el proceso de reporte de incidencias y cómo se realiza la depuración.
- **Gestión de la variabilidad**: Se realizará una aproximación a lo que sería la línea de producto software en AOSP para, posteriormente, analizar las características del producto resultante mediante un modelo de características.
- **Integración y despliegue continuos**: Como AOSP no contempla la integración ni despliegue continuos, explicaremos una forma de agregarlo por nuestra cuenta.
- **Las pruebas**: Se describen los tipos de pruebas que se realizan en AOSP y el procedimiento para ejecutarlas.
- **Mapa de herramientas**: Se describen las herramientas necesarias para trabajar con AOSP en las diferentes actividades de la Gestión de la Configuración, y la relación que tienen entre ellas.

Además se aportarán unas conclusiones y se realizarán sugerencias de mejora sobre la Gestión de la Configuración en AOSP para cada apartado, según el criterio de los miembros del equipo. Se plantearán unos ejercicios prácticos a realizar sobre AOSP, como por ejemplo la construcción de una parte del mismo o la configuración del entorno de desarrollo. Tanto los ejercicios como el diario de grupo serán entregables adjuntos a esta memoria ya que se pretende complementarla.

Muy estrechamente relacionado con AOSP está Android, que describimos como una plataforma de código abierto para una gran variedad de dispositivos, particularmente

Smartphones. Sus propósitos principales son la creación de una plataforma de software abierta disponible para todos (fabricantes, desarrolladores, etc.) sobre la que poder realizar aplicaciones y mejorar la experiencia Smartphone para los usuarios, que evite que alguien pueda restringir o controlar las innovaciones de cualquier otro, además de reducir fallos trabajando en conjunto. El resultado es un producto que, al ser abierto, permite personalización, portabilidad y estabilidad.

El control del proyecto lo lleva a cabo Google, existiendo colaboración por parte de otras empresas, organizaciones, comunidades, etc. Todas intervienen y colaboran en el proyecto, bien de forma directa o en forma de contribución a subproyectos.

AOSP usa varias licencias de código abierto para licenciar su contenido. Principalmente usa la licencia Apache Software License, Version 2.0 (Apache 2.0)[5]. Dicha licencia permite la modificación y distribución del producto sin necesidad de distribuir el código fuente, pero sí obliga a mantener el copyright.

Por sus características AOSP es un proyecto que se adapta muy bien a los objetivos de esta memoria y a su justificación, ya que es un proyecto grande, complejo, documentado, abierto que cuenta con una amplia comunidad de desarrollo por detrás, lo cual nos permite estudiar los aspectos citados anteriormente mayor con facilidad.

Debido a la naturaleza de AOSP, que se centra especialmente en la gestión del código y de incidencias con la ayuda de la comunidad, no contempla por sí mismo algunos aspectos como por ejemplo la integración continua. Esto se debe a que AOSP es un proyecto que, como se ha descrito antes, está orientado a muchos dispositivos con diferentes características de hardware, y a ninguno en concreto. AOSP constituye una base para que fabricantes y desarrolladores tomen y modifiquen a su antojo.

La gestión del código fuente es la parte mejor documentada del proyecto. Para esta tarea se pone a disposición pública un repositorio Git donde se controla el versionado del código de todos los proyectos que componen. Para gestionar las contribuciones se utiliza Gerrit, el sistema web de revisión de código creado por Google específicamente para el proyecto AOSP, mediante el cual se envían parches al código para ser revisados y aceptados si procede. Pero de cara al público sólo se gestiona la contribución con mejoras y arreglos de fallos. Cambios de mayor envergadura que afecten a funciones o a las APIs de la plataforma se rechazan. Este tipo de cambios son añadidos por Google durante el desarrollo de nuevas versiones de forma privada, y son publicados en conjunto cuando se presenta al público dicha versión. Este enfoque de evolución cerrada, a pesar de ser AOSP un proyecto abierto, se hace por motivos de estrategia de mercado.

Para la construcción se facilitan todas las herramientas y se describen los procesos necesarios. La construcción debe hacerse siempre para un dispositivo como objetivo, pues hay que integrar distintas partes de AOSP además de otros componentes, normalmente privativos y distribuidos como binarios, que no forman parte del proyecto pero son necesarios para su

[5] http://www.apache.org/licenses/LICENSE-2.0

funcionamiento como producto final. Por suerte, el procedimiento es muy similar para cualquier dispositivo, y sólo es necesario variar la configuración básica de las herramientas según el dispositivo elegido.

Por otro lado se gestionan las incidencias del proyecto. Existe un *Issue Tracker* oficial en la plataforma de Google Code dedicado de forma exclusiva al reporte de errores por parte de usuarios y desarrolladores. Además del canal oficial, los ingenieros de Google acostumbran a ser activos en redes sociales (especialmente Twitter, Google+ y StackOverflow), por lo que no es extraño verlos discutiendo y comentando problemas y sugerencias con los usuarios.

El caso de la integración y despliegue continuos es más grave. Por los motivos comentados, AOSP de forma genérica no puede ser construido, integrado ni desplegado de manera continua y automática. Por ello, la integración y despliegue continuos es responsabilidad del fabricante o grupo encargado de realizar el producto final, y no se especifican indicaciones oficiales al respecto. Suponemos que la mayoría de fabricantes, incluído Google, tendrán su sistema de integración continua propio, pero estos no publican información sobre cómo lo llevan a cabo. Nosotros hemos propuesto un método de integración continua con Jenkins basándonos en la información proporcionada por la comunidad de desarrollo.

El despliegue sufre una limitación similar. AOSP no contempla un sistema de despliegue continuo, corre a cuenta del fabricante utilizar el método que más se adapte a su modelo de negocio. Algunos implementan actualizaciones automáticas a través del propio dispositivo, otros lo hacen a través de software especializado, y otros ni siquiera lo contemplan. Lo que sí provee AOSP es el protocolo Fastboot, un protocolo que permite modificar la memoria flash de los terminales a través de USB, y con ello desplegar nuevas versiones del producto, por lo que explicaremos en detalle su funcionamiento. Aun así no todos los fabricantes proveen este método, lo cual no ayuda a poder establecer una política de despliegue clara.

Las pruebas de AOSP sí están bien definidas, tanto a nivel individual de cada subproyecto como a nivel global de la plataforma. Cada proyecto que compone AOSP define sus propias pruebas dependiendo de la naturaleza de dicho proyecto, ya que muchos están hechos en diferentes lenguajes de programación. Por otra parte, AOSP define una suite de pruebas sobre el producto final, el *Compatibility Test Suite*, que consiste en pruebas de diverso tipo para comprobar que el resultado se comporta como se espera de un producto basado en AOSP. Mediante estas pruebas se concede el certificado de compatibilidad, que certifica que el producto es compatible con el resto de la plataforma, según los requisitos establecidos en el *Compatibility Definition Document*.

La gestión de la variabilidad es especialmente interesante en AOSP, por ser un proyecto tan abierto y genérico. Como hemos comentado, no está enfocado a ningún dispositivo concreto, aunque se centra principalmente en Smartphones y Tablets. AOSP está organizado por subproyectos en forma de módulos, de manera que estos pueden ser integrados o no según las características deseadas en el producto final. Son el caso de características dependientes del hardware como la conexión a redes móviles, determinados sensores de movimiento y posición, etc. Por otro lado, existe una variabilidad más allá de AOSP, que consiste en la personalización que cada fabricante quiera introducir en el producto. Es

decir, aplicaciones propias como tiendas de aplicaciones o escritorios diferentes, ajustes en el diseño de las interfaces del sistema, características especiales de hardware, etc. Además, pueden darse casos de productos basados en AOSP muy diferentes al propósito principal de éste, como relojes inteligentes, gafas de realidad aumentada, centros multimedia, sistemas empotrados, e incluso electrodomésticos. La variabilidad es el punto más fuerte de AOSP y su razón de ser.

# 6. Objetivos y Justificación

Los objetivos del proyecto serán:

- Estudiar y comprender la gestión de la configuración de AOSP, es decir, conceptos de la evolución del proyecto (software).
- Reunir información sobre las técnicas, procesos y herramientas para el manejo eficaz y sistemático de la evolución y exponerlas junto a una conclusión.
- Analizar y documentar la gestión de la configuración en AOSP y proponer mejoras, por cada apartado de la gestión de la configuración: gestión del código fuente, gestión de la construcción, gestión del cambio, gestión de las entregas y gestión del despliegue.
- Desarrollar una máquina virtual con las herramientas necesarias para trabajar en AOSP.
- Sacar conclusiones sobre la gestión de la configuración en AOSP y sobre la importancia de la gestión de la configuración en general para los proyectos software.
- Para cada apartado del proyecto desarrollar una actividad, así como indicar su solución.

Como justificación este proyecto surge a elección por parte de los miembros del grupo como respuesta a la propuesta realizada por los profesores de la asignatura de elaborar una memoria para el estudio de los contenidos estudiados en la asignatura, es decir sobre evolución y gestión de la configuración, sobre un proyecto de software libre.

En concreto, hemos elegido Android Open Source Project (AOSP) por cumplir con los requisitos necesarios que son: a) ser un proyecto de código abierto, b) contar con la documentación necesaria para estudiar cómo se lleva a cabo la evaluación y gestión de la configuración, c) estar actualmente activo y d) contar con un desarrollo profesional del mismo. Adicionalmente, es un proyecto de interés para los integrantes del grupo.

# 7. Glosario de Términos

| Término | Descripción |
|---|---|
| AOSP | AOSP (*Android Open Source Project*) es un proyecto de código abierto creado con la intención de orientar el desarrollo de la plataforma Android y que todo aquel que lo desee pueda instalar dicha plataforma en su dispositivo. |
| Android (plataforma) | Android, entendido como plataforma, es el conjunto formado por el sistema operativo Android, middleware y aplicaciones móviles que funcionan sobre ese sistema operativo. |
| Android (producto) | Android también se conoce como el producto final que incluyen los fabricantes en sus dispositivos, el propio Sistema Operativo. Cuando el producto no incluye personalización de fabricantes se suele conocer como "Android Puro" o "*Stock Android*". |
| API | Las API (*Application Programming Interface*) son conjuntos de funciones y/o objetos que definen la forma de interactuar con un artefacto software (clase, objeto, programa, sistema, etc) |
| Bootloader (software) | Un bootloader (gestor de arranque) es un programa software encargado de preparar todo aquello que el sistema operativo necesita para cargar y empezar a funcionar. Cada fabricante incluye el suyo propio, pues depende del hardware del dispositivo. |
| Bootloader (modo) | El modo bootloader se conoce al modo especial de arranque en el que el dispositivo carga los componentes básicos de hardware sin llegar a arrancar el Sistema Operativo, para realizar tareas de mantenimiento. |
| Bug | Un bug o fallo es la causa que provoca un conflicto en el sistema. |
| Código fuente | El código fuente son las instrucciones que componen un programa informático o software. |
| Construcción | La construcción de software se refiere a la creación de software, a través de una combinación de codificación, prueba, y depuración.[6] |
| CTS | El CTS (*Compatibility Test Suite*) de AOSP es un conjunto de pruebas automatizadas que se realizan a un dispositivo para comprobar que funcione correctamente según los requerimientos de la plataforma Android. |

[6] http://www.sites.upiicsa.ipn.mx/polilibros/portal/Polilibros/Complemento%20Material%20Didactico/Maest-Ing-Soft-Sergio/Cuerpoconocimiento/Construcci%C3%B3n%20del%20software.htm

| | |
|---|---|
| CDD | El CDD (*Compatibility Definition Document*) es un extenso y detallado documento donde se especifican los requisitos a cumplir para pasar el CTS. |
| CyanogenMod | Es una versión modificada y personalizada de Android, basada en AOSP, desarrollada de forma comunitaria. Es la ROM cocinada con mayor popularidad del ecosistema Android. |
| Dalvik | Dalvik es la máquina virtual que AOSP utiliza para ejecutar el código java. La máquina virtual Dalvik (DVM) sacrifica parte de la portabilidad característica de Java para proporcionar un mayor rendimiento y un menor consumo de batería. |
| Depuración | La depuración consiste en localizar y corregir los errores que surgen en la programación. |
| Despliegue | El despliegue consiste en instalar el software en el entorno conocido como producción, es decir, entorno real donde ha sido contemplado durante su desarrollo. Se realiza tras la construcción y las pruebas. |
| Entregable | Un entregable es cada uno de los productos (código fuente, scripts, documentación, etc.) que surgen en un proceso de desarrollo software. Tiene implícito una fecha de entrega. |
| Fallo del sistema | Un fallo del sistema es la incapacidad del sistema para realizar una tarea, es decir, el resultado obtenido difiere del resultado que se esperaba obtener. |
| Fastboot | El fastboot es un protocolo estándar de Android que permite modificar la memoria flash de los terminales a través de USB. |
| *Flash* / "Flashear" | La acción de instalar una imagen en la partición correspondiente del dispositivo. |
| Gerrit | Gerrit es una aplicación web para la revisión de código por pares, desarrollada por Google, que utiliza Git como Sistema de Control de Versiones. |
| Imagen | Es el nombre que reciben los archivos que contienen una copia de una partición del sistema de archivos, que puede ser copiada al dispositivo mediante alguna herramienta. |
| Integración | La integración consiste en juntar los distintos artefactos sofware para formar un sistema. |
| Issue | Una issue o incidencia es cualquier comportamiento inesperado de un software que provoca un resultado inesperado. Influye en la calidad. |
| Jenkins | Jenkins es un servidor de integración contínua escrito en java y basado en el proyecto Hudson. |
| Línea de producto | Diagrama que muestra la personalización en un proyecto software. Muestra por un lado el núcleo invariable y por otro las variantes que se |

| | |
|---|---|
| software | añaden para construir productos software personalizados y adaptados a las necesidades del mercado. |
| Middleware | Software encargado de conectar una aplicación con todo aquello que necesite (otras aplicaciones, redes, SO, etc.). El uso de este software conlleva una mejora de la calidad de las comunicaciones, la seguridad, etc. |
| NDK | El NDK de Android consiste en varias herramientas que posibilitan embeber código máquina (nativo) compilado en lenguajes C y/o C++. |
| NFC | (*Near Field Communication*) Tecnología de comunicación por proximidad sin contacto. |
| OTA | (*Over-the-Air update*) Es como se llama al despliegue automático de nuevas versiones de Android en dispositivos a través de Internet y desde el propio dispositivo, sin necesidad de conectarlo a un ordenador ni usar software adicional. |
| *Patch* / Parche | Un conjunto de cambios atómicos realizados en el código fuente que se envían a Gerrit para su revisión y posible aceptación. |
| (Sub) Proyecto | Un subproyecto (o proyecto por acortar) de AOSP es una parte del mismo destinada a una función concreta, con su propio repositorio separado del resto. En total AOSP se compone por 420 proyectos (en la versión 4.4.2). |
| Prueba | Proceso que busca examinar el software buscando un correcto funcionamiento, o en caso de un mal funcionamiento, dar información para su solución. |
| Rama (repositorio) | Una rama es una línea de desarrollo independiente que puede compartir parte de historia común con otras.[7]<br>En el caso de AOSP, una rama se utiliza para diferenciar el desarrollo de una versión concreta. |
| Repo | Herramienta escrita en Python utilizada para gestionar los diversos repositorios Git de AOSP, facilitando tareas repetitivas y el trabajo con Gerrit. |
| Reporte | Descripción detallada de un error. Normalmente se aporta un título, descripción, secuencia de pasos que hacen que se produzca el error y una traza del mismo. |
| Repositorio | Lugar donde se lleva la gestión del código fuente (control de versiones, ramas, etc.) para cuyo uso se emplea una herramienta. |
| | Es como se conoce al archivo que se emplea para instalar una construcción determinada de Android. Recibe su nombre debido a que la |

[7] Diapositivas de la asignatura: Tema 3 - Gestión del código fuente

| | |
|---|---|
| ROM | instalación se realiza sobre las memorias de tipo ROM (Read-Only Memory). |
| Root | En sistemas Unix es el nombre convencional del usuario con permisos de administración.<br><br>En el ecosistema Android, se llama así al estado de un usuario cuando el mismo tiene permiso de administración, y por tanto puede modificar cualquier parte del sistema operativo, asumiendo los riesgos que ello conlleva. |
| SDK | Conjunto de herramientas para el desarrollo en la plataforma Android. |
| Versión | Etiqueta que sigue una determinada política e indica un cambio en el software. |

Tabla 1 Glosario de Términos

# 8. Gestión del Código Fuente

La gestión del código fuente pretende evitar el caos, encargándose de mantener organizado el código fuente, centralizado, seguro, y controlado las distintas versiones del mismo. Todo ello se realiza a través de una herramienta que se conoce como repositorio. En nuestro caso, Android está alojado en un repositorio y cuyo estudio se encuentra a continuación.

Para la gestión del código fuente AOSP utiliza repositorios tipo Git a través de Repo, una herramienta para la gestión de repositorios que funciona sobre Git. Cada vez que se hace un cambio, éste se gestiona a través de Gerrit, una herramienta de gestión de parches. Cuando se usa Repo para subir un cambio, esta herramienta lo envía directamente a Gerrit para ser aprobado y pasarlo posteriormente a la rama del repositorio correspondiente.

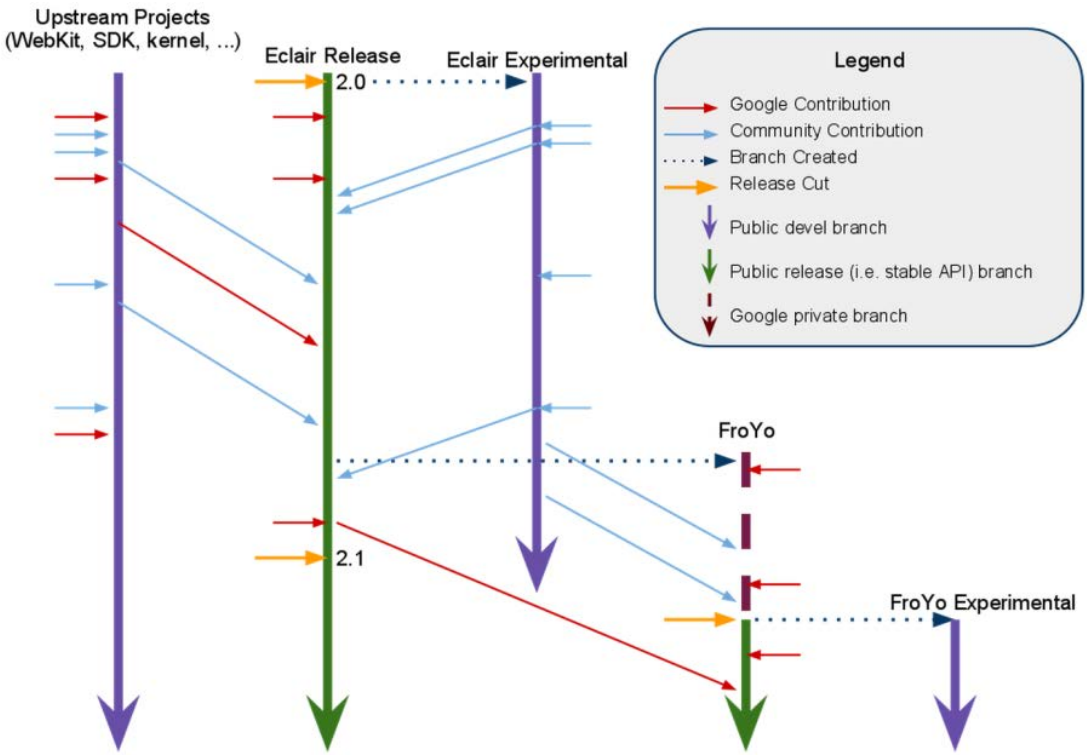
En este apartado vamos a estudiar aspectos relacionados con la gestión del código fuente cómo son: la estructura de ramas del proyecto, la gestión de contribuciones, los roles del proyecto y el seguimiento del código.

## Estructura de ramas del proyecto

El código de AOSP sigue un desarrollo iterativo, basado principalmente en 3 "líneas de código" paralelas para cada versión. Cada línea puede estar compuesta de una o más ramas en el repositorio. Hay que tener en cuenta que, aunque en alto nivel se trate el repositorio de código como uno solo, a más bajo nivel realmente cada subproyecto de AOSP (420 para la versión 4.4.2) constituye en realidad un repositorio Git independiente, pero respetando la misma estructura y nomenclatura de ramas entre ellos. La gestión de estos distintos repositorios se hace automáticamente gracias a la herramienta Repo que se explicará más adelante.

En todo momento existe una línea de código de "versión actual" de la plataforma, normalmente en forma de rama "master" en el árbol del repositorio. En esta rama se mantiene siempre una versión estable de la plataforma, y es de la que los fabricantes y desarrolladores deben tomar el código fuente para desarrollar dispositivos y software.

Paralela a ésta, existe una línea de código de "versión experimental" o de desarrollo correspondiente a la de "versión actual". Sobre ésta línea, fabricantes y contribuyentes en general pueden corregir errores, experimentar con nuevas funciones o incluir lo necesario para lanzar nuevos dispositivos. Los cambios que se demuestren estables se añadirán finalmente a la línea de código de la "versión actual", siempre se traten de arreglos de errores, mejoras, y cambios que no afecten a las APIs de la plataforma.




8. 1 Líneas de Desarrollo

Además de estas líneas públicas, Google trabaja internamente en la próxima versión de la plataforma sobre una línea de código privada según los objetivos propuestos para el producto por la propia compañía. Normalmente dicha versión se desarrolla junto con un fabricante de dispositivos para marcar la dirección que desean que siga el ecosistema Android. Los cambios realizados sobre la línea "experimental" se incorporan en esta línea de código privada según necesidad. Una vez la siguiente versión está lista, se publica como una nueva línea de código, que se convierte en la de "versión actual", y junto con ella se crea otra línea "experimental" correspondiente a dicha versión. En la documentación oficial proporcionan una explicación del porqué de esta línea privada a pesar de ser AOSP un proyecto libre[8]:

> *"The source management strategy above includes a code-line that Google will keep private. The reason for this is to focus attention on the current public version of Android.*
>
> *OEMs and other device builders naturally want to ship devices with the latest version of Android. Similarly, application developers don't want to deal with more platform versions than strictly necessary. Meanwhile, Google retains responsibility for the strategic direction of Android as a platform and a product. Our approach focuses on a small number of flagship devices to drive features while securing protections of Android-related intellectual property.*
>
> *As a result, Google frequently has possession of confidential information from third parties. And we must refrain from revealing sensitive features until we've secured the appropriate protections. In addition, there are real risks to the platform arising from having too many platform versions extant at once. For these reasons,*

[8] http://source.android.com/source/code-lines.html#about-private-code-lines

*we have structured the open-source project -- including third-party contributions -- to focus on the currently-public stable version of Android. "Deep development" on the next version of the platform will happen in private until it's ready to become an official release.*

*We recognize many contributors will disagree with this approach. We respect others may have a different point of view; however, this is the approach we feel is best, and the one we've chosen to implement."*

AOSP hace uso de multitud de proyectos de código abierto (de terceros o del propio AOSP pero gestionados de manera independiente), como son el Kernel de Linux, WebKit, el SDK para desarrolladores, Dalvik, y muchos más. Son denominados proyectos "upstream". Generalmente se desarrollan independientemente en repositorios públicos, y las contribuciones para Android se hacen sobre ellos. Periódicamente se incorporan nuevas versiones de estos proyectos tanto a la "versión actual" como a la línea interna de Google.

## Gestión de las contribuciones

Debido al carácter abierto del proyecto AOSP, usuarios y empresas que forman la comunidad pueden contribuir enviando "parches" o cambios al código fuente de AOSP. El proceso de contribución sigue dos etapas claramente definidas: el envío de parches y su aprobación.

**Envío de contribuciones**

Una vez configurado el entorno local y descargado el código fuente del proyecto o proyectos que se quieran modificar, se crea una "rama temática" en el proyecto con la herramienta Repo sobre la que modificar los archivos con el sistema de commit propio de Git.

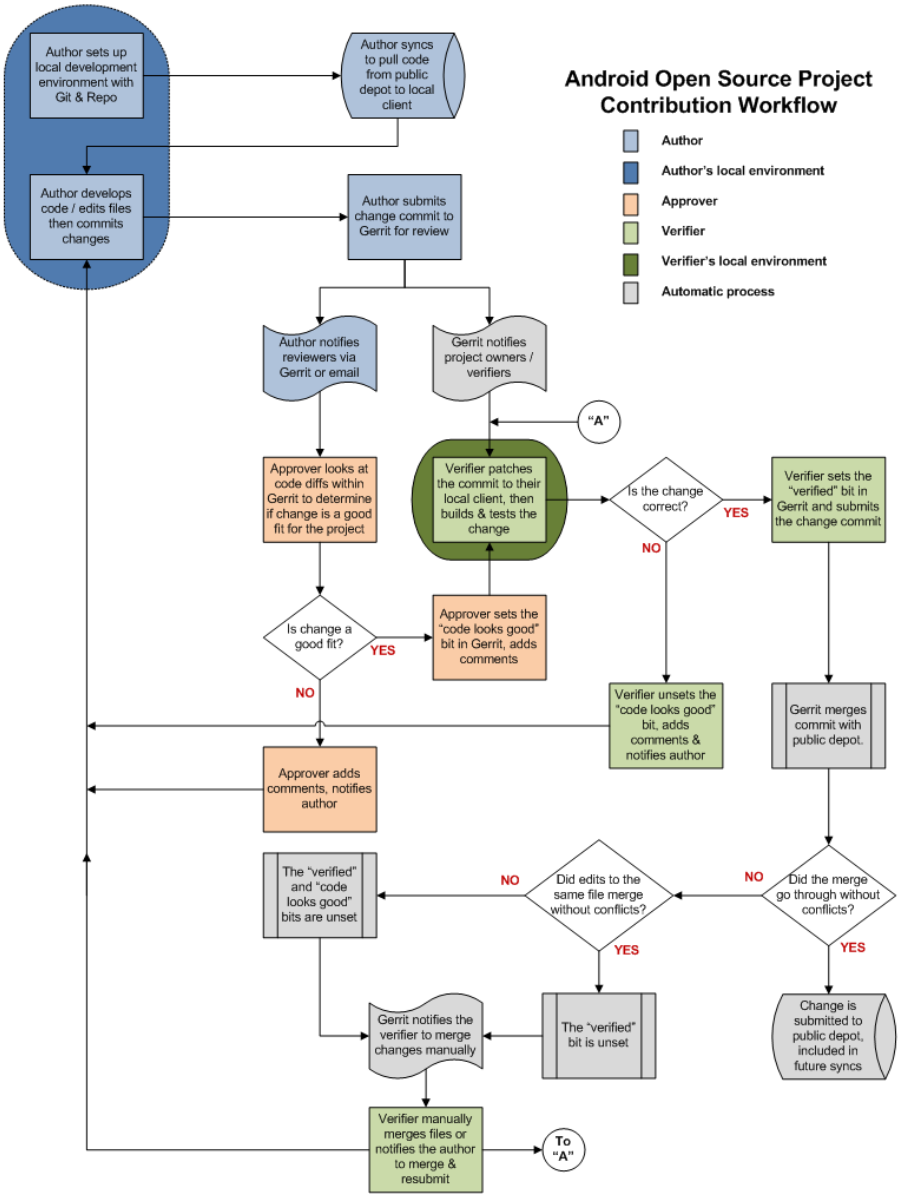
**Aprobación de contribuciones**

Cada parche sigue un ciclo de vida establecido antes de ser aceptado. Cualquier persona con las herramientas suministradas y el entorno de desarrollo configurado según se indica en la web del proyecto puede enviar un parche.

Una vez enviado, éste pasa dos procesos paralelos y necesarios en el sistema Gerrit. Los revisores comparan el código del parche para determinar si es un buen cambio para el proyecto, y en caso afirmativo marcan la bandera "el código luce bien". Por otra parte, los verificadores se encargan de descargar el código con el parche en su cliente local, compilar el proyecto y hacer las pruebas. Si el parche no pasa la verificación, se desmarca la bandera "el código luce bien" y se notifica al autor para que modifique lo necesario. Si la pasa, se marca la bandera "verificado" y se envía el commit al repositorio público y se intenta unir automáticamente a la rama correspondiente.

En este punto, si se producen conflictos se desmarca la bandera “verificado” y se notifica al verificador para que haga la unión manualmente o solicite al autor realizar los cambios necesarios y se repite el proceso. Una vez se une correctamente, el parche queda subido al repositorio público para futuras sincronizaciones.

Estos parches se aplican sobre la línea de código de versión experimental y se sincronizarán o no con la versión estable según decidan los responsables del proyecto.




8. 2 Flujo de Colaboración

## Seguimiento del código: Gerrit

Gerrit es un sistema de revisión de código basado en Web para facilitar las revisiones online de proyectos que usan el sistema de control de versiones Git. Según su página web[9]:

> *"Gerrit makes reviews easier by showing changes in a side-by-side display, and allowing inline comments to be added by any reviewer.*
>
> *Gerrit simplifies Git based project maintainership by permitting any authorized user to submit changes to the master Git repository, rather than requiring all approved changes to be merged in by hand by the project maintainer. This functionality enables a more centralized usage of Git."*

Se desarrolló para servir el código de AOSP, inicialmente como modificación de un software similar llamado Rietveld, que creó un ingeniero de Google como alternativa libre a Mondrian, otro software de revisión de pares desarrollado y usado internamente por la compañía, pero de código cerrado. Finalmente se decidió separar como un proyecto paralelo, y se le puso el nombre de Gerrit en honor al arquitecto Gerrit Rietveld. Actualmente es una potente herramienta usada no sólo por AOSP, sino también por otros proyectos relacionados con Android como CyanogenMod o AOKP, u otros como Chromium, Eclipse, LibreOffice o Wikimedia.

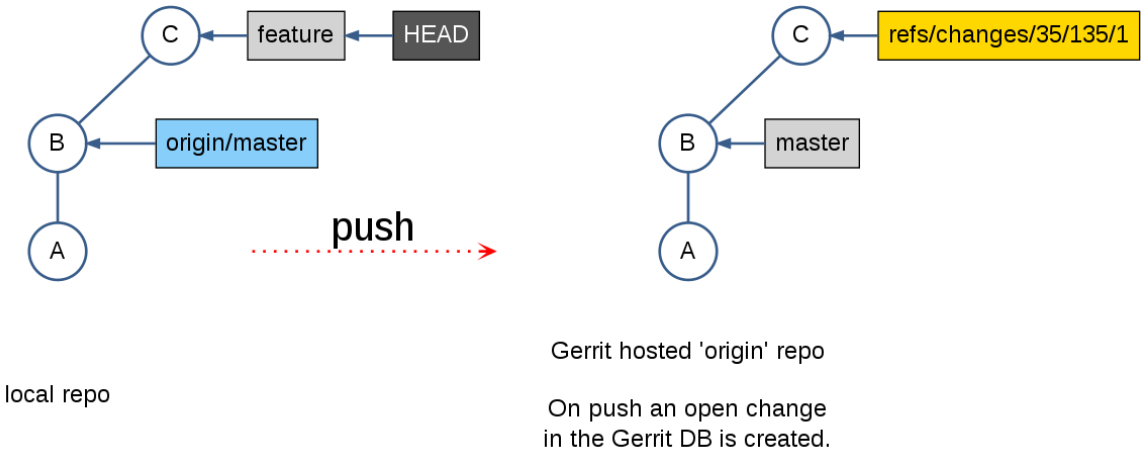
La decisión de usar este sistema para gestionar el código de AOSP fue debida a la amplia experiencia de los ingenieros de Google con el sistema de Mondrian y los repositorios Git.

**Funcionamiento de Gerrit**

Gerrit permite gestionar uno o más repositorios Git en un proyecto de manera centralizada, añadiendo control de acceso a cada uno. Su funcionamiento se basa en "parches" y "revisiones" de código.

El desarrollador registra sus commits en una rama local propia, llamada "feature branch". Para enviar un parche con los cambios en dicha rama, el desarrollador hace un push con Git al repositorio remoto de origen. Gerrit lo intercepta y crea una nueva rama para almacenar ese parche. Si el parche contiene más de un commit, se crea una rama para cada uno, y se establece la dependencia entre ellos para formar un conjunto de parches.

[9] https://code.google.com/p/gerrit/




8. 3 Funcionamiento de Gerrit

En la interfaz web, el resto de desarrolladores puede revisar el parche (o conjunto), haciendo comentarios, y estableciendo una puntuación. Las posibles valoraciones van entre -2 y +2:

- *+2 Looks good to me, approved*
- *+1 Looks good to me, but someone else must approve*
- *0 No score*
- *-1 I would prefer that you didn't submit this*
- *-2 Do not submit*

El rango de puntuación que a un usuario se le permite establecer es ajustable mediante el control de acceso según el rol del usuario. Además, los usuarios con rol de verificador pueden establecer votos de "verificado" con +1 ó -1. Son los encargados de comprobar que el código del parche funciona correctamente. Son necesarias puntuaciones altas tanto en revisión como en verificación para que se permita aceptar el parche.

Cada parche tiene asociada una URI de rama para que los revisores y verificadores puedan descargarlos y probarlos en su entorno local. Si un parche no pasa la revisión, el autor puede modificar el conjunto de parches con nuevos commits, incluyendo como metadato al final del mensaje de commit el identificador del parche.

Si finalmente se decide que el parche es suficientemente bueno y apropiado para ser añadido al proyecto, se hace una entrega (submit). Gerrit automáticamente intentará unir la rama del parche en la rama central mediante un merge de Git. Si se producen conflictos, por ejemplo si la rama principal ha evolucionado desde que se inició el parche, se notifica a la persona o personas encargadas para realizar un merge manual.

**Uso en AOSP**

AOSP utiliza Gerrit para gestionar la evolución del código fuente. Cada proyecto de AOSP es un repositorio Git y un proyecto en Gerrit. Cualquier persona con una cuenta de Google puede enviar un parche, comentar los parches existentes y puntuar la revisión con -1 ó +1. Además, hay un grupo de Android Maintainers formado por ingenieros de Google con permisos para verificar los parches, y otros grupos de verificadores específicos para algunos proyectos determinados.

Para la gestión local de los repositorios por parte de los desarrolladores se usa la herramienta Repo, que facilita las tareas repetitivas con Git. El uso de Repo se explica a continuación en este apartado.

## Repo

El repositorio de AOSP está construido sobre Git con una interfaz web llamada Gerrit para la revisión del código y aprobación de los parches. Tal como se explica en el apartado de Gestión del Código Fuente, AOSP está compuesto de múltiples proyectos, cada uno con su propio repositorio, por lo que administrar el código sólo con Git sería complicado y laborioso.

Para facilitar esa tarea se creó Repo, un script realizado en Python que actúa como una capa superior a Git, ya que nos abstrae de ejecutar comandos para cada repositorio individual. De este modo, con una sola orden podemos gestionar y sincronizar el código fuente de uno, varios o todos los proyectos.

El uso directo de Git se limita mayormente a la hora de trabajar localmente para crear commits o usar ramas propias durante el desarrollo local, y Repo para enviar los cambios o actualizar el código del repositorio online.

### Comandos de Repo

A continuación explicamos los comandos más de la herramienta Repo. El esquema de uso es:

*repo COMANDO OPCIONES*

| Comando | Descripción |
|---|---|
| help COMANDO | Proporciona ayuda detallada sobre cualquier comando. |
| init -u URL [OPCIONES] | Instala Repo en el directorio actual, generando un repositorio Git para cada proyecto especificado en el archivo manifest.xml de la URL. Como parámetro opcional se selecciona una rama del repositorio. |
| sync [PROYECTOS] | Descarga el código de los proyectos especificados (todos por defecto) mediante *pull* de Git. |
| upload [PROYECTOS] | Envía los commits realizados en los proyectos indicados (todos por defecto) para su revisión en Gerrit. |
| start RAMA [PROYECTOS] | Comienza una nueva rama para el desarrollo local en los proyectos indicados (todos por defecto). |

Tabla 2 Comandos Repo

## Roles

Estos son los roles existentes en AOSP para trabajar sobre el repositorio.

### Contributor

Los Contributor o Contribuidores traducido al español son las distintas personas que contribuyen desarrollando para AOSP, dentro de esta categoría se incluye también a los empleados de las distintas empresas que colaboran en el proyecto no habiendo distinción con los Contribuidores que no trabajan para ninguna empresa.

### Developer

Una distinción clara para diferenciar un Developer y un Contributor es que los Developers son usuarios de AOSP y desarrollan para la plataforma y Contributor desarrollan AOSP.

### Verifier

Los Verifiers son los encargados de realizar las pruebas y verificar los cambios propuestos. Este rol es asignado por el Lider del proyecto una vez el usuario ha colaborado continuamente y con códigos considerado de calidad.

### Approver

Los Approvers son usuarios con una larga experiencia de colaboración en AOSP y que han demostrado sus cualidades con aportaciones técnicamente significativas. Este rol es asignado por el Líder del proyecto y su misión principal es decidir cuáles cambios se aplican y cuáles no.

### Project Lead

AOSP está compuesto por proyectos más pequeños, teniendo cada uno de ellos un Project Lead o Líder del proyecto. Estos usuarios suelen ser trabajadores de Google y tienen una larga experiencia como desarrolladores. Como líder de un proyecto sus tareas son:

- Administra todos los aspectos técnicos del proyecto, la línea de ruta, el desarrollo, los ciclos de despliegue, el versionado y el aseguramiento de la calidad.
- Asegurarse de que el proyecto es probado por el Aseguramiento de la Calidad para las fechas de los releases de Android.
- Designar los Verifiers y los Approvers para los parches enviados.
- Ser justo e imparcial analizando los cambios. Aceptar o rechazar un parches basado en méritos técnicos y su alineación con la estrategia de Android.
- Analizar los cambios de manera oportuna y realizando el mejor esfuerzo comunicando cuando un cambio no es aceptado.
- Opcionalmente mantener una web del proyecto para información y documentos relacionados.
- Actuar como moderador en la resolución de conflictos técnicos.
- Ser la cara pública del proyecto y la persona que resuelva las dudas relacionadas con él.

# 9. Gestión de la Construcción

La gestión de la construcción consiste en decidir cómo se va a construir/formar el software a partir del código. El software, como sabemos, es complejo y durante su desarrollo se han de tener en cuenta complicaciones: equipo de trabajo distribuido, varias construcciones y entregas, y uso de librerías externas al proyecto.

Está estrechamente relacionado con la gestión del código fuente (entre otros aspectos de la evolución y gestión de la configuración) ya que el código se encuentra en un repositorio, el cual debe estar organizado (ramas, etc.) para facilitar entre otras cosas la construcción.

En este apartado estudiamos cómo realizar la construcción en AOSP seleccionando una rama determinada del repositorio que contendrá (según la política de gestión de código fuente de AOSP) con una versión estable (o no) de Android, junto con las librerías necesarias.

Cuando estamos realizando la construcción se produce una integración. El software dada su complejidad esta desarrollado por módulos, paquetes, etc. y para obtener el producto software necesitamos indicar e integrar las partes correspondientes. De hecho, la construcción es una de las patas sobre las que se sostiene la integración.

## Construcción del proyecto

Vamos a describir los pasos que se siguen a la hora de realizar una construcción de una versión cualquiera de Android. Para ello vamos a usar las herramientas comentadas en el apartado anterior. La construcción del código de AOSP debe hacerse sobre los Sistemas Operativos Linux o Mac OS, las herramientas no están disponibles para Windows.

**Inicialización del entorno de construcción**

En primer lugar, debemos elegir la rama del repositorio ("*branch*") que deseamos compilar, dado que diferentes ramas/versiones de AOSP tienen distintos requisitos en cuanto a herramientas.

Vamos a especificar únicamente las instrucciones de configuración para un entorno tipo Linux. Concretamente la distribución Ubuntu 12.04, pues es la recomendada oficialmente; aunque la mayoría de distribuciones tienen disponibles las herramientas necesarias comentadas anteriormente.

Realizamos la instalación de las herramientas que no estén en el sistema operativo y/o los paquetes que vamos a necesitar. Además, si la compilación se va a desplegar en un dispositivo físico, hay que configurar el acceso a los USB para el usuario, debido a que la configuración por defecto sólo permite acceso *root*. Opcionalmente, se puede configurar *Ccache*, una herramienta que ayuda a acelerar las compilaciones sucesivas.

Si tenemos varios volúmenes de almacenamiento, podemos declarar un directorio de salida separados para que la construcción se almacene en un volumen distinto, lo que acelera bastante el proceso. Más aún si el volumen de destino está optimizado para velocidad.

**Descarga del código fuente**

En primer lugar debemos contar con la herramienta Repo instalada, que es una interfaz sobre Git utilizada para gestionar la sincronización del repositorio de AOSP.

Una vez instalado, se inicializa el repositorio configurando a la vez la rama que queremos sincronizar, por defecto "*master*". Debemos configurar el usuario para el repositorio con nuestro nombre real y la dirección de correo que utilizaremos posteriormente en Gerrit, lugar donde se gestionan las aportaciones. Opcionalmente además podemos definir la lista de proyectos individuales que queremos descargar, por defecto descargará todos los importantes.

```
$ repo init -u https://android.googlesource.com/platform/manifest -b android-4.0.1_r1
```

Para descargar el arbol del código de AOSP tenemos que hacer *sync* con Repo, . Cada subproyecto de AOSP forma un repositorio de Git propio, por lo que descargar y sincronizar manualmente cada repositorio sería una tarea pesada. Este comando descarga el código completo para cada proyecto de la rama elegida anteriormente de manera ordenada.

El repositorio tiene una cuota limitada de acceso por IP, para evitar la sobrecarga de conexiones por un mal uso. Opcionalmente, podemos usar autenticación para sobrepasar el límite, sobre todo si se trabaja sobre el repositorio masivamente desde una organización compartiendo la misma IP detrás de un NAT, entre otros escenarios. Se usa la cuenta de Google, y la contraseña se genera en https://android.googlesource.com/new-password

**Construcción y ejecución**

Podemos construir cualquier rama del proyecto AOSP de la forma que se detalla a continuación.

Una vez seleccionada la versión necesitamos inicializar el entorno. Para ello se proporciona un script que configura en la consola las herramientas propias usadas en la compilación, mediante el comando "source build/envsetup.sh". Tras esto, estarán disponibles comandos especiales para la construcción, especialmente "lunch".

*Lunch* es el comando con el que se selecciona el objetivo de construcción. Es decir, configura el entorno para construir e integrar diferentes partes según el dispositivo de destino. En AOSP se incluyen algunas configuraciones genéricas para las arquitecturas soportadas (arm, mips, x86…) además de los dispositivos Nexus de Google. Para construir el código para otros fabricantes es necesario usar las configuraciones y archivos binarios que proporcionan estos.

Tras seleccionar el objetivo, se ejecuta la construcción con el comando *make* -jX (donde X es el número de trabajos paralelos que pueden correr en el procesador). Una vez finalizado (unas horas después) se genera el archivos de imagen *system.img*, que puede ser instalado (*flasheado*) en el dispositivo correspondiente con la herramienta *Fastboot*.

Alternativamente, se pueden compilar proyectos de manera independiente, como las aplicaciones del sistema. Por ejemplo, para compilar la calculadora se ejecutaría *make Calculator -jX*.

# 10. Gestión de Despliegue

El despliegue podemos decir que consiste en instalar el software en el entorno conocido como producción, es decir, entorno real donde ha sido contemplado durante su desarrollo.

Previamente requiere de una construcción-integración, pero no necesariamente de pruebas en caso de que estemos realizando las propias pruebas.

En AOSP este apartado requiere de una serie de matices debido a la particularidad del proyecto. Como ya se ha comentado en otros apartados AOSP contempla, al menos de manera pública, la gestión del código fuente y la gestión de incidencias, no existe un proceso único, unas herramientas y soporte para llevar a cabo el despliegue. Por tanto, lo que ocurre es que cada miembro de AOSP o bien cualquier que quiera hacer una construcción y un despliegue tiene que hacer esto último manual o automáticamente.

En este apartado vamos a estudiarlo desde el punto de vista de un dispositivo Nexus, dispositivos Android de Google.

## Sistema de Despliegue: Protocolo Fastboot

Fastboot es el protocolo de diagnóstico incluido en el SDK de AOSP que permite modificar el sistema de archivos flash del dispositivo. Es el protocolo oficial de Android y que incluye la mayoría de dispositivos.

Para su funcionamiento requiere iniciar el dispositivo en modo bootloader, conectado a un ordenador por USB con las herramientas del SDK instaladas, mediante una combinación especial de teclas (que varía según el dispositivo) o mediante el comando “adb reboot bootloader”.

Una vez conectado al dispositivo en Fastboot, se pueden borrar, formatear o flashear las particiones de de la memoria (recovery, boot, system) a partir de un archivo de imagen de la partición correspondiente.

Además, en algunos dispositivos Fastboot es capaz de “desbloquear el bootloader”, es decir, permitir instalar imágenes de particiones sin firmar por el fabricante original. Gracias a esto, los usuarios pueden instalar recoverys modificados con funciones avanzadas como modificar partes del sistema, o guardar y restaurar copias de seguridad de otras particiones (datos, memoria externa, etc); o instalar imágenes del sistema modificadas y sin firmar por el fabricante, conocidas como “ROMs” o “ROMs cocinadas”, construidas por otras compañías o comunidades de desarrollo. El desbloqueo del bootloader puede ocasionar la anulación de la garantía para la mayoría de fabricantes, y por seguridad el proceso obliga a borrar todos los datos almacenados en el dispositivo.

Pero aparte de Fastboot, algunos fabricantes incluyen en sus dispositivos sistemas de actualización automática que facilitan el proceso para usuarios con menos conocimientos técnicos. Estos sistemas se cubren en el apartado de Integración y Despliegue Continuos.

## Plataformas compatibles

Android está diseñado y enfocado principalmente a dispositivos con un procesador de arquitectura ARM, aunque hay proyectos funcionando en arquitecturas MIPS y x86 aunque no de manera oficial por parte de Google.

Entre los dispositivos donde encontramos AOSP y podemos desplegarlos encontramos smartphones, tablets, centros multimedia, sistemas embebidos, gafas interactivas, televisores, etc. Como vemos prácticamente no hay dispositivos sobre el que AOSP no pueda ser desplegado para su funcionamiento.

# 11. Gestión de la Variabilidad

La gestión de la variabilidad busca que el software sea producido en masa y que a su vez sea personalizable, lo que hace que los productos software se ajusten mejor a lo que necesitan y quieren los clientes.

Es un nuevo paradigma dentro del desarrollo software, conocido como Línea de Producto Software (en inglés SPL), que consiste en tener un núcleo e ir añadiendo características al mismo de modo que vayan surgiendo productos parecidos enfocados a distintos clientes u objetivos.

El problema es que la variabilidad es difícil de modelar, ya que el software tradicionalmente se ha estructurado en capas y con este nuevo enfoque es difícil encajar la variabilidad debido a que esta se organiza de manera matricial asociando características (*features*) a los distintos productos que se quieren obtener aplicando capas de personalización al núcleo.

AOSP es un claro ejemplo de producto personalizable, por lo que en este proyecto importa mucho llevar una gestión de la variabilidad. AOSP consiste en un núcleo formado por varios subproyectos de mayor importancia (núcleo de Linux, Maquina Virtual, etc.) a la que se añaden capas de funcionalidad variable según el hardware (NFC, capacidad telefónica, cámara trasera y/o frontal, botones físicos o virtuales, etc.) y el software (diferentes tipos de escritorios, aplicaciones personalizadas del fabricante, etc.).

Por tanto, AOSP tiene unas características que son comunes en todas las construcciones que se hagan para todos los dispositivos, pero otras características concretas son añadidas o suprimidas en otras construcciones según la finalidad de la misma.

## Análisis del dominio y modelo de variabilidad

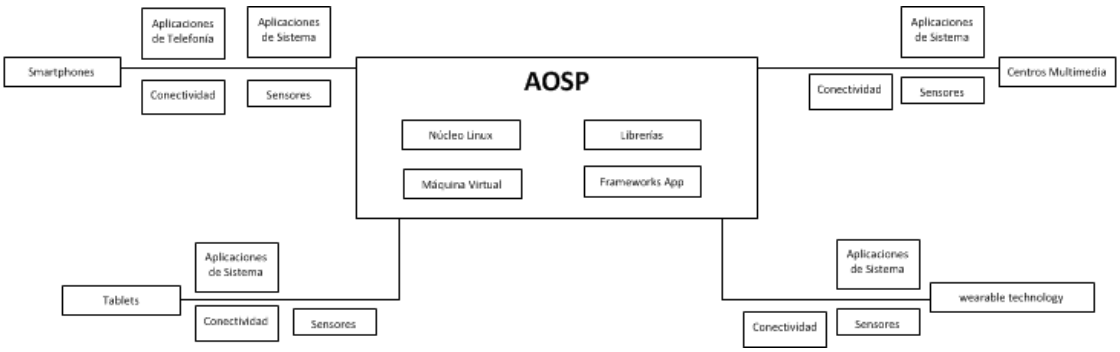
En AOSP, dada su envergadura y el mercado hacia el que está orientado, las características son muy dependientes del hardware y del fabricante. Es decir, nos encontramos con características que dependen del dispositivo para el que se quiere producir el producto final, y por otro lado de la capa de personalización o los cambios que introduzca la marca o el desarrollador.

La capa de personalización se corresponde mayormente con las aplicaciones instaladas de fábrica, tiendas de aplicaciones, funciones extra, etc. que decida quien realiza la construcción de la versión de Android, dependiendo del dispositivo y el mercado al que se orienta.

AOSP gestiona esta variabilidad a través de la modularización del código, encontrando en él distintos paquetes que se integran cuando se realiza la construcción, incluyendo los paquetes necesarios según las características finales que se quieran obtener.

## Aproximación de línea de producto software

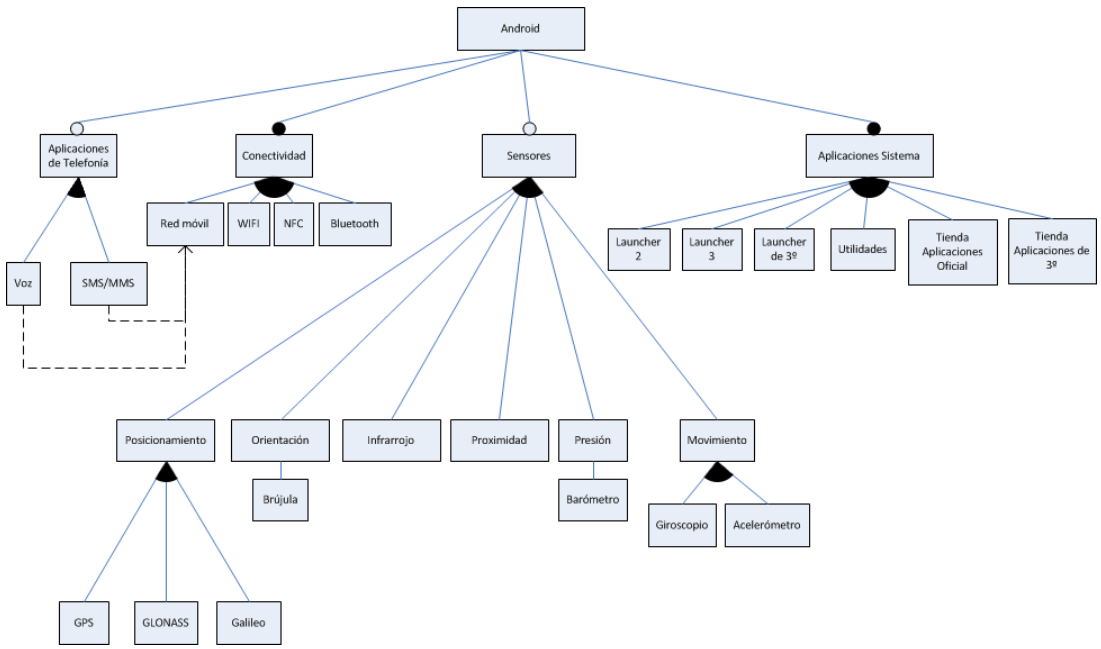
En el siguiente diagrama se muestran 4 ejemplos de dispositivos para los que se utiliza Android, y observamos cómo los paquetes que intervienen en la construcción varían según el dispositivo para el que esté orientado ésta. Por ejemplo, se mantiene un núcleo (Kernel de Linux, librerías básicas, entorno de ejecución virtual, framework de aplicaciones) y para los smartphones se añaden las aplicaciones necesarias hacer uso de características de telefonía (llamadas, SMS, conexión a Internet por red móvil, etc.).




11. 1 Línea de Productos Software

## Diagrama de modelado de características

Aquí se observan con más detalles algunas características variables que posee AOSP y cuáles son sus subcaracterísticas, detallando lo que se ve en la aproximación de línea de producto software.



11. 2 Modelado de Características

# 12. Gestión de Incidencias y Depuración

En el apartado de incidencias trataremos lo referente a los bugs o sugerencias indicadas por los usuarios. Por incidencia entendemos la obtención de un error o un funcionamiento no esperado en el producto software.

Antes que nada hay que entender que este apartado sólo explica los bugs propuestos por los desarrolladores y por tanto no entra lo referente a los fallos que puedan tener los usuarios cuando estén ejecutando el sistema. Para este caso los usuarios ya tienen unos canales independientes donde poder reportar los errores.

AOSP dispone de una plataforma (pública) de seguimiento de fallos o problema donde cualquiera puede informar de la existencia de un fallo, y además AOSP utiliza la misma plataforma para que cualquier usuario (o un grupo de ellos) pueda solicitar nuevas funcionalidades dentro del propio proyecto.

## Roles

Los roles para la gestión de incidencias y depuración son los mismos que se encuentran en el subapartado **Roles** del apartado **8. Gestión del código fuente**.

En AOSP no hay distinción entre los roles para la gestión del código y para la gestión de incidencias.

## Seguimiento de las incidencias

AOSP tiene un sistema público de seguimiento de "*Issues*" (a.k.a. problemas) donde cualquier persona puede reportar un error o solicitar una nueva característica para AOSP.

La dirección de este sistema es: https://code.google.com/p/android/issues/list. Esta es una herramienta técnica dentro de la Open Source community, pero hay que saber que no es una herramienta de soporte/atención al cliente.

Cada error declarado sigue un ciclo de vida que se puede resumir:

1. El error es detectado y pasa a tener categoría "*New*".
2. Un miembro de AOSP conocido como mantenedor revisa periódicamente y prioriza el error. Los errores son priorizados en una de las 4 categorías: "*New*", "*Open*", "*No-Action*", o "*Resolved*".
3. Cada categoría incluye un número de estados que aportan más detalles sobre lo que puede suceder con el problema.

4. Los errores que están en la categoría "Resolved" serán incluidos eventualmente en una futura versión de Android que será lanzada.

Respecto a la categoría "*New Issues*" consiste en nuevos reportes de error sobre los que no se ha realizado ninguna acción. Hay dos estados para esta categoría:

- "*New*": el error no ha sido detectado anteriormente (esto es, revisado por un mantenedor de AOSP).

- "*NeedsInfo*": El reporte de error no cuenta con la suficiente información para tratarlo. La persona que ha reportado el error debe proveer información adicional. Una vez pasado un tiempo suficiente sin proveer nueva información el error es cerrado por defecto y pasa a la categoría "*No-Action*".

Respecto a la categoría "*Open Issues*", ésta contiene aquellos fallos que requieren acción pero que todavía están por resolver y, además, están a la espera de un cambio en el código fuente. Hay dos estados para esta categoría:

- "*Unassigned*": El reporte de error ha sido reconocido como adecuadamente detallado pero aún no ha sido asignado a un contribuidor de AOSP para su corrección.

- "*Assigned*": Como "*Unassigned*", pero el error ha sido actualmente asignado a un contribuidor correspondiente de AOSP para su reparación.

Normalmente, un error comenzará en "*Unassigned*", donde permanecerá hasta que alguien tenga la intención de resolverlo, en cuyo caso pasará al estado "*Assigned*". Sin embargo pueden darse casos de bugs que pasen directamente de estado "*unassigned*" a "resolved".

En general, si un error está en uno de estos estados de "*Open*", el equipo que forma AOSP lo ha reconocido como un problema, y una contribución de corrección de gran calidad para ese error es probable que sea aceptado. Sin embargo, es imposible garantizar una solución a tiempo para una versión particular.

Respecto a la categoría "*No-Action Issues*" contiene errores que por alguna razón se ha determinado que no requieren ninguna acción. Contiene los siguientes estados:

- "*Spam*": "*A kind soul sent us some delicious pork products, that we, regrettably, do not want.*"

- "*Duplicate*": Ya existe un informe idéntico en el seguimiento de incidencias.

- "*Unreproducible*": Un colaborador AOSP intentó reproducir el comportamiento descrito, y fue incapaz de hacerlo. A veces, esto significa que el reporte de error es correcto pero que son difíciles de reproducir. Incluso, a veces, significa que el error se corrigió en una versión anterior.

- "*Obsolete*": Similar a irreproducibles, pero con la certeza de que existiera el error en la versión en la que se reporta pero ya se corrige en una versión posterior.

- "*WorkingAsIntended*": Un mantenedor AOSP ha determinado que el comportamiento descrito no es un error, sino que es el comportamiento previsto. Este estado también se conoce comúnmente como "WAI".

- "*Declined*": Es similar a "WorkingAsIntended", salvo que se utiliza normalmente para las peticiones de características en lugar de errores. Significa que un mantenedor AOSP ha determinado que la nueva funcionalidad solicitada no se va a implementar en Android.

- "*NotEnoughInformation*": El reporte no cuenta con la información suficiente para realizar acciones sobre el error.

- "*UserError*": El informe fue el resultado de un uso incorrecto de Android por parte de un usuario, por ejemplo, escribir una contraseña incorrecta y, por tanto, no ser capaz de conectarse a un servidor.

- "*WrongForum*": El informe no puede ser manejado dentro de AOSP, típicamente debido a que se relaciona con un dispositivo personalizado o a una aplicación externa.

- "*Question*": alguien ha confundido el seguimiento de incidencias con un foro de ayuda.

- Por último, la categoría "*Resolved Issues*" contiene los errores que han sido tratados y ahora son considerados como resueltos. Contiene dos estados:

- "*Released*": Este error se ha corregido y se incluye en una versión de AOSP ya liberada. Cuando se establece este estado se trata de indicar en qué versión se ha corregido.

- "*FutureReleased*": Este error se ha corregido (o la característica añadida) en el árbol del código fuente, pero la versión en la que se ha corregido (o añadido) no se ha liberado todavía.

Los estados y el ciclo de vida anterior son la forma en la que generalmente se realiza el seguimiento del software. Sin embargo, Android contiene una gran cantidad de productos software y obtiene un gran número de errores. Como resultado, a veces los errores no siguen todos los estados en una progresión formal. Se intenta mantener el sistema al día, realizando revisiones periódicas para actualizar los datos.

## Reportar un error

1. Para realizar un reporte correcto se siguen los siguientes pasos:

2. Consultar si el reporte del error ya ha sido creado por otro usuario de la comunidad.

3. Si encontramos la incidencia y para nosotros es importante, debemos hacer click en la estrella de dicho reporte para destacarla. De este modo, los responsables de Android son conscientes de las incidencias más relevantes para los usuarios.

4. Si nadie antes ha reportado nuestro error, debemos hacerlo nosotros. Podemos usar una de las siguientes plantillas:

    - User bug report
    - Developer bug report
    - Open-Source bug report
    - Feature request
    - Tools bug report
    - Tools feature request
    - Developer Documentation
    - Tools GPU bug report
    - Security bug report
    - Android Studio bug

En cada plantilla se indica para qué está destinada y los datos que se deben añadir para que el reporte se entienda y pueda ser solucionado.

Es aconsejable que los reportes de error tengan toda la información posible sobre el contexto del error, como por ejemplo el código del error, el código fuente sobre el que se trabaja, el comportamiento esperado, etc.

En lo que se refiere a los bugs de seguridad pueden ser reportados mediante la forma anterior o siguiendo un cauce más directo y privado. Según se indica en la FAQ de Android si un usuario encuentra un bug que esté relacionado con la seguridad tendría que enviar un correo electrónico a la dirección security@android.com. Los bugs de seguridad reportados mediante el formulario anterior no serán públicos como en el caso del resto tipo de bugs, sino que son estudiados y solucionados por el personal interno de Google y, una vez solucionado, se libera al público para que éste no pueda ser explotado malintencionadamente.

La corrección del bug se realiza de forma interna en el equipo de Google de Android, aunque realizan una comunicación asidua con el usuario que reportó el bug para mantenerlo informado sobre su avance. Una vez es corregido, éste es aplicado como un patch normal.

## Proceso de depuración

El proceso de depuración comienza en el momento en que un issue es asignado a un colaborador. Este colaborador será el encargado de solucionar o llevar a cabo lo que el issiues solicita. Este proceso se realizará según lo indicado en el subapartado **Gestión de las contribuciones** del apartado **8. Gestión del Código Fuente** de este documento. En dicho apartado se comenta desde el punto de vista de la aplicación de un "patch", pero es el mismo sistema para la gestión de incidencias, al igual que ocurre con los roles.

## Buenas y malas prácticas en los reportes

Reportar un error no es muy complejo pero a menudo se realiza de forma inadecuada. En la web de AOSP podemos ver dos ejemplos de reportes, uno realizado de forma correcta y otro de forma incorrecta, los cuales citamos y comentamos a continuación:

**Mal reporte**[10]

> Title: Error message
>
> When running Eclipse I get an "Internal Error" that says "See the .log file for more details".
>
> Steps to reproduce:
> Happens when "Object o = null". Doesn't happen when changed to "Object o".
>
> Expected results:
> I wouldn't get the error message--would work with Object o = null.
>
> Observed results:
> See above.

Como vemos el reporte no aporta mucho detalle sobre cómo se produce, por lo que es difícil de resolver para el encargado. Además no da información sobre el entorno de desarrollo, por lo que sería imposible saber si éste afecta o no. Este conflicto, por tanto, sería prácticamente imposible de solucionar.

[10] http://source.android.com/source/report-bugs.html#a-poor-bug-report

**Buen reporte**[11]

Title: Stepping over "Object o = null" causes Eclipse "Internal Error"

Interesting bug, while using Eclipse 3.3.1.1 with m37a of android and the following code:

```
package com.saville.android;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class TestObjectNull extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);

        Object o = null;

        o = "hi";

        Log.v(TAG, "o=" + o);
    }

    static final String TAG = "TestObjectNull";
}
```

Eclipse indicates an "Internal Error" with "See the .log file for more details" and then asks if I want to exit the workbench. This occurs when I place a break point on "setContentView(R.layout.main);" and then single step over "Object o = null;"

If I change "Object o = null;" to "Object o" all is well.

The last lines of the .log file are:

```
!ENTRY org.eclipse.core.jobs 4 2 2008-01-01 13:04:15.825
!MESSAGE An internal error occurred during: "has children update".
!STACK 0
java.lang.InternalError: Invalid signature: "<null>"
        at
org.eclipse.jdi.internal.TypeImpl.signatureToTag(TypeImpl.java:307)
        at
```

[11] http://source.android.com/source/report-bugs.html#a-good-bug-report

```
org.eclipse.jdi.internal.LocalVariableImpl.tag(LocalVariableImpl.java:185)
	at
org.eclipse.jdi.internal.StackFrameImpl.getValues(StackFrameImpl.java:128)
	at
org.eclipse.jdi.internal.StackFrameImpl.getValue(StackFrameImpl.java:73)
	at
org.eclipse.jdt.internal.debug.core.model.JDILocalVariable.retrieveValue(JDILocalVariable.ja
va:57)
	at
org.eclipse.jdt.internal.debug.core.model.JDIVariable.getCurrentValue(JDIVariable.java:66)
	at
org.eclipse.jdt.internal.debug.core.model.JDIVariable.getValue(JDIVariable.java:88)
	at
org.eclipse.debug.internal.ui.model.elements.VariableContentProvider.hasChildren(VariableC
ontentProvider.java:62)
	at
org.eclipse.jdt.internal.debug.ui.variables.JavaVariableContentProvider.hasChildren(JavaVaria
bleContentProvider.java:73)
	at
org.eclipse.debug.internal.ui.model.elements.ElementContentProvider.updateHasChildren(El
ementContentProvider.java:223)
	at
org.eclipse.debug.internal.ui.model.elements.ElementContentProvider$3.run(ElementConten
tProvider.java:200)
	at org.eclipse.core.internal.jobs.Worker.run(Worker.java:55)
```

No es muy difícil ver la diferencia entre ambos reportes. Como podemos ver se detalla tanto el entorno de desarrollo como la clase donde se produce el bug, además de la salida completa que produce dicho bug. Con estos datos el encargado de solucionarlo podrá replicarlo y solucionarlo.

# 13. Gestión de Pruebas

Según la RAE, prueba se define como "*Hacer examen y experimento de las cualidades de alguien o algo*"[12]. En todos los dominios, ingenieriles o no, se realizan pruebas. En el software son de vital importancia, todas la metodologías contemplan las pruebas (aunque cada una de una manera distinta) y el abanico de tipos de pruebas que existe es muy amplio.

Básicamente busca comprobar que lo que se está haciendo se comporta como se esperaba. La dificultad en el caso del software es que tiene que pasar pruebas de tipo funcional y no funcional, es decir, aspectos de seguridad, concurrencia, internacionalización, localización, etc.

El objetivo de las pruebas es encontrar fallos, errores, incidencias, etc de manera que estos se identifiquen y se solucionen. AOSP contempla este aspecto, y además lo hace de dos maneras distintas. Durante el desarrollo, como cualquier otro proyecto software, se realizan pruebas definidas en cada proyecto de manera individual. Además, existe el CTS o *Compatibility Test Suite*, que prueba que todas las partes de los diferentes proyectos funcionan entre sí y con compatibles con el ecosistema de Android. Si bien no se aclara de manera pública quién tiene qué hacer qué pruebas, se sabe que todos los parches aceptados mediante Gerrit deben haber sido debidamente probados antes por un *verifier*, tal como se explica en el apartado de roles en la Gestión del Código Fuente.

Por otra parte, AOSP tiene un sistema público de seguimiento de errores o "*Issue Tracker*", donde se provee un medio de reporte a los usuarios y a la comunidad de desarrollo. Nótese la diferencia respecto a las pruebas de software, ya que estas "pruebas" son realizadas por el usuario, normalmente al encontrarse un fallo o un comportamiento inadecuado durante el uso del software final.

En cuanto a los tipos de pruebas, en AOSP se llevan a cabo pruebas unitarias, pruebas de integración y sobre todo pruebas de compatibilidad (cabe mencionar que android se ejecuta en un amplio rango de distintos dispositivos). Podemos encontrar dichas pruebas en la suite de test de compatibilidad o CTS.

## Pruebas de compatibilidad: CTS (Compatibility Test Suite)

El CTS es un conjunto de pruebas automatizado compuesto por dos componentes software:

El CTS en sí, que se ejecuta en un ordenador y gestiona la ejecución de los test.

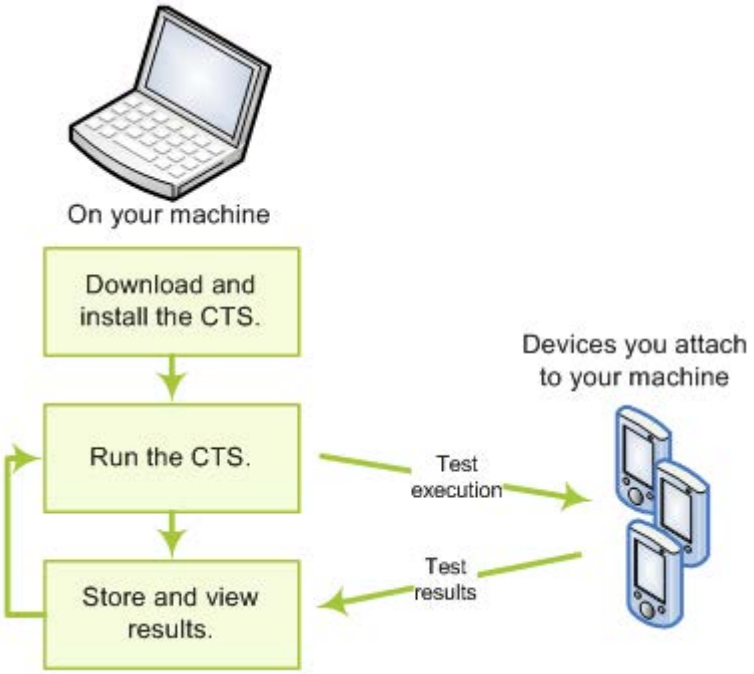
[12] Diccionario RAE

El conjunto de casos de test individuales que se ejecutan sobre los terminales conectados en el ordenador o sobre un emulador. Dichos casos de test están escritos en java como test unitarios de JUnit y empaquetados con el formato .apk para ser ejecutados en un dispositivo determinado.

El CTS puede complementarse con el CTS Verifier, el cual provee un conjunto de APIs para aquellas pruebas que no puedan ser realizadas sobre un dispositivo sin entrada manual (por ejemplo, pruebas sobre la calidad del audio, sobre el acelerómetro, etc.).

## Uso del CTS

La forma en la que se utiliza el CTS se expone en el siguiente diagrama:




13. 1 Ejecución de CTS

Cómo se puede observar en la imagen el primer paso es instalar el CTS en el equipo en el que se vayan a realizar las pruebas y a continuación lanzar las pruebas sobre los dispositivos deseados. Tras finalizar la pruebas se almacenan y visualizan los resultados de los test, los cuales informan sobre si se han encontrado incompatibilidades de software.

## Tipos de casos de prueba

El CTS incluye los siguientes cinco tipos de casos de prueba:

- **Pruebas unitarias.**
- **Test funcionales:** pruebas sobre combinaciones de APIs para comprobar su correcto funcionamiento cohesión con respecto a casos de uso de alto nivel.
- **Test de referencia a aplicaciones:** pruebas en forma de una aplicación de ejemplo completa para probar un conjunto completo de APIs y servicios de Android.
- **Test de robustez:** pruebas sobre la entereza del sistema bajo situaciones de estrés (alta carga de trabajo).
- **Test de rendimiento:** pruebas sobre el sistema frente benchmarks definidos (por ejemplo, de renderizado de fotogramas por segundo).

Cabe notar que los dos últimos tipos de pruebas (test de robustez y test de rendimiento) no están actualmente disponibles y se tiene intención de añadirlos en el futuro.

## Áreas cubiertas por los test.

Las áreas cubiertas por los test son las siguientes:

- Test de firma de las distintas APIs públicas.
- Test sobre las APIs de la plataforma concreta.
- Test sobre la máquina virtual Dalvik.
- Test sobre la plataforma del modelo de datos.
- Test sobre el mecanismo de intenciones (intents) de la plataforma.
- Test sobre los permisos disponibles en la plataforma.
- Test sobre los tipos de fuentes disponibles en la plataforma.

## Tipos de pruebas

Las pruebas de AOSP, como se ha expuesto anteriormente, se realizan creando pruebas a partir del CTS. Por tanto, las pruebas que se realizan son de tipo **gray-box**, ya que se tiene conocimiento sobre parte de la estructura del sistema (se tiene acceso a la estructura

del sistema, es decir, la aplicación desarrollada pero no se tiene acceso a la implementación de los métodos definidos en el CTS).

Se realizan pruebas tanto a nivel unitario, de integración como de sistema.

## Ejecución del CTS

A continuación se describe de forma resumida los pasos a seguir para realizar las pruebas del CTS según se indica en la web de AOSP[13]. Para saber más sobre CTS está disponible el manual en la siguiente dirección: http://source.android.com/compatibility/android-cts-manual.pdf

1. Download the CTS and CTS media files.
2. Attach at least one device (or emulator) to your machine.
3. For CTS versions 2.1 R2 through 4.2 R4, set up your device (or emulator) to run the accessibility tests:
    1. *adb install -r android-cts/repository/testcases/CtsDelegatingAccessibilityService.apk*
    2. On the device, enable Settings > Accessibility > Accessibility > Delegating Accessibility Service
4. For CTS 2.3 R4 and beyond, set up your device to run the device administration tests:
    1. *adb install -r android-cts/repository/testcases/CtsDeviceAdmin.apk*
    2. On the device, enable the two android.deviceadmin.cts.CtsDeviceAdminReceiver* device administrators under Settings > Location & security > Select device administrators

    **Note**: Make sure the android.deviceadmin.cts.CtsDeviceAdminDeactivatedReceiver stays disabled in the same menu.
5. For CTS 2.3 R12 and beyond, the CTS media files must be copied to the device's external storage. Check section 4.2 of the latest CTS manual for further details on copying these files:
    1. Unzip the CTS Media zip file.
    2. Run the following command. If no resolution is specified, the default maximum resolution of 480x360 is assumed:

    *copy_media.sh [720x480|1280x720|1920x1080|all] [-s serial]*
6. Launch the CTS. The CTS test harness loads the test plan onto the attached devices. For each test in the test harness:
    1. The test harness pushes a .apk file to each device, executes the test through instrumentation, and records test results.
    2. The test harness removes the .apk file from each device.
7. Once all the tests are executed, you can view the test results in your browser and use the results to adjust your design. You can continue to run the CTS throughout your development process.

[13] http://static.googleusercontent.com/media/source.android.com/es//compatibility/android-cts-manual.pdf

# 14. Integración y Despliegue Continuo

La integración continua surge porque el software es complejo, y como tal requiere probar los componentes desde el principio para detectar los problemas lo más pronto posible. Está relacionado con el despliegue puesto que se puede entender como el objetivo posterior a la integración.

Se basa en tres pilares: la construcción (privada, para integración y relacionada con una entrega o *release*), pruebas (automáticas/*smoke*) y obtener *feedback*.

AOSP es un proyecto muy genérico, dependiendo del dispositivo final necesita unos componentes u otros, por tanto no contempla ningún sistema de integración y despliegue continuo, al menos de manera pública, debido a que no se desarrolla para un único dispositivo. En AOSP se gestionan únicamente los arreglos y mejoras al código, por lo cual cada miembro de la comunidad se encarga de realizar la integración y despliegue del modo que considere y para los dispositivos que considere.

Describiremos cómo se haría de manera manual la integración y despliegue para un dispositivo determinado, y posteriormente hablaremos de cómo se podría hacer utilizando *Jenkins*.

## Integración continúa con Jenkins

Ya que AOSP no documenta información sobre cómo hacer la integración continua, ni de cómo hacen la integración y despliegues continuos fabricantes miembros del proyecto (Google, Samsung, etc.), nosotros vamos a mostrar cómo se podría hacer con Jenkins en un servidor propio. Para ello vamos a explicar una guia de IC aplicada a AOSP para desarrolladores de ROMs personalizadas.

La herramienta necesaria, a parte de la indicada en el entorno de desarrollo será **Jenkins** con el plugin **Repo Plugin**.

La construcción de AOSP es mediante *make* por lo que una vez configurado el *Job* en Jenkins habrá que especificar que ejecute una serie de comandos en consola a la hora de hacer el *build*. Lo recomendable seria ejecutar a través de Jenkins un *script* con los comandos que nos interese a la hora de construir el paquete. Con este sistema podríamos automatizar la construcción de AOSP, además si añadimos otros plugin de *Gerrit* podemos indicarle que ejecute *build* cada vez que se produce un cambio en el repositorio.

## Despliegue continuo

Al igual que la integración continua, AOSP no contempla ningún sistema de despliegue continuo puesto que es un proyecto enfocado a ofrecer solo el código fuente el cual ya cada usuario utiliza para construir el SO para su dispositivo.

Tal y como está pensado el proyecto no sería posible proponer un sistema de despliegue continuo, por ello vamos a hablar del sistema de despliegue continuo que utiliza Google con los dispositivos Nexus mediante el sistema OTA de actualizaciones.

OTA (Over The Air) es el sistema de actualizaciones que proporciona Google para que las organizaciones o empresas que utilicen este sistema operativo pueden implementar un sistema de actualización sin que el usuario tenga que preocuparse hacer nada. Para ello cada organización tiene que montar su propia infraestructura de servidores para poder ofrecer este servicio de actualizaciones.

El proceso de lanzamiento a los dispositivos es llevado a cabo mediante los servidores propios y para gestionarlo se lleva a cabo mediante lo que se conoce como "roll out", que consiste en que llegue progresivamente. Esto es importante para identificar y evitar que se propaguen posibles problemas que vayan surgiendo, normalmente se lanza para el 1% de los usuario, y se va incrementando este porcentaje a lo largo de los días hasta alcanzar el 100% en un periodo de 1-2 semanas de media.

La instalación de la actualización se lleva a cabo de manera automática, una vez se notifica al usuario la disponibilidad de la actualización, este acepta, descarga e instala la misma. Una vez descargado, el dispositivo se reinicia y procede a la instalación conservando los datos, configuraciones y demás aplicaciones del usuario.

Este proceso es el recomendado para usuarios que no tienen conocimientos avanzados, como por ejemplo del uso del protocolo fastboot detallado anteriormente.

Por último, comentar que otros fabricantes ofrecen los mismos sistemas de actualización, bien uno de los dos o ambos. Por ejemplo, HTC ofrece ambos métodos. El uso de uno u otro depende de la capacidad (infraestructura, etc.) que tenga dicho fabricante.

# 15. Gestión de Entregables

La definición de entregable podría ser cada uno de los productos (código fuente, scripts, documentación, etc.) que surgen en un proceso de desarrollo software. Tiene implícito una fecha de entrega.

AOSP, de manera pública, realiza la entrega del código fuente mediante la liberación del mismo a través de una nueva rama en el repositorio (que pasa a ser la rama por defecto). También entrega documentación, a través de una página web que es actualizado con cada liberación, entrega los binarios para que los desarrolladores de la comunidad pueden usar para hacer sus construcciones personalizables, y entrega los instaladores de la nueva versión.

A continuación vamos a estudiar cómo se identifican estos entregables y la política de versionado que se sigue.

## Código fuente

Dentro del proyecto AOSP se pueden identificar diferentes entregables. El principal, debido a la naturaleza del proyecto, es el código fuente. Periódicamente se realizan *releases* de nuevas versiones, que son desarrolladas de manera privada por Google, y volcadas al repositorio público de AOSP una vez presentadas en un plazo de pocos días.

En el repositorio, se mantiene la rama de la versión más reciente hasta la fecha, y se crea una nueva rama para la nueva versión, que pasa a ser la rama maestra. Para *releases* de revisiones menores que no varían la versión del código, se actualiza la misma rama y se crean *tags* en el repositorio para marcarlas.

Dependiendo del alcance de los cambios realizados se publicará con un número y nombre determinado, según la política de versiones detallada más abajo.

## Documentación

La documentación de AOSP se considera otro entregable más del proyecto, tanto la referente a AOSP como la documentación para desarrolladores de aplicaciones.

Normalmente, la actualizaciones en la documentación se liberan con cada nueva entrega del código fuente en la *línea de código* estable, es decir, con cada nueva versión de Android que se publica. La actualización incluye información sobre los cambios realizados en la plataforma, como nuevas APIs o herramientas. A veces, la liberación de la documentación se realiza antes que la del propio código fuente, cuando Google presenta al público la nueva versión de Android, que puede ser días antes de su liberación en AOSP.

Además, de forma periódica se hacen cambios en la documentación para mejorar la información, añadir ejemplos, corregir errores, etc. La liberación de estos cambios no sigue

un patrón ni una política establecidos.

## Herramientas

En AOSP también se incluyen como proyectos el SDK y las herramientas para desarrolladores.

El SDK se actualiza con cada liberación de una nueva versión de Android, pues incluye todo lo necesario para desarrollar aplicaciones para dicha versión. Además, el equipo responsable de las herramientas puede liberar actualizaciones de diferentes partes o herramientas del SDK según las desarrollen, sin depender de la publicación de versiones de Android.

Por ejemplo, el IDE de desarrollo principal para aplicaciones, el Android Studio basado en IntelliJ, es constantemente actualizado (cada pocos días) con los cambios que el equipo realiza en él, y utiliza una de versionado independiente de la plataforma, que no se especifica de forma pública.

## Imágenes de fábrica y Binarios

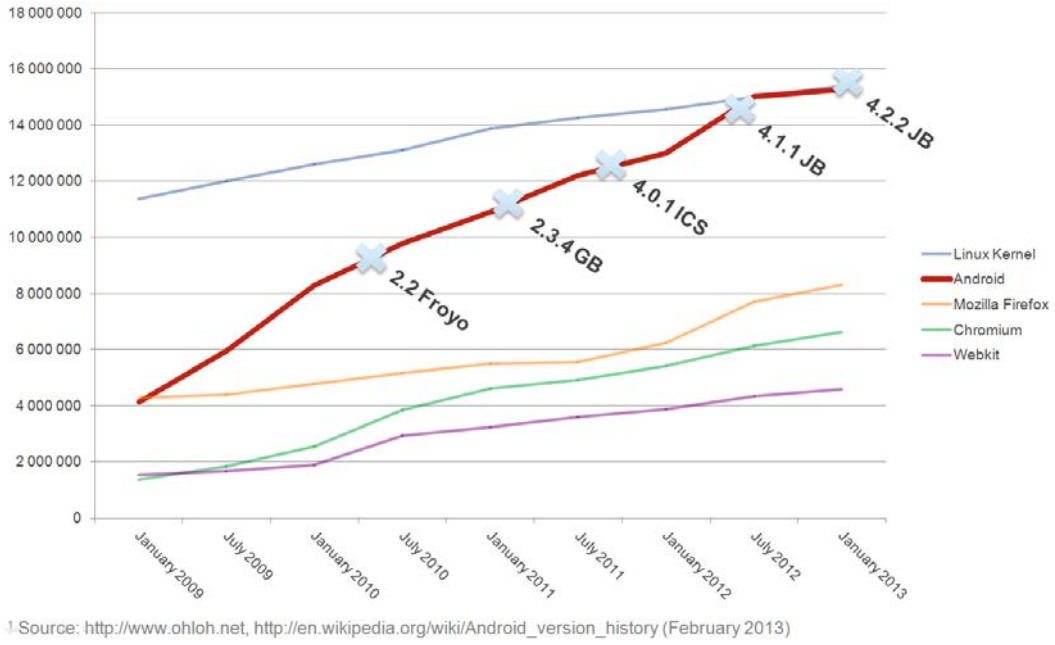
Otro de los entregables, por parte de Google, cuando hay una actualización son las imágenes de fábrica y los binarios.

La imagen de fábrica es un conjunto de elementos que son actualización del bootloader, la radio (relativo a las comunicaciones), el kernel de Linux y el instalador en sí mismo de la nueva versión. Todo ello para una instalación manual, si bien esta llegara a través del proceso OTA descrito en la gestión de despliegue.

Por último, los binarios se corresponden con los driver de los dispositivos Nexus para la versión a actualizar, por lo cual son públicos y pueden ser utilizados para la construcción de ROM personalizables para dichos dispositivos.

## Política de versiones

La política de versiones tiene una total relación con varios aspectos dentro de la gestión de la configuración, pero en concreto para la gestión de código fuente implica identificar los cambios que se dan en el software. Concretamente, la notación utilizada tiene un significado, y el repositorio es el que permite gestionar todo esto, de ahí que a los repositorios se les conozca también con el nombre de "Control de Versiones".




15. 1 Grafica de Versiones

En Android la política de versiones a lo largo del desarrollo utiliza varias notaciones. En primer lugar, cada versión es identificada con una secuencia de **número** del tipo X.Y.z, donde cada cambio de gran impacto en la plataforma supone un incremento del valor de X (inicialmente 1), y cada iteración de menor relevancia sobre esa versión supone un incremento del valor Y. Un incremento del valor z supone cambios mucho más pequeños, comúnmente dedicados a corrección de errores.

Además, cada versión tiene asociado lo que se conoce como **nivel de API**, que identifica un cambio sobre las API que ofrece el sistema operativo a los desarrolladores, y que se incrementa con cada nueva versión. No obstante hay otro elemento, conocido como NDK que corresponde al nivel de API para código nativo (escrito y compilado en C++, a diferencia del escrito en Java para la máquina virtual Dalvik), el cual no es modificado siempre con cada iteración del proyecto AOSP, sino que se incrementa a un ritmo inferior.

Por último, las distintas versiones de Android se asocian a un **nombre** más comercial, que no tienen una correspondencia directa con las políticas de versiones anteriores, sino que suelen modificarse cuando se producen cambios muy visibles para

los usuarios, normalmente referentes a la interfaz de usuario. Estos nombres son tomados de dulces o postres conocidos, y se incrementa en cada versión la letra por la que comienza el postre (por ejemplo, en la versión 4.3 se corresponde con la letra J por tanto el nombre finalmente fue "Jelly Bean", y para 4.4 correspondiente a la letra K el nombre designado fue "KitKat". Además, cada nombre tiene una nueva "mascota" o logo relacionada con dicho postre o dulce.

En lo que a compilaciones del código se refiere, se utilizan códigos identificativos para indicar la versión del código sobre la que se realizó, la rama del mismo y la fecha. Estos códigos son de la forma "FRF85B", donde la primera letra corresponde a la inicial del nombre de la versión (ej: F es "Froyo"); la segunda letra indica la rama del código que se utilizó (ej: R es "Release"); la siguiente letra y dos dígitos representan la fecha de compilación, donde la letra indica el cuarto de año, contado desde el Q1 de 2009.

En cuanto a los dígitos, éstos indican el día dentro de dicho cuarto (ej: F85 es el 24 de junio de 2010); y la última letra indica diferentes versiones correspondientes al mismo código de fecha, dado que es común que compilaciones con variaciones menores mantengan dicho código.

# 16. Mapa de Herramientas

## Lista completa de herramientas usadas en AOSP

- **Sistema Operativo Linux o Mac OS:** Se recomienda Ubuntu LTS 12.04, aunque la mayoría de distribuciones disponen de las herramientas necesarias. Windows no está soportado.

- **Git 1.7 o superior:** El sistema de control de versiones que utilizan los repositorios de AOSP es de tipo Git.

- **Repo:** Herramienta escrita en Python utilizada para gestionar los diversos repositorios Git facilitando tareas repetitivas, y facilita el trabajo con Gerrit.

- **Python 2.6-2.7:** Necesario para el uso de la herramienta Repo.

- **Gerrit**: Sistema de revisión de código basado en Web para facilitar las revisiones online de proyectos que usan el sistema de control de versiones Git.

- **GNU Make 3.81 - 3.82:** Herramienta que se encarga de construir a partir del repositorio local para un determinado dispositivo usando los compiladores adecuados para cada parte del proyecto.

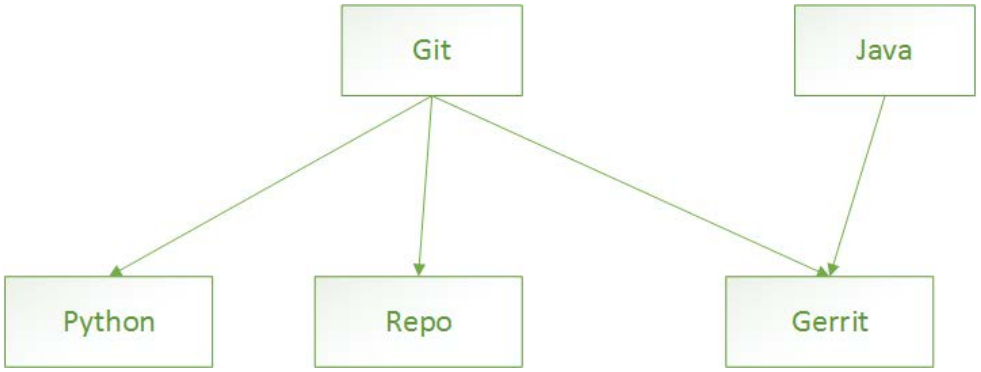
- **Java JDK 6:** Necesario para compilar la parte del repositorio que está escrita en Java.

- **GCC:** Compilador de C++ para el código nativo del proyecto.

- **Google Code:** Se utiliza un proyecto de Google Code exclusivamente para el *Issue Tracker* oficial de AOSP.

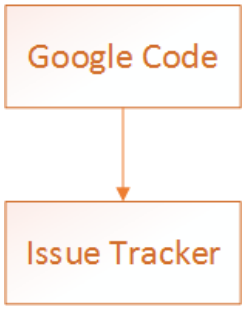
## Esquemas de relaciones

**Gestión del código:**



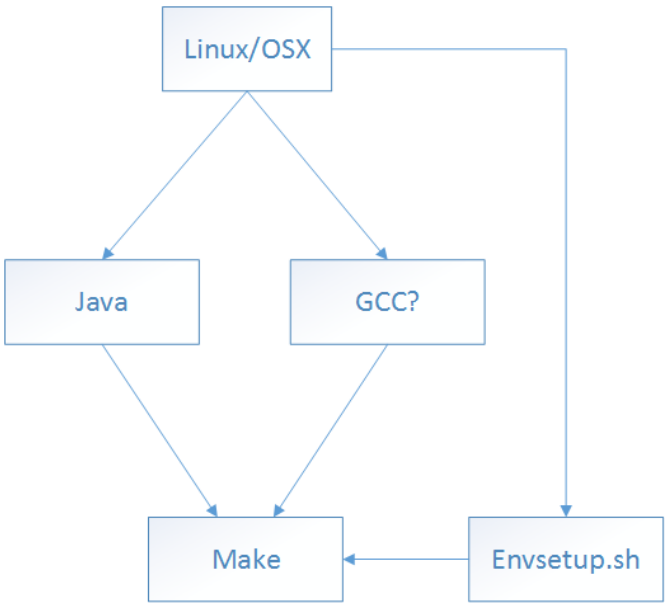

16. 1 Herramientas de Gestión del Código

**Gestión de incidencias:**




16. 2 Herramientas de Gestión de Incidencias

**Construcción:**




16. 3 Herramientas de Gestión de la Construcción

# 17. Ejercicios

A continuación indicamos una serie de ejercicios con su solución para los distintos apartados del documento.

## Gestión del código fuente

**Ejercicio 1**

Instalar, utilizando Ubuntu LTS 12.04, las herramientas necesarias para trabajar con el repositorio.

***Solución:***

*Trabajando sobre una instalación limpia de **Ubuntu 12.04**, es necesario instalar primero el **JDK** de Java. Para ello se realizan los siguientes pasos:*

*Añadir el repositorio de **WebUpd8**:*

```
$ sudo add-apt-repository ppa:webupd8team/java
```

*Instalar el **JDK**:*

```
$ sudo apt-get update
$ sudo apt-get install oracle-java6-installer
```

*Comprobar que la versión instalada es la correcta*

```
$ java -version
```

*Una vez instalado el **JDK**, hay que instalar los paquetes de las herramientas necesarias y sus dependencias, disponibles en los repositorios de **Ubuntu**. Según la documentación oficial:*

```
$ sudo apt-get install git gnupg flex bison gperf build-essential \
zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
libgl1-mesa-dev g++-multilib mingw32 tofrodos \
python-markdown libxml2-utils xsltproc zlib1g-dev:i386 \
libglapi-mesa:i386 libgl1-mesa-dri:i386

$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

*Por último, para instalar la herramienta **Repo** creamos una carpeta donde descargar el script y la añadimos al **Path**.*

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

**Ejercicio 2**

Descargar, haciendo uso de las herramientas descargadas del apartado anterior, la rama por defecto (ultima estable) del repositorio de AOSP. Al mismo tiempo, crear una nueva rama sobre el repositorio en local.

***Solución:***

*Con las herramientas del apartado anterior correctamente instaladas, creamos un directorio en el que descargar el código completo de la rama estable actual del repositorio.*

```
$ mkdir WORKING_DIRECTORY
```

```
$ cd WORKING_DIRECTORY
```

*Inicializamos el repositorio de manera que se instale el cliente de* ***Repo*** *en el directorio de trabajo con la configuración de los distintos repositorios en el proyecto* ***AOSP****.*

```
$ repo init -u https://android.googlesource.com/platform/manifest
```

*Y damos la orden para sincronizar los repositorios, es decir, descargar el código:*

```
$ repo sync -j16
```

*(Puede tardar varias horas, dependiendo de la velocidad de la red)*

## Gestión de la construcción

**Ejercicio 1**

Realizar la construcción del subproyecto de la calculadora ("*Calculator*"), teniendo descargado el código según el ejercicio 2 correspondiente a la gestión del código fuente, usando el comando *lunch* para configurar como objetivo de construcción un dispositivo ARM genérico y *make* para ejecutarlo.

***Solución:***

*Añadir los comandos de construcción al entorno con* ***envsetup.sh****:*

```
$ source build/envsetup.sh
```

*Seleccionar el objetivo de **lunch**.*

```
$lunch aosp_arm-en
```

*Lanzar la construcción del subproyecto:*

```
$make Calculator -j9
```

**Ejercicio 2**

Realizar la construcción del código fuente descargado en el ejercicio 2 correspondiente a la gestión del código fuente. Para ello, se podrá elegir libremente el dispositivo y demás configuraciones necesarias para la realización del ejercicio.

***Solución:***

*Añadir los comandos de construcción al entorno con **envsetup.sh**:*

```
$ source build/envsetup.sh
```

*Seleccionar el objetivo de **lunch**.*

```
$lunch <objetivo elegido(ej: full_hammerhead-userdebug>
```

*Lanzar la construcción de todo el proyecto*

```
$make -j9
```

*(puede tardar varias horas, se prevé que entre 5 y 8 dependiendo de la potencia del ordenador)*

## Gestión del despliegue

**Ejercicio 1**

Indique los pasos a seguir para realizar un despliegue en un dispositivo Android de una versión oficial determinada, indicando ambos

***Solución:***

*Vamos a decir que usamos un Nexus 5 como modelo, con la versión 4.4.2 de Android. El proceso sería igual para el resto de dispositivos Nexus.*

*Primero debemos descargar la imagen de fábrica desde la web oficial, y contar con el **SDK** correctamente instalado (incluyendo el funcionamiento del comando "**adb**") y la opción de depuración USB activada en las opciones de desarrollador en el dispositivo.*

*Para realizar el proceso es necesario tener "desbloqueado" el **bootloader**, por lo que si no se ha hecho antes habrá que desbloquearlo primero. Esto se lleva a cabo una sola vez, a menos que se vuelva a*

*cerrar. Con el dispositivo conectado, ejecutamos:*

adb reboot-bootloader

*, para reiniciar el dispositivo en modo* ***bootloader*** *y poder utilizar el protocolo* ***fastboot****.*

fastboot oem unlock

*, para proceder al desbloqueo. Esta operación borra los datos del dispositivo.*

*El* ***bootloader*** *se puede volver a bloquear con:*

fastboot oem lock

*Descomprimimos el archivo* ***.tgz*** *descargado, y en él encontramos los archivos correspondientes a las imágenes de cada partición, además de scripts para la instalación automática. Para hacerlo de la manera más didáctica posible vamos a ejecutar los comandos de forma manual, aunque se podrá comprobar que los scripts hacen lo mismo que vamos a realizar nosotros paso a paso.*

*Primero se actualiza la partición del* ***bootloader****, el gestor de arranque del dispositivo. En actualizaciones importantes de la versión de Android suele ser necesario tener una versión actualizada del* ***bootloader*** *para funcionar correctamente. Se instala con el comando flash:*

fastboot flash bootloader bootloader-hammerhead-hhz11k.img
fastboot reboot bootloader (reiniciamos tras cada flasheo)

*A continuación instalamos el firmware de radio, que es el encargado de controlar la conexión telefónica, redes móviles, GPS, Bluetooth, etc. Es específico para el hardware de cada dispositivo. Se instala con el comando flash:*

fastboot flash radio radio-hammerhead-m8974a-1.0.25.0.23.img
fastboot reboot-bootloader (reiniciamos tras cada flasheo)

*Y finalmente, procedemos a actualizar la partición del sistema del archivo* ***.zip****, que contiene el código de* ***AOSP*** *compilado con los ajustes y archivos propietarios del dispositivo. Se instala con el comando* ***update****.*

fastboot -w update image-hammerhead-kot49h.zip

*(El argumento -w es opcional, e indica que se haga una instalación limpia borrando todos los datos del usuario. Para una simple actualización no debería haber problema por mantener los datos, así que en este caso se podría omitir.)*

## Gestión de la variabilidad

**Ejercicio 1**

Desde el punto de vista de la personalización el núcleo de Linux se ha explicado como algo invariable para distintos productos, y que cambia solo a lo largo de las versiones que son lanzadas. Si bien esto a nivel de construcción de ROM personalizada no tiene por qué ser así. Explique la personalización del núcleo de Linux para construcciones propias.

***Solución:***

*Android es el producto resultante de construir **AOSP** para un dispositivo y mercado concreto. Cada fabricante, dependiendo del dispositivo y otros criterios, realiza la construcción integrando los paquetes que necesite por ejemplo si se construye para una tablet la construcción puede no integrar la parte de aplicaciones del sistema porque quien construye tiene las suyas propias, y no integrará las aplicaciones relativas a la telefonía (aplicación de telefono para llamar, aplicacion para enviar SMS, etc.).*

*Esto respecto a la capa personalizable, pero **AOSP** como se ha estudiado en al gestión de la variabilidad se compone de un núcleo invariable formado por el núcleo Linux correspondiente a la versión de Android que se construye, librerías, máquina virtual, etc. Pero al realizar la construcción se puede también personalizar ese núcleo, en este caso modificar el núcleo de Linux.*

*Esto es útil cuando la construcción es para un dispositivo que ya no recibe soporte oficial y las últimas versiones del núcleo no funcionan bien en dicho dispositivo y queremos construir una versión de Android superior a la soportada oficialmente.*

*Por tanto, ocurre que el núcleo en cierto modo es personalizable y es un detalle que no suele ser muy común dentro de la variabilidad encontrarse con supuestos donde se modifica el núcleo sobre el que se realiza la construcción.*

## Gestión de incidencias y depuración

**Ejercicio 1**

Simule la realización de un reporte, siguiendo correctamente las instrucciones de cómo debe hacerse un buen reporte, sobre un error que ha encontrado en Android.

***Solución:***

*Ejemplo de reporte en el apartado **Buen Ejemplo** de la Sección **12. Gestión de Incidencias y Depuración***

## Integración y despliegue continuos

**Ejercicio 1**

Indique los pasos a seguir para realizar la instalación de la herramienta Jenkins en Ubuntu LTS 12.04.

***Solución:***

*Para instalar Jenkins en Ubuntu se pueden seguir una de las 2 opciones siguientes:*

***Instalar Jenkins desde los repositorios de software de ubuntu:*** *basta con ejecutar en la consola el comando*

```
apt-get install jenkins
```

***Desplegar Jenkins en el servidor Tomcat****: para desplegar Jenkins en tomcat es necesario seguir los siguientes pasos:*

*- Crear una carpeta para jenkins con los permisos apropiados: para ello utilizamos los 2 comandos siguientes:*

```
sudo mkdir /usr/share/tomcat7/.jenkins
sudo chown tomcat7:nogroup /usr/share/tomcat7/.jenkins
```

*- Crear un usuario para Jenkins en /var/lib/tomcat7/conf/tomcat-users.xml: para ello añadimos las siguientes líneas en el fichero:*

```
<role rolename="admin"/>
<user username="jenkins-admin" password="secret" roles="admin"/>
```

*- Reiniciamos el servicio de tomcat: para ello utilizamos el comando sudo service tomcat7 restart*

*- Descargamos el fichero war y lo desplegamos en tomcat: para ello podemos usar el gestor web que se encuentra en la url* http://localhost:8080/manager/html

## Mapa de herramientas

**Ejercicio 1**

Describa para las siguientes herramientas cuál sería su uso y dentro de que aspecto (gestión del código fuente, gestión de la construcción, etc.) se enmarca: Git, Gerrit y Jenkins.

***Solución:***

***Git****: es un tipo de repositorio, que en AOSP se utiliza a través de Repo. Se utiliza para llevar a cabo a gestión del código fuente principalmente.*

***Gerrit****: "Gerrit is a free, web-based team software code review tool. Software developers in a team can review each other's modifications on their source code using a Web browser and approve or reject those changes. It integrates closely with Git, a distributed version control system." (Wikipedia inglesa)*

***Jenkins****: es una herramienta que ayuda a realizar la integración continua y la construcción a partir del repositorio. Se enmarcaría por tanto en integración continua y gestión de la construcción.*

# 18. Sugerencias

En general, la documentación oficial se encuentra desactualizada y es muy escueta, se proporciona poca información respecto a la Gestión de la Configuración, y muchas actividades no se contemplan siquiera. La responsabilidad de la documentación recae demasiado en Google, creemos conveniente que intentaran un enfoque más abierto y colaborativo como el de las wikis, como ya utilizan con éxito otros proyectos de software libre.

## Gestión del código fuente

No nos ha gustado ver como un proyecto libre se desarrolla de forma tan cerrada. Aunque el código esté a disposición pública, y aunque cualquiera pueda contribuir mediante mejoras y correcciones de errores, no se aceptan cambios significativos o nuevas funciones, dado que estas se desarrollan de forma privada y es Google quien tiene el control exclusivo. Entendemos que es el enfoque que han decidido dar a su estrategia de negocio con la intención de sacar lo mejor de Android, pero nos gustaría ver un desarrollo más abierto al público.

## Gestión de la construcción

El procedimiento básico de construcción está bien explicado en la documentación oficial, pero no pasa de ahí. Para un funcionamiento más avanzado hemos tenido que buscar información en otros medios que expliquen en mayor detalle las posibilidades de construcción. Echamos en falta esta explicación detallada de forma oficial, pues facilitaría enormemente la tarea de investigación.

Además, hemos observado que la comunidad ha mejorado las herramientas oficiales o creado otras nuevas. Por ejemplo, CyanogenMod utiliza programas adicionales para facilitar la configuración de la construcción y una versión mejorada de make paralelizable. Todas estas herramientas deberían ser debidamente probadas e incluidas de forma oficial en AOSP.

## Gestión de la variabilidad

La variabilidad de AOSP es un punto muy importante, pues el proyecto en sí no constituye un producto final. Es responsabilidad de los desarrolladores del producto elegir qué partes incluir y qué partes excluir, pero la documentación al respecto es escasa o casi inexistente. De manera oficial no se explica cómo preparar AOSP para funcionar en determinados dispositivos, más allá de las configuraciones que incluyen voluntariamente unos pocos fabricantes. Echamos en falta una explicación más detallada sobre el procedimiento a seguir para obtener un producto final teniendo en cuenta la variabilidad.

## Gestión de incidencias y depuración

Encontramos la gestión de incidencias un poco apartada del resto de la gestión del proyecto. Sería conveniente permitir referenciar el seguimiento de fallos con los parches aplicados al código para tener una traza clara de a qué fallos corresponde qué parche, ya que funcionan de manera totalmente independiente.

Además, en el sistema de incidencias se incluyen también las sugerencias y propuestas de nuevas funcionalidades, por lo que el resultado es caótico. Veríamos mejor separar ambas cosas, de manera que las sugerencias y comentarios de usuarios sin conocimientos técnicos se gestionen de manera diferente a las incidencias técnicas reportadas por usuarios con conocimientos avanzados.

Por otro lado, encontramos que este apartado de la gestión de la configuración tiene demasiado poco peso en AOSP. Vemos casos de reportes de errores relativamente sencillos de corregir, algunos con la propia corrección en el reporte, que tardan meses en ser siquiera aceptados y años en ser corregidos. Esta dejadez da la sensación de que las incidencias son ignoradas, lo que lleva a que muchos usuarios no se molesten en reportarlas.

## Pruebas

Las pruebas son una parte crucial de cualquier proyecto software, y en AOSP encontramos muchas pruebas de diverso tipo. El Compatibility Test Suite ayuda mucho en este aspecto, gracias a su documentación extensa y detallada. Pero las pruebas individuales de cada subproyecto de AOSP están poco documentadas. Pensamos que podrían mejorar en este sentido, documentando al menos una metodología de prueba estándar según el tipo de proyecto (aplicación, librería del sistema, proyecto externo…).

## Integración y despliegue continuos

La documentación oficial sobre integración y despliegue continuos es absolutamente inexistente. Entendemos que en AOSP no se contemple el uso de IC de manera pública por la generalidad del proyecto, pero nos parece frustrante que no se den indicaciones para los desarrolladores que deseen realizarla por su propia cuenta. Especialmente ya que AOSP hace uso de varias herramientas especiales interconectadas entre ellas como Gerrit y Repo, de las que no se dispone de información suficiente.

## Gestión de entregables

Destacamos que no haya unos plazos definidos para la liberación de los entregables, sino que estos se liberan a discreción de Google y a veces por sorpresa. Es una forma algo opaca y distanciada de la filosofía de código abierto.

# 19. Conclusión

En esta memoria se presenta el análisis de AOSP desde el punto de vista de la Gestión de la configuración, enmarcado dentro de la asignatura de Evolución y Gestión de la Configuración. Por tanto, hemos estudiados los aspectos vistos en la asignatura, como gestión del código fuente, gestión de la construcción, etc., del proyecto elegido.

AOSP como proyecto abierto y que cuenta con una comunidad, así como de la complejidad del proyecto, cuenta con mucha documentación y aspectos muy cuidados en cuanto a gestión de la configuración los cuales hemos analizado a lo largo de este documento. Por ejemplo, gestión del código fuente, gestión de la construcción, gestión de la variabilidad son aspectos que por las características del proyecto y del producto resultante del mismo tienen mucho potencial.

Pero dado que AOSP recae en su mayoría sobre la empresa Google tiene unos aspectos opacos que nos han dificultado la tarea de análisis de los mismos. Dichos motivos se presentan en el apartado de sugerencia y también en los apartados de esta memoria. Por ejemplo, la integración continua, el despliegue continuo, etc. No obstante, esto nos ha facilitado la tarea a la hora de realizar las sugerencias exigidas en el desarrollo de este trabajo.

# 20. Bibliografía


### Página principal del proyecto AOSP:

http://source.android.com/

### Gerrit del proyecto:

https://android-review.googlesource.com/

### Página del proyecto Gerrit:

https://code.google.com/p/gerrit/

### Documentación de Gerrit:

http://gerrit-training.scmforge.com/git-gerrit-workshop.html#gerrit-intro

### Integración contínua con Gerrit, Git y Jenkins:

https://www.open.collab.net/media/pdfs/Mobile-Dev-Git-Android-Jenkins.pdf

### Wiki de Cyanogenmod:

http://wiki.cyanogenmod.org/

### Cómo compilar AOSP para Nexus 4:

http://nosemaj.org/howto-build-android-nexus-4

### Cómo compilar partes específicas de una ROM:

http://xda-university.com/as-a-developer/downloadcompile-specific-rom-parts

### Manual de Compatibility Test Suite:

http://source.android.com/compatibility/android-cts-manual.pdf