# Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

**Maestría en Diseño Electrónico**


# PLATAFORMA DE INFO-ENTRETENIMIENTO BASADO EN ANDROID AUTOMOTIVE (PIE 9)

---

**TRABAJO RECEPCIONAL** para obtener el **GRADO** de
**MAESTRO EN DISEÑO ELECTRÓNICO**

Presentan:
**JOSE ALEJANDRO GUTIERREZ TIRADO**
**JUAN EDUARDO LOPEZ FLORES**
**ISIDRO SANTOS LECHUGA**

Director **VICTOR HUGO HUIDOBRO GARCIA**

Tlaquepaque, Jalisco. 5 de junio de 2023.

# AGRADECIMIENTOS

Los autores de este trabajo
agradecen sinceramente a:

- Alejandro: Agradezco a mis padres por sus esfuerzos y sacrificios para enseñarme un camino de educación y autosuperación.
- Eduardo: Agradezco la oportunidad de aportar en el desarrollo de este tema de trabajo recepcional a mis compañeros y familiares que me impulsaron a cerrar este ciclo de mi vida académica.
- Isidro: Agradezco a mis padres por siempre apoyarme y motivarme en los diferentes ámbitos de mi vida. Ellos me han enseñado el valor del sacrificio y el trabajo duro, lo que me ha permitido ser quien soy el día de hoy. Muchas gracias por todos sus sacrificios.
- A nuestro asesor Victor Hugo Huidobro Garcia, por el apoyo técnico y en la revisión y redacción de este documento.
- Al ITESO por proporcionar las herramientas, laboratorios y facilidades para el desarrollo de este Proyecto de grado.

# Resumen

*Este documento evalúa la mejora en los tiempos de ejecución de un proyecto automotriz al introducir el uso de una plataforma de software libre, en este caso Automotive Android Open Source Project (AAOSP). Tomamos como referencia el tiempo invertido durante las primeras fases del desarrollo de un producto propiedad de la compañía Tier 1, basado en software propietario, y se compara con la creación de un sistema de Info-entretenimiento con software libre.*

*Desde el proceso de diseño hasta los resultados de las pruebas, han sido documentados y se pretende ilustrar los principales retos y las decisiones de diseño más importantes tomadas en el desarrollo del prototipo. El documento, tiene capítulos especializados donde se aborda con detalle el diseño, implementación y pruebas, así como una sección dedicada a anexos en la que se podrán encontrar las partes más importantes del proyecto que han sido desarrolladas. Debido a la relación de este proyecto con la compañía Tier 1. y la confidencialidad que se ha acordado, algunos puntos importantes no serán mencionados, estos puntos fueron acordados después de una revisión interna en la compañía Tier 1. Cabe mencionar que Android Automotive OS es libre, así que cualquier persona puede acceder a su documentación a través de los servidores de Google. En este documento se mencionan las referencias de la información usada durante el desarrollo del prototipo.*

# Contenido

# Introducción

Un sistema de Info-entretenimiento es una interfaz que por medio de software integra diferentes funcionalidades dentro de un vehículo. Este tipo de sistemas se encargan primordialmente de las comunicaciones, navegación y funciones de entretenimiento, como pueden ser la radio o la reproducción de medios digitales.

Durante los últimos años, estos sistemas han evolucionado conforme la tecnología ha avanzado. Esta necesidad por tener vanguardia tecnológica (en parte solicitada por los usuarios de los vehículos a las OEMs) y la gran cantidad de funcionalidades, han vuelto más complejos aún a los sistemas de info-entretenimiento, dando como resultado un incremento en el tiempo de desarrollo y muchas veces, la necesidad de más recursos económicos. Debido a esto, el sector automotriz ha incorporado metodologías de trabajo basadas en resultados incrementales, las cuales ganaron popularidad al haber sido usadas en el desarrollo de otras áreas en la industria del software.

El proyecto que se presenta en este documento de trabajo recepcional es un sistema de info-entretenimiento basado en Android Automotive OS. El sistema cuenta con dos unidades principales: la primera es responsable por los servicios de HMI, así como también aplicaciones; y la segunda es responsable de la red de comunicaciones en el vehículo y diagnósticos. Para el caso del prototipo, la segunda unidad será responsable de manejar la reproducción de audio y manejo de señales de radio AM/FM. La idea del proyecto fue generada dentro de la compañía Tier 1 y éste ha sido desarrollado en colaboración con ITESO. Debido a los criterios de confidencialidad, algunas secciones en este documento serán omitidas conforme los criterios de la compañía Tier 1.

En el capítulo 4 Diseño y arquitectura del prototipo, en este documento, se describen los requisitos definidos, así como también se presentan diferentes diagramas que muestran tanto la arquitectura a nivel sistema, como a nivel software. Dicho capítulo está enfocado al desarrollo se ha documentado en forma de pasos que reflejan las recomendaciones de la documentación de Google en una manera sencilla, así como también las diferentes aportaciones durante el desarrollo

de este proyecto. Esta parte incluye la elaboración de una aplicación de media que tiene como objetivo ser la vista principal del radio.

El capítulo 7 Verificación y Validación está enfocado a las pruebas se presenta el uso de la herramienta "Compatibility Test Suite" desarrollada por Google, el cual corre en un PC y ejecuta diferentes casos de pruebas predefinidos para el hardware. El conjunto de pruebas que contiene CTS han sido desarrolladas para ser integradas tanto de manera manual como en sistemas de *pruebas de integración continua* de software, y para el caso de este proyecto, esta ha sido usada para validar el prototipo desarrollado en un estilo "*Incremental*". Dichas pruebas han sido documentadas para tener una adhesión a un proceso donde se pueda ejecutar CTS de forma estructurada. De la misma manera, en dicho capítulo también se describe el uso de "CTS verifier", un suplemento de CTS que provee pruebas para APIs y funciones que no pueden ser probadas en un dispositivo sin entradas manuales. Este ha sido usado para simular algunos entornos propios de un vehículo.

# 1. Generalidades

La intención de este capítulo es brindar un contexto más amplio del planteamiento del problema y la definición de objetivos en este proyecto. Se pretende que el lector pueda comprender la estructura de la idea desarrollada, así como también el problema que se pretende abordar.

## 1.1. Planteamiento del problema

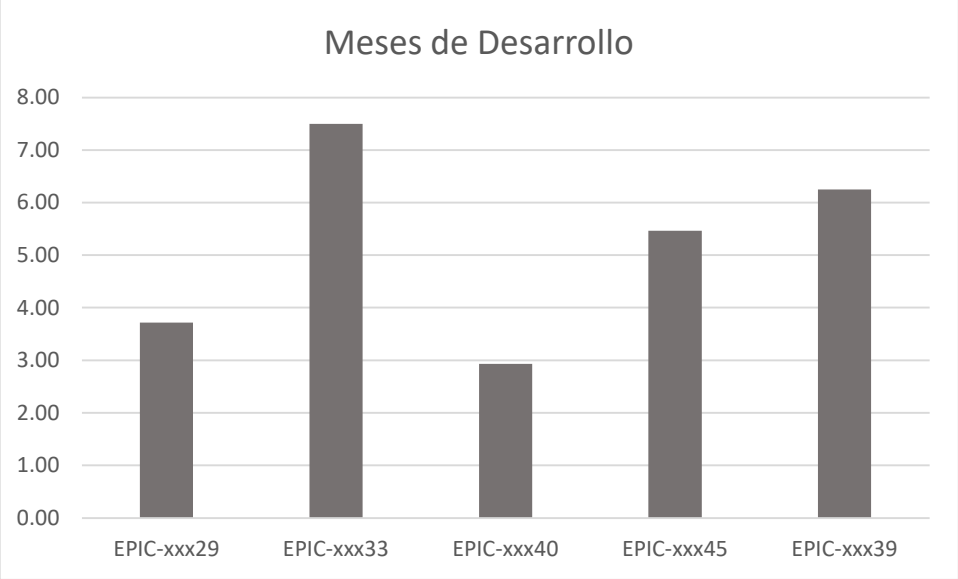
Las necesidades de acceso a servicios de conectividad móvil, localización y comunicación, han ido en aumento desde hace ya varios años al incrementarse la sofisticación y complejidad de los usuarios o sistemas que los emplean. La evolución de los teléfonos celulares de hoy en día son un buen ejemplo de lo antes mencionado. Si se comparan los primeros modelos de 1983 con los actuales en 2020, estos han pasado de ser sólo un dispositivo para realizar y recibir llamadas, a uno mucho más complejo y sofisticado que provee servicios que incluso involucran el uso de aprendizaje profundo.

Actualmente, el panorama global de las tecnologías 5G y su inminente introducción en las redes de comunicaciones y de datos, acentúan aún más esta tendencia. La escalada abrupta en la cantidad y velocidad de acceso a los datos proyecta un incremento explosivo y sin antecedentes en las aplicaciones para las tecnologías móviles.

Los beneficios que estos usos modernos ofrecen han provocado una tendencia a buscar las mismas funcionalidades en otros dispositivos, como es el caso de los sistemas de Info-entretenimiento en los vehículos. Sin embargo, el proceso de integrar dichas funcionalidades en un ambiente automotriz se ha vuelto complejo al grado que compromete los tiempos tradicionales de desarrollo de sistemas en esta industria.

La Gráfica 1 muestra el tiempo invertido en las diferentes tareas de desarrollo del tipo “Epic” (ver sección 2.1) ejecutadas para levantar (BringUp) el Hardware Abstraction Layer en un

proyecto de Network Access Device (NAD) automotriz. Las tecnologías involucradas son del tipo móviles (3G, 4G) y el software fue migrado y adaptado desde un proyecto anterior.




*Gráfica 1.Tiempo empleado en el BringUp el HAL en un NAD automotriz*

Analizando las tareas relacionadas, que son mostradas en la Tabla 1 encontramos que la mayoría de ellas fueron ejecutadas en paralelo, de acuerdo con su disciplina. Sin embargo, el desarrollo del framework de aplicación y la integración en el sistema, destacan entre las de mayor tiempo invertido.

| TASK ID | Development Epic/Milestone | Meses de Desarrollo |
|---|---|---|
| Epic-xxx29 | HAL Infrastructure Standalone bringup | 3.71 |
| Epic -xxx33 | HAL application Framework Standalone | 7.50 |
| Epic -xxx40 | HAL Security, Media, Location | 2.93 |
| Epic -xxx45 | HAL Integration | 5.46 |
| Epic -xxx39 | HAL systems | 6.25 |

*Tabla 1. Descripción de las tareas de BringUp de la NAD automotriz.*

El reto de la integración de estas innovaciones en los sistemas automotrices radica principalmente en el retrabajo de adaptación y su logro dentro de los tiempos de proyecto, sin

alejarse del cumplimiento de los elevados requisitos de seguridad que es necesario asegurar a los usuarios en este sector.

## 1.2. Formulación del problema

Basado en el supuesto de usar una plataforma ya disponible en el mercado, es razonable que el analizar el tiempo de desarrollo del sistema completo no brinda información relevante para el objeto de estudio. Debido a esto, el problema que se pretende abordar es: ¿De qué manera se puede acortar tiempo de desarrollo e integración de "features" claves en sistemas de Info-entretenimiento, sin afectar la calidad ni la innovación en productos automotrices?

### 1.2.1 Formulación de los problemas específicos

- ¿Es posible integrar servicios y aplicaciones existentes sin afectar las normas automotrices?
- ¿Qué estrategias se deberían de emplear para solucionar los desafíos recurrentes de calidad y desempeño en el desarrollo de dispositivos electrónicos de Info-entretenimiento, tales como: tiempos de arranque, administración de memoria, validación y verificación?

## 1.3. Objetivos

### 1.3.1 Objetivo general

Desarrollar el prototipo de un sistema de Info-entretenimiento e integrar un servicio de audio, que sirvan de punto de partida para comparar los tiempos de desarrollo en proyectos que usaron plataformas propietarias.

### 1.3.2 Objetivos específicos

- Usar Android Automotive OS en el prototipo para asegurar que las restricciones automotrices no se vean comprometidas.
- Ejecutar un set de pruebas de validación y verificación para aplicación.
- Analizar y evaluar el manejo de memoria en ART (Android Run time) para aplicaciones en sistemas automotrices tomando como base una aplicación de audio.
- Evaluar las estrategias disponibles de arranque del sistema, considerando las necesidades de la industria automotriz.

## 1.4. Justificación

Con la llegada de los dispositivos móviles y sistemas operativos con alcances de hardware tan altos como Android, los retos relacionados a las expectativas y percepciones del usuario final aumentan en alto grado la complejidad. Algunos estudios han demostrado que los usuarios finales esperan que las aplicaciones carguen y respondan rápido a las interacciones del usuario, sean fluidas y placenteras para la vista, todo esto a medida que los ingresos económicos (ventas) aumentan. Usualmente las aplicaciones en Android que están construidas sin cuidar el desempeño son desinstaladas en el mismo momento que fallan.

En el caso de la telefonía móvil, en 2007 esta industria unió sus fuerzas con Google para formar parte del "Open Handset Alliance (OHA)". La misión fue establecer Android como un sistema de código abierto. Lo cual significa que cualquiera podría tener acceso, descargar y modificar el código fuente de Android de manera libre para ayudar a desarrollar aplicaciones, dispositivos móviles, o incluso un sistema operativo competitivo.

Los Beneficios fueron inmediatos. Las compañías pudieron adaptar Android para construir experiencias únicas para sus clientes. A los desarrolladores de aplicaciones se le dieron accesos a una audiencia global. Y tal vez lo más impactante, los fabricantes de dispositivos pudieron instalar Android en otros dispositivos de manera gratuita, sin licencias o desarrollar un sistema operativo propio. Esto les permitió a los fabricantes recortar los gastos de desarrollo, lo cual ayudo a bajar el precio de los teléfonos inteligentes alrededor del mundo. El precio promedio de los teléfonos inteligentes cayó un 25% a nivel mundial. Como la tendencia continúa, nos acercamos a un mundo con tecnología móvil accesible que puede brindar la información global a cualquiera.

Los sistemas de Info- en los autos comparten pequeñas similitudes con los radios de los primeros automóviles, estos eran muy simples, nada más que simples radios analógicos. Sin embargo, actualmente no sólo deben reproducir música y dar notificaciones al conductor, así como también a sus pasajeros, además de eso reproducen música desde dispositivos de almacenamiento como tarjetas SD y USB; y como radios, pueden reproducir múltiples bandas, cubriendo tanto transmisiones analógicas como digitales. Otra característica importante es la conexión con dispositivos móviles vía Bluetooth, Wi-Fi o de manera inalámbrica (como 5G). También los sistemas de navegación se han vuelto un estándar en los autos modernos, usando GPS y datos de internet en tiempo real para buscar las rutas más rápidas para el conductor.

La tecnología móvil a progresado tan rápido en los años pasados que las personas demandan las mismas capacidades de entretenimiento, información, y comunicación cuando están en la carretera, dentro de su vehículo. Debido a esto la industria automotriz tiene retos importantes que cumplir. Brett Berk, un escritor freelancer que ha colaborado con medios tales como Autoblog, Automobile, Autoweek y Bloomberg, escribió para wired.com acerca de este dilema en su artículo

"The unending Struggle to Make Your Car Fell Like Your Phone" [1]. Berk habla de cómo la durabilidad y los requisitos relacionados a seguridad del pasajero vuelven de algo simple a algo complejo el tener las últimas tecnologías móviles dentro del vehículo. Menciona que puesto que un proyecto automotriz puede tomar de 3-5 años en desarrollarse y los clientes esperan tener sus autos en promedio 11 años, lo que era nuevo e innovador cuando el fabricante de automóviles lo instalo en el automóvil, estará desactualizado mucho antes de que el motor deje de funcionar. También cita a Derek Jenkins, director de diseño de la empresa emergente de automóviles eléctricos de california Lucid Motors: *"La industria automotriz se enfoca en lo que hace bien, estilo automotriz tradicional, comodidad, seguridad y desempeño"*, *"Los sistemas de Info-entretenimiento han sido una carga"*.

Este trabajo recepcional está motivada por el interés de investigar los retos ya mencionados que enfrenta la industria automotriz y para ello se busca desarrollar un prototipo de plataforma de un sistema de Info-entretenimiento, el cual será basado en Android Automotive OS integrando una aplicación.

### 1.4.1 Ejecución del set pruebas de validación y verificación para un producto con Android Automotive

La mayoría del contenido y los trabajos de investigación en lo que respecta al desarrollo de un producto con base en un sistema embebido se enfocan en el desarrollo del código del procesador o procesadores del sistema, sin embargo, hay un conjunto de nuevas prácticas que, empujadas por la confiabilidad y prestigio del producto en el mercado de dicho producto, se ha abierto horizontes en una serie de actividades que se dirigen a que este sistema sea satisfactorio a un cliente final. Del punto en concreto que se habla es ingeniería de pruebas.

Si bien, el área de ingeniería de pruebas o "Testing" es identificada en ciertos sectores o niveles e industrias, esta no existe formalizada en sus diferentes vertientes, y, con el termino vertientes se puede ubicar al nivel de pruebas ejecutado, así como el tipo de aplicación a la cual el mercado está dirigido. Por ejemplo, no es lo mismo probar interfaces como APIs dentro de un sistema diseñado para probar las conexiones con otros sistemas dentro de un universo o sistema

de sistema mucho más grande; es decir, en un automóvil donde aproximadamente existen cerca de 100 sistemas (de acuerdo a la experiencia de mercado de una compañía Tier 1), existen pruebas de *verificación* iniciando desde el Software más unitario hasta una integración completa en una arquitectura hasta pruebas de validación en un vehículo donde se integraría el producto final.

Ubicando ahora el tema de ingeniería de pruebas en los conceptos técnicos del producto, se puede pasar a repasar algunos números que, además de los conceptos técnicos que se aborde, en el caso de estudio para ingeniería de pruebas para Android Automotive debería ser un aliciente el tema financiero de ingeniería de pruebas. Según un estudio externo de GMI [2] donde se analizan las tendencias de mercado, el tamaño de mercado para el 2022 fue de 45 mil millones de dólares estadounidenses. Con las tendencias tecnológicas de más de 5% anual, dentro de 10 años se puede estimar que este mercado va a crecer casi en un 100% a más de 90 mil millones de dólares.

Ahora regresando al tema de desarrollo de donde todo diseño parte, los desarrolladores de sistemas automotrices tanto de Info-entretenimiento como de otros dominios dentro del mismo automóvil han estado enfrentando retos más tenaces en la trama de información que manejan, ya que aspectos como conectividad, imagen y video incrementan sustancialmente la complejidad de requisitos y por ende de las soluciones de software.

A pesar de que las implementaciones han sido cada vez más complejas, el uso de conceptos como el reúso, plataforma y arquitectura modular llegan a facilitar tanto la parte de implementación de código, como la parte de verificación de sistemas y sub-sistemas. Lo anterior no ha pasado inadvertido a los creadores de Google Android Automotive, quienes han creado casos de prueba genéricos, así como secuenciadores de pruebas dentro de *Android Studio*. Estos también se pueden conocer como una colección de "documentos" que avalan el alineamiento del software en cuestión de arquitectura con capas bajas de Hardware, así como servicios al sistema operativo, el cual exige cumplir ciertos parámetros de estructura para la compatibilidad propuesta por el proveedor de dicho sistema operativo. Por ejemplo, en el "Android Compatibility Definition Document" o simplemente CDD, se desprenden conceptos para ser usados en secuenciadores de pruebas tipo "CTS".

A pesar de la compatibilidad de Android Automotive, éste aún adolece de ciertos elementos que han madurado previamente al lanzamiento de esta apuesta, la cual son los ciclos de desarrollo tanto de HW como de SW que existen en la industria automotriz. En esta ocasión nos referiremos al modelo de procesos que es un marco de referencia basado en un estándar IEC 15504, comúnmente llamado Automotive SPICE. Este modelo tiene varias categorías las cuales suelen enmarcarse en un ciclo en forma de "V", lo cual es una de las partes relevantes de la propuesta de V&V (verificación y validación) presentadas. Su adaptación al estilo de trabajo de la metodología SAFe ágil será abordada en este proyecto.

### 1.4.2 Manejo de memoria en Android OS y Aplicación de media

El desempeño del software es independiente de la ejecución de la tecnología de cómputo o disponibilidad de recursos adecuados de software. Algunos cuellos de botella en el desempeño de software pueden darse cuando accesos a recursos compartidos se vuelven contencioso y retrasan la ejecución funcional o extiende el tiempo para que se realice una función.

En una plataforma tan grande como Android, el estudiar el entorno y diseñar apropiadamente puede ayudar a evitar problemas de desempeño que actualmente se presentan. Puesto que dicho OS soporta miles de diferentes plataformas de hardware, es importante elegir los métodos adecuados para administrar los recursos.

Uno de los temas a desarrollar en este trabajo recepcional será el manejo de memoria para obtener el mejor desempeño de software posible, dicho tema se desarrollará junto con una aplicación de reproducción de medios para así demostrar los resultados obtenidos.

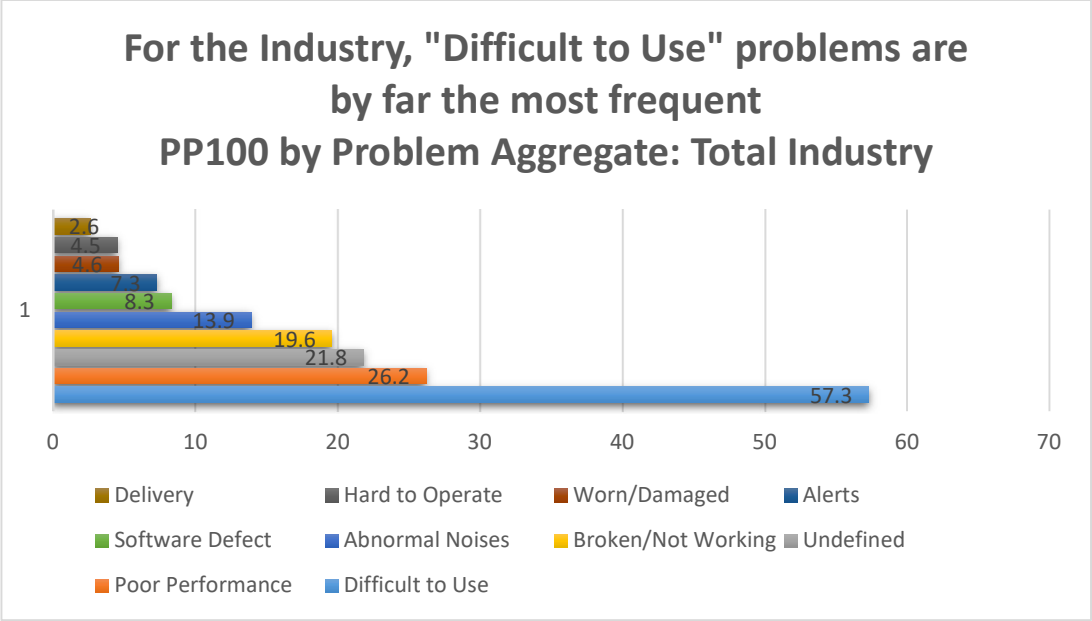
Como ejemplo de en lo que puede resultar el desempeño del software podríamos revisar algunos estudios en servicios web, los cuales proveen amplio contexto para aplicaciones móviles en Android o cualquier otro sistema operativo. Hay muchos estudios que demuestran que mejorar el desempeño de un sitio web incrementa el compromiso de los usuarios y así también las ventas.

Según Todd Hoff, en la página “High Scalability” [3], Amazon y Walmart [4] han reportado de manera independiente resultados similares. Dos de los mayores vendedores encontraron que sólo 100ms de retraso en sus páginas web causaron que sus ingresos bajaran por 1%.

El ejemplo anterior, podría no resultar ser totalmente aplicable a la industria de Info entretenimiento automotriz, pero, no es así, el incremento en malas reseñas de autos con sistemas de Info-entretenimiento es notable. En el pasado J.D. Power 2020 Initial Quality Study [5], se puede notar que los fabricantes de automóviles todavía tienen un largo camino por recorrer para mejorar los sistemas de Info-entretenimiento, que se vuelven más importantes y controlan más funciones que nunca.

Según J.D. Power, el Info-entretenimiento es la categoría más problemática de cualquier característica en los automóviles probada en las encuestas de calidad iniciales del 2020, y casi una cuarta parte de todos los problemas citados por los propietarios de vehículos nuevos se relacionan con el Info-entretenimiento. Por lo general, se relacionan con el reconocimiento de voz, la conectividad Bluetooth, la integración de Apple CarPlay y Android auto y los sistemas de navegación. Los datos se basan en las respuestas de casi 90000 compradores y arrendatarios de automóviles nuevos encuestados después de 90 días de propiedad.

La Gráfica 2 muestra los resultados de los problemas que dificultan el uso de los sistemas de Info-entretenimiento en los vehículos. Si bien, se puede notar que del 100% de los problemas 26% son relacionados al mal desempeño. Así como también, los problemas de software son el 8%, es en este tipo de retroalimentaciones donde podemos notar la importancia de mantener calidad y un buen desempeño en el software.




*Gráfica 2. Problemas más frecuentes para la industria, resultados basados en el estudio J.D. Power 2020 Initial Qualty Study*

Según el estudio el 62% de los usuarios que reportaron problemas tienen una aplicación del fabricante y el 6.5% de problemas son relacionados a la conexión de las aplicaciones. Lo anterior se muestra en la Imagen 1.

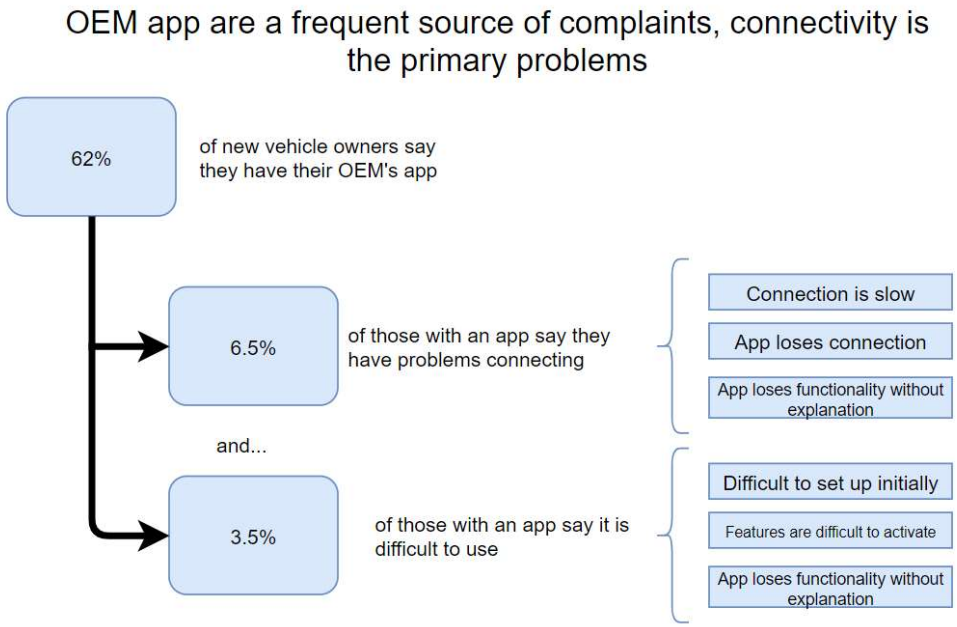



*Imagen 1. Quejas más frecuentes según el estudio J.D. Power 2020 Initial Quality Study.*

Usualmente los sistemas de Info-entretenimiento son lentos comparados con los celulares actuales, el hardware y software usado en ambos sistemas son muy diferentes puesto que usualmente un smartphone es diseñado para durar sólo unos años, mientras que se espera que todos los componentes de un automóvil duren al menos 10 años, y cuando se habla de vehículos hay otros temas a considerar como la distracción, seguridad del usuario y ciber-seguridad; incrementando los retos de desarrollo y volviendo más complejo el conservar los mismos estándares de calidad y desempeño.

### 1.4.3 Evaluación de las diferentes estrategias disponibles para el arranque del sistema.

Parte del gran reto de la industria automotriz con los sistemas críticos, radica en la diversidad de legislaciones y especificaciones que existen en las diferentes regiones del mundo. Una aplicación en un teléfono inteligente, que puede involucrar ya requerimientos altos y específicos para alguna región del mundo en cuanto a seguridad y manejo de la información, como es el caso de las aplicaciones bancarias, tiene sin embargo gran flexibilidad en cuanto a tiempos de respuesta e inicialización cuando se le compara con funcionalidades críticas, tal como una llamada de emergencia en un vehículo.

Una aplicación móvil que requiere un alto grado de ciber seguridad, puede sin embargo tomarse el tiempo necesario para iniciar o reiniciar, ya sea al momento de encender el teléfono o al activarse, siempre y cuando esto no afecte o degrade la experiencia del usuario. Las aplicaciones críticas, deben en cambio asegurar tiempos específicos no sólo de respuesta, sino también de disponibilidad en el sistema de acuerdo con regulaciones de seguridad que son también específicas dependiendo de la región del mundo en donde se pretendan validar.

En el caso de una llamada de emergencia, ésta presenta todos los retos en cuanto a inicialización mencionados anteriormente:

- Tiempos críticos de disponibilidad en el sistema
- Tiempos críticos de respuesta

- Necesidad de satisfacer regulaciones locales para dichos tiempos

Una iniciativa de la comunidad europea estableció para ello el programa llamado “eCall”, que volvió obligatorio que todos los autos nuevos integraran esta funcionalidad a partir de abril del 2018. Esto se dio a conocer en un sitio oficial de la Unión Europea con el artículo “eCall in all new cars from April 2018” [6]. Iniciativas similares existen también, por ejemplo, en Rusia con su “ERA-GLONASS” y en Canadá y los Estados Unidos con su “NG9-1-1”. Existen además iniciativas de servicios privados, como el caso de “OnStar” de General Motors.

Android Automotive OS presenta ya un modelo que permite configurar la estrategia de inicialización de las aplicaciones en el sistema y de los servicios de los cuales éstas dependen. Se pretende entonces, evaluar el uso y opciones de configurabilidad para adaptar dichas estrategias en el caso de una aplicación de llamada de emergencia y entender como beneficia en el desarrollo de un sistema automotriz. Un análisis de dichas estrategias y los requerimientos de alguno de los estándares disponibles, permitirá establecer su viabilidad e implementación final en el prototipo.

# 2. Metodología

Para el desarrollo de este proyecto se usó una metodología ágil llamada “Scaled Agile Framework” (SAFe) que es un conjunto de patrones de organización y flujo de trabajo que sirve para implementar practicas agiles a escala empresarial. El marco constituye un cumulo de conocimientos que incluye instrucciones estructuradas sobre las funciones y responsabilidades, la forma de planificar y gestionar el trabajo, y los valores que hay que defender.

SAFe promueve la coherencia, la colaboración y la gestión a través de un gran número de equipos ágiles y se formó alrededor de tres cuerpos de conocimientos principales: el desarrollo de software ágil, el desarrollo de productos “lean” y el pensamiento sistémico.

A medida que las empresas crecen en tamaño, SAFe proporciona una estrategia estructurada para escalar de forma ágil. Existen cuatro configuraciones en SAFe para adaptarse a varios niveles: Essential SAFe, Large Solution SAFe, Portfolio SAFe y Full SAFe. La Imagen 2 muestra el ciclo ágil.

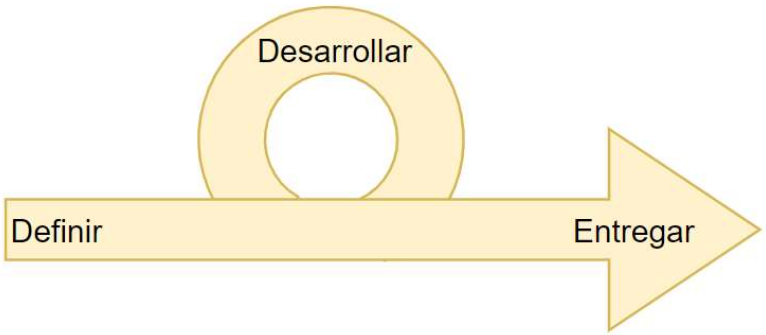



*Imagen 2. Ciclo ágil*

## 2.1. Implementación de SAFe Essential

Como ya se mencionó anteriormente, para el desarrollo de este proyecto y el presente documento se optó por utilizar SAFe Essential como metodología de desarrollo principal. Si bien, el alcance este proyecto y sus objetivos no son lo suficientemente amplios para hacer una implementación completa de SAFe Essential, es posible implementar ciertos elementos principales que ayudaran a definir, administrar y ejecutar las actividades necesarias para completar los objetivos propuestos.

Para la administración de objetivos y actividades se utilizarán los siguientes elementos de clasificación definidos en SAFe:

- **Epic:** Lo que se conoce como "Epic" dentro del enfoque de trabajo de ágil en el ambiente de la herramienta "JIRA", el cual es el grupo de que enmarca varias tareas con características específicas, conocidas también como "historias de usuario", y lo anterior tomando en cuenta las necesidades de los clientes o usuarios finales (recordar que no sólo en cliente en un implicado). Los "Epics" constituyen una práctica habitual entre los equipos que consideran el enfoque de trabajo tipo ágil como ciclo de desarrollo.
- **Story:** Una historia es un elemento que define o describe una característica de software de una manera muy general o incluso informal, para más información revisar la documentación de ágil [7].

Si bien, el "Epic" no es exclusivo de SAFe Essential, pero es parte de la herramienta de Jira usada para implementar SAFe, es debido a esto que se decidió usar este elemento de la metodología. Es prácticamente un contenedor de "Stories", y a su vez un "Story" puede contener tareas, acciones y reportes de problemas. La Imagen 3 muestra la relación entre un "Epic" llamado "Complete Thesis Document" y sus historias definidas.

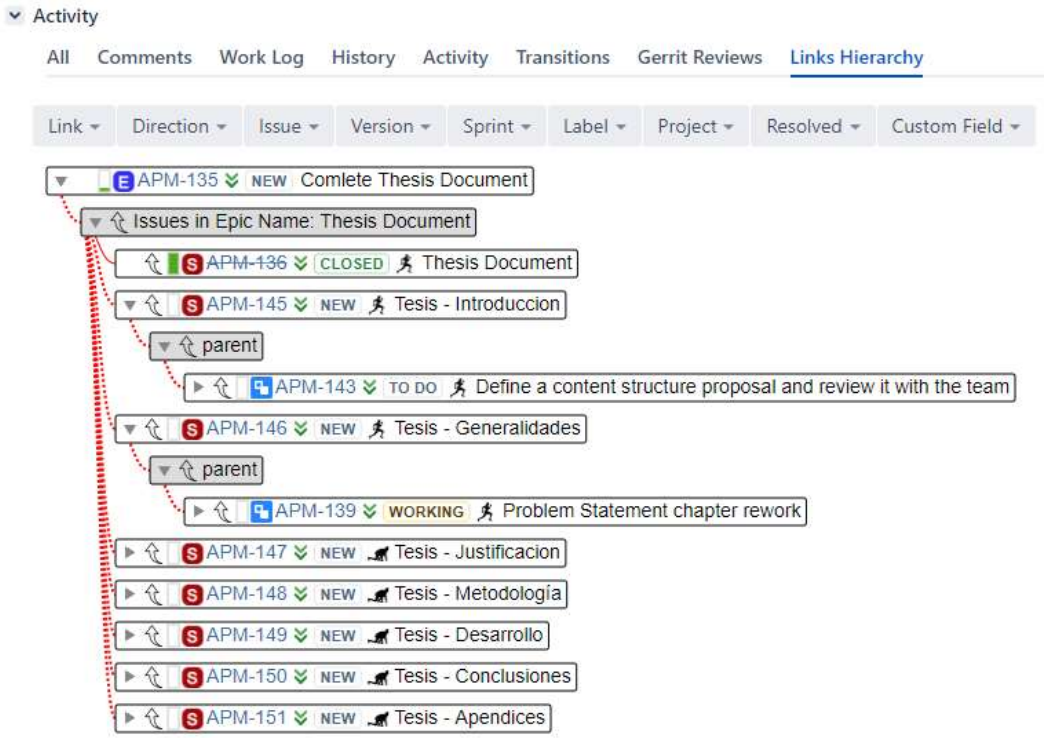



*Imagen 3. Muestra la relacion entre los elementos planeados dentro de Jira.*

Dentro de Jira, cada elemento creado contiene un identificador único y las relaciones entre Epics, Stories, Tasks y Actions pueden o no definirse sin afectar el uso de la herramienta, aunque ayuda significativamente en la trazabilidad de las actividades, así como también a la visualización del avance a alto nivel. Dentro de Jira hay diferentes atributos que pueden ser definidos por cada elemento, la Imagen 3 muestra un ejemplo de cómo se visualiza un Epic dentro de Jira.

Con respecto a la planeación y asignación de tareas, Jira permite crear “Sprints” el cual es un periodo de tiempo fijado en un ciclo de desarrollo continuo durante el cual, los equipos completan tareas procedentes del “Backlog” del proyecto. Normalmente, al final del sprint un equipo habrá compilado e implementado un incremento del proyecto. Jira convierte el backlog en el centro de la reunión de planificación del sprint, de forma que se pueden estimar historias, ajustar el alcance del sprint, comprobar la velocidad del equipo y cambiar las prioridades de las incidencias en tiempo real. Para el desarrollo de este proyecto se definieron 6 sprints en base al tiempo de entrega, la Imagen 4 muestra la planeación definida para este proyecto.

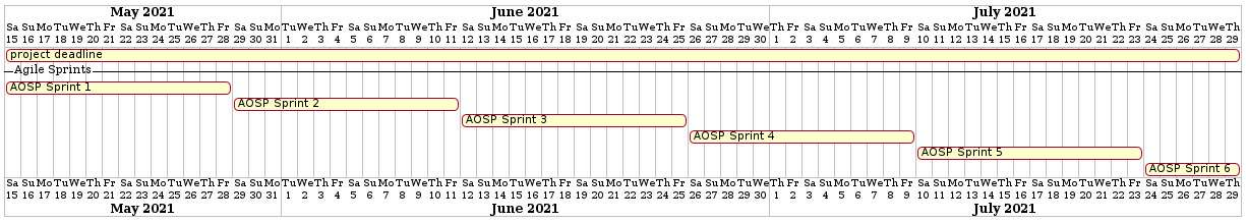



*Imagen 4. Distribución de sprints dentro del tiempo de desarrollo definido.*

Durante el periodo mostrado en la Imagen 4, se llevarán a cabo juntas de tipo "Daily Stand-Up" durante los martes y sábado. Este tipo de juntas permite al equipo comprender el estado del proyecto, escalar problemas y obtener ayuda de otros miembros del equipo. Durante este evento, cada miembro del equipo describe lo que hicieron para avanzar en los objetivos del sprint, en que van a trabajar para lograr los objetivos del sprint y comentar cualquier bloqueo que encuentren durante el sprint. Como parte de las excepciones, esta junta no se desarrollará de manera diaria como se define en SAFe y de igual manera el Scrum Máster será un miembro del equipo de desarrollo.

Durante la ejecución del Daily Stand-up el equipo crea y prueba historias con el objetivo de completar el mayor número posible de historias. Con la ayuda de Jira el equipo puede visualizan las historias y su progreso a lo largo del Sprint. Al hacerlo, a ser utilizan algunos pasos de desarrollo como columnas moviendo historias de izquierda a la derecha a lo largo del tiempo, como lo muestra la Imagen 5.

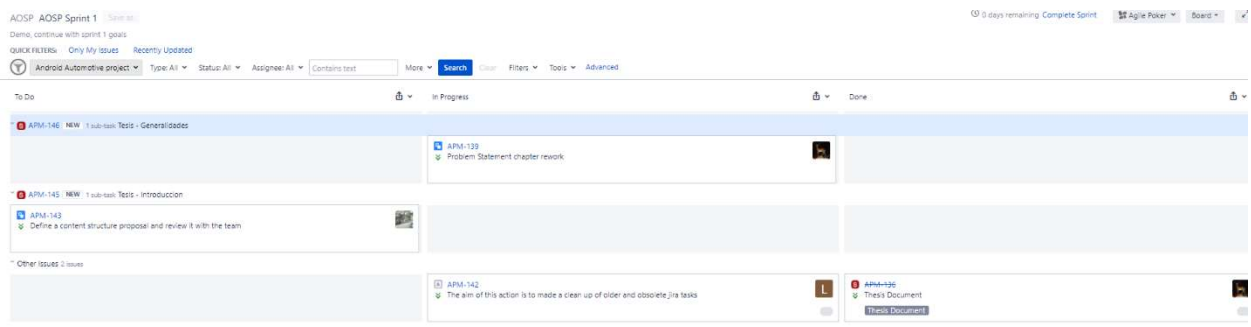



*Imagen 5. Tablero de control de tareas en Jira.*

# 3. Alcance y limitaciones del proyecto

## 3.1. Alcances

- El proyecto busca determinar la factibilidad de la instalación y desarrollo de gestión de recursos de un sistema embebido.
- Se analizarán las ventajas de instalar un sistema operativo como Android Automotive y se revisará el manejo de ciertos recursos como la RAM o el sub-sistema de arranque como ejemplos.
- Se llevará a cabo un desarrollo planificado valiéndose de recursos con estrategias y metodologías ágiles.
- Se asegurará que las soluciones presentadas en el proyecto cumplan con los requisitos desarrollados por el propietario del sistema operativo.

## 3.2. Limitaciones

- La implementación sólo estará enfocada en la manera de portar e iniciar el sistema operativo Android Automotive y usar sólo una aplicación.
- La implementación contemplará el uso de conceptos de arquitectura lo más genéricos posible para demostrar adicionales aplicaciones en un futuro.

# 4. Diseño y arquitectura del prototipo

## 4.1. Selección de la plataforma

Parte de los objetivos es establecer la viabilidad de uso no sólo de Android, sino también de los ambientes de desarrollo listos para su uso como alternativa a opciones propietarias. Por esto se hizo una selección inicial del Hardware y Software a usar como plataforma, contemplando que ésta podría cambiar.

### 4.1.1 Selección inicial: S820Am v2 Automotive Development Platform

Como opción inicial, se estableció el uso de Hardware y Software provistos por un tercero, basados en AAOSP. Se planteó el siguiente ambiente de desarrollo mostrado en la Tabla 2:

| | Selección | Descripción |
|---|---|---|
| Hardware | Qualcomm® Snapdragon™ S820Am v2 Automotive Development Platform | Plataforma de desarrollo basada en chipset de Qualcomm, fabricada por Intrinsyc, ahora Lantronix, Inc. [7] , específicamente para Qualcomm [8]. |
| Software | MSM8996AU.LA.1.2 Linux Android Release 00003 for MSM8996AU | Software liberado por Qualcomm provista previamente al Tier 1 para evaluación de las necesidades en sus productos. |

*Tabla 2. Selección inicial del ambiente de Hardware y Software.*

La intención de esta primera selección (ver Imagen 6) era beneficiar el desarrollo del sistema con la relación de Qualcomm con la compañía Tier 1. Se buscaba obtener soporte tanto del fabricante de la tarjeta de desarrollo, como del proveedor de software. No obstante, encontramos las siguientes dificultades a lo largo de los primeros meses:

1. La tarjeta de desarrollo requería una configuración inicial que no estaba completamente documentada para el modelo específico adquirido.

2. La información no estaba libremente disponible, y fue necesario deducirla consultando y comparando otras guías de usuario.
3. El fabricante de la tarjeta de desarrollo requería un convenio especial para brindar soporte específico.
4. El ambiente de software no podía ser adquirido por todos los miembros del equipo, se requería acceso especial otorgado a la compañía Tier 1.
5. El ambiente de compilación requería adecuaciones específicas para el uso de clang, que no estaban disponibles por default en el ambiente de software.
6. El procedimiento de flasheo de las imágenes generadas requería herramientas específicas de Qualcomm, que también necesitaban permiso especial para su uso.


*Imagen 6. Qualcomm® Snapdragon™ S820Am v2 Automotive Development Platform*

Los procedimientos intentados y contratiempos encontrados se documentaron en un sitio privado que se muestra en la Imagen 7, interno en la compañía Tier 1 (AAOSP Home) debido a la

naturaleza del software empleado. Actualmente, la información de esta plataforma de desarrollo se encuentra disponible en el sitio de Lantronix [9], debido al proceso de adquisición de Intrinsyc en 2020 [10].

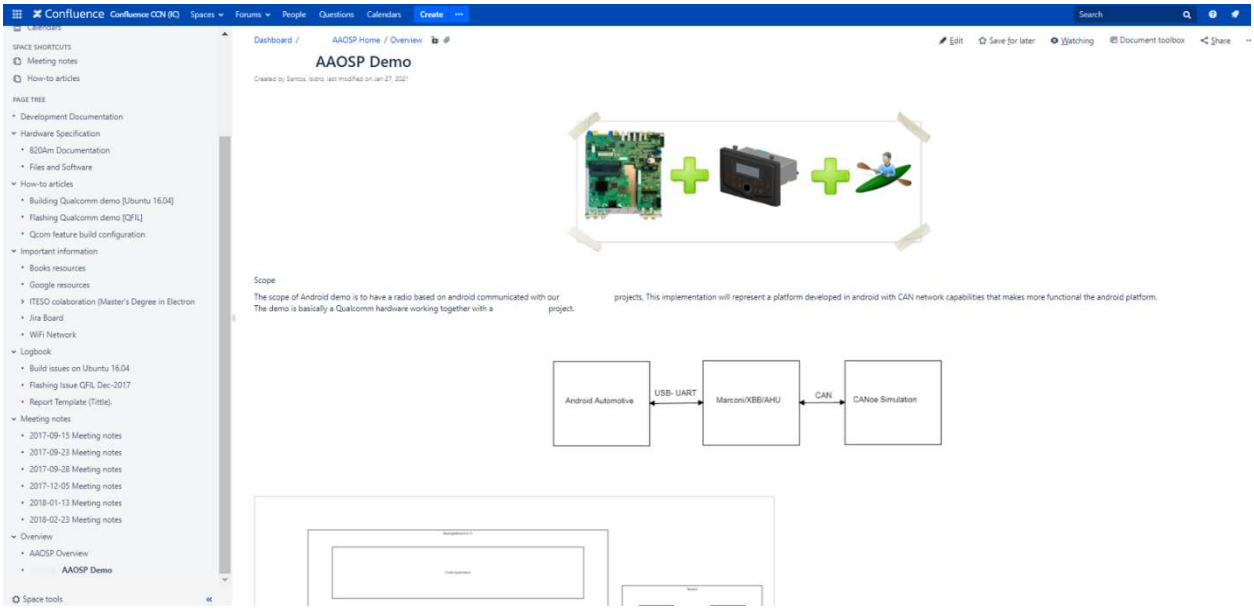



*Imagen 7. AAOSP Home Confluence space.*

### 4.1.2 Selección final: AM57xx BeagleBoard-X15 Sitara platform

Debido a las demoras generadas por las complicaciones encontradas, se optó por cambiar el ambiente de desarrollo a uno completamente libre. Se planteó una segunda opción mostrada en la Tabla 3:

| | **Selección** | **Descripción** |
|---|---|---|
| Hardware | AM57xx BeagleBoard-X15 Sitara platform | Plataforma de desarrollo basada en chipset de Texas Instruments [11], fabricada por BeagleBoard.org Foundation [12], para la comunidad Open Source. |
| Software | TI processor sdk-android am57xx, release 06.03.00.106 | Software liberado por Texas Instruments [11], orientado al desarrollo Open Source. |

*Tabla 3. Selección inicial del ambiente de Hardware y Software.*

Muy pronto, al cambiar el enfoque de ambiente de desarrollo a uno orientado a Open Source (Imagen 8), se logró avanzar significativamente debido a los siguientes puntos principales:

1. Información completa tanto en Hardware como en Software, ambos completamente disponibles.
2. Soporte necesario es patrocinado por el fabricante de Hardware, basado en casos de uso, y problemáticas reales de la comunidad de desarrollo Open Source. Todos los casos completamente documentados y accesibles.
3. Ambiente de Software disponible sin restricción a todos los miembros del equipo.
4. Herramientas de flasheo disponibles en la comunidad Open Source.

*Imagen 8. Tarjeta de desarrollo BeagleBoard*

## 4.2. Ambiente de desarrollo

### 4.2.1 Configuración del ambiente

Las instrucciones para la creación del ambiente de desarrollo, así como la compilación del proyecto de software y la carga de imágenes en el dispositivo, están documentadas en el reporte de aplicación "Enabling Android Automotive on Your TI Development Board" [13].

Los requerimientos y preparaciones básicas para el ambiente de trabajo están disponibles para Android en "Requirements | Android Open Source Project" [14] El ambiente es Linux, Ubuntu 16 / Ubuntu 18, de 64bits. 16G RAM, 500G HD.

Es necesario instalar en el ambiente de desarrollo todas las librerías requeridas. Esto se puede lograr ejecutando el siguiente comando:

```
$sudo apt-get install git-core gnupg flex bison build-essential zip curl
zlib1g-dev   gcc-multilib   g++-multilib   libc6-dev-i386   libncurses5-dev
lib32ncurses5-dev x11proto-core-dev libx11-dev lib32z1-dev libgl1-mesa-dev
libxml2-utils xsltproc unzip fontconfig
```

El proyecto para desarrollo en AM57X debe descargarse del sitio de Texas Instruments SDK AM57X [15]. La versión usada para este desarrollo se fijó en el reléase 06.03.00.106, que está disponible también en el repositorio del proyecto https://gitlab.com/iteso_mde_project.

Una vez instalado el proyecto, se puede trabajar en el software y posteriormente generar imágenes y cargarlas en el dispositivo.

### 4.2.2 Generación de imágenes

La generación de imágenes se realiza directamente siguiendo la sección 3.1 “Build Instructions” de “Enabling Android Automotive on Your TI Development Board” [13]. Los siguientes pasos describen brevemente el procedimiento:

1. En el área de trabajo, ejecutar el script de configuración

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

2. Ejecutar launch para seleccionar la variante de compilación

```
$launch
```

3. Compilar

```
$make -j
```

### 4.2.1 Carga de imágenes

Las instrucciones para la conexión física del dispositivo están disponibles en BeagleBoard-X15_Quick-Start-Guide [16]

Una vez conectado el dispositivo, se necesitarán:

- Una terminal al dispositivo conectado,
- Una consola abierta en el ambiente de desarrollo
- Una tarjeta de memoria

Los siguientes pasos explican el procedimiento para cargar las imágenes generadas en el dispositivo conectado:

1. Hacer una copia del folder de imágenes precompiladas para que pueda usarse como base.

```
$cp prebuilt-images emmc_files
```

2. Copiar todos los nuevos binarios de Linux a la carpeta con precompilados.

```
$cp -v aosp-9.0/out/host/linux-x86/bin/{simg2img,mkbootimg,fastboot,adb}
emmc_files/
```

3. Copiar todos los binarios nuevos específicos de TI a la carpeta con precompilados.

```
$cp -v aosp-9.0/device/ti/beagle_x15/fastboot.sh emmc_files/
```

4. Copiar todos los binarios nuevos, específicos de la tarjeta de desarrollo a la carpeta con precompilados.

```
$cp -v aosp-
9.0/out/target/product/beagle_x15/{boot.img,boot_fit.img,recovery.img,syst
em.img,userdata.img,vendor.img} emmc_files
```

5. Copiar todos los binarios nuevos del kernel a la carpeta con precompilados.

```
$cp -v linux-4.19.98+gitAUTOINC+8a8fd7ef1c-
g8a8fd7ef1c/arch/arm/boot/zImage emmc_files/zImage-beagle_x15.bin
```

6. Copiar todos los binarios nuevos del am57 a la carpeta con precompilados.

```
$cp -v linux-4.19.98+gitAUTOINC+8a8fd7ef1c-
g8a8fd7ef1c/arch/arm/boot/dts/am57*.dtb emmc_files/
```

7. Copiar todos los binarios nuevos del boot loader MLO a la carpeta con precompilados.

```
$cp -v u-boot-2019.01+gitAUTOINC+333c3e72d3-g333c3e72d3/MLO emmc_files/
```

8. Copiar todos los binarios nuevos de u-boot a la carpeta con precompilados.

```
$cp -v u-boot-2019.01+gitAUTOINC+333c3e72d3-g333c3e72d3/u-boot.img
emmc_files/
```

9. Crear la SD-card con el contenido default, ejecutando el script que se incluye con el proyecto de software.

```
$sudo bin/create-sdcard.sh
```

10. Abrir la terminal al dispositivo. Usar baudrate 115200 a 8bits.

11. Iniciar el dispositivo con la tarjeta instalada.

12. El sistema preguntara por opciones de arranque, hay que presionar cualquier tecla en la consola **antes que el contador expire**.

13. En la terminal, ejecute el siguiente comando para enviar el dispositivo a fastboot

```
$fastboot 0
```

14. De nuevo en la consola, cambie al folder con los archivos nuevos de la emmc

```
$cd board-support/emmc_files
```

15. Ejecute fastboot para cargar los nuevos binarios y reiniciar.

```
$sudo ./fastboot.sh; sudo ./fastboot reboot
```

16. Cuando la carga esté terminada, el sistema reiniciara con la nueva imagen cargada. Los logs se muestran en la Imagen 9. Así como también, en la pantalla se mostrarán las pantallas mostradas en las Imagen 10 e Imagen 11.

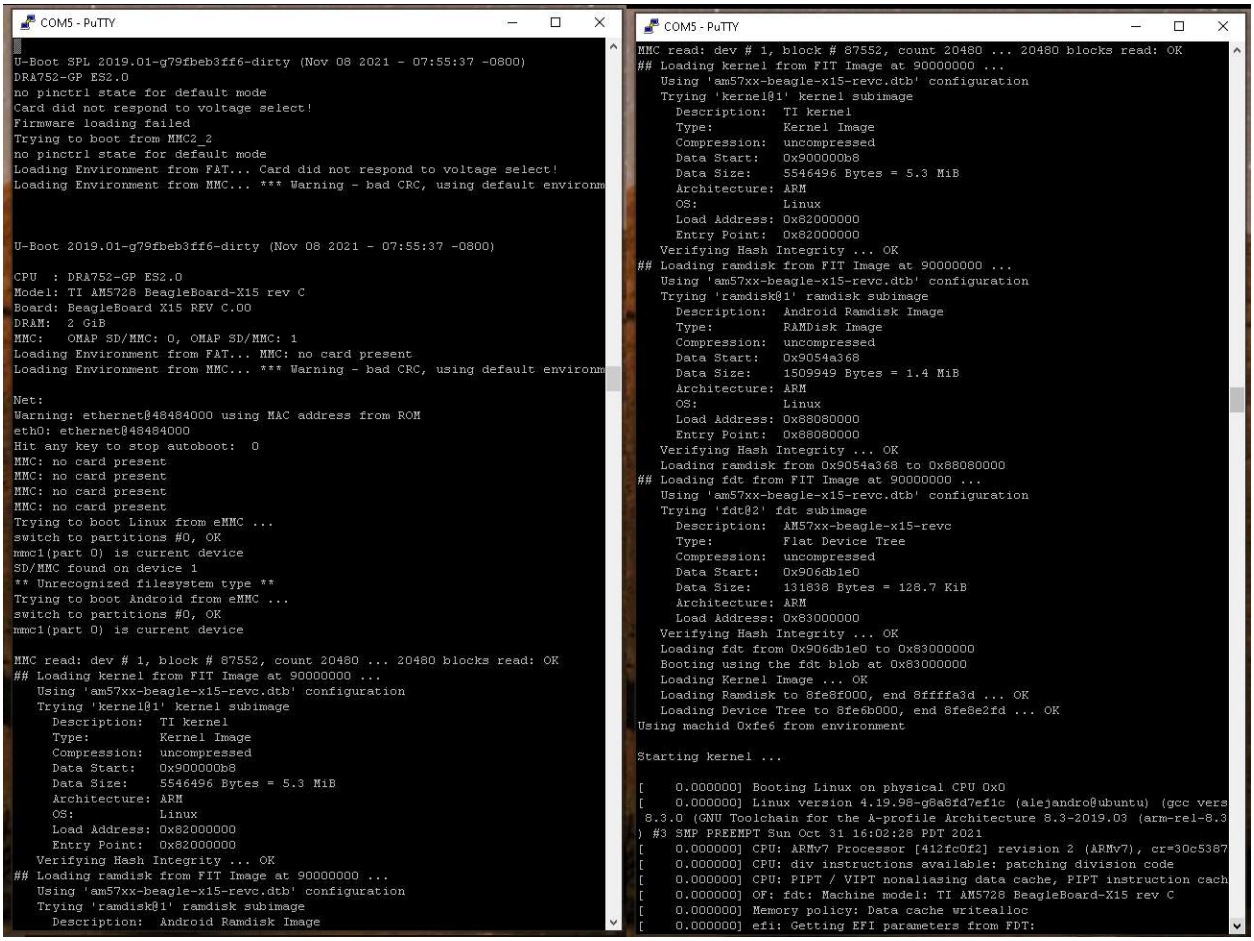



*Imagen 9. Log de arranque en terminal*


*Imagen 10. Interface principal*


*Imagen 11. Display del sistema booteando*

## 4.3. Ambiente de simulación

Google provee un simulador de Android Automotive OS incluido en Android Studio, para poder utilizar dicho simulador es necesario habilitar el canal Canary o descargar la versión Canary de Android Studio (ver Imagen 12. Android Canary).

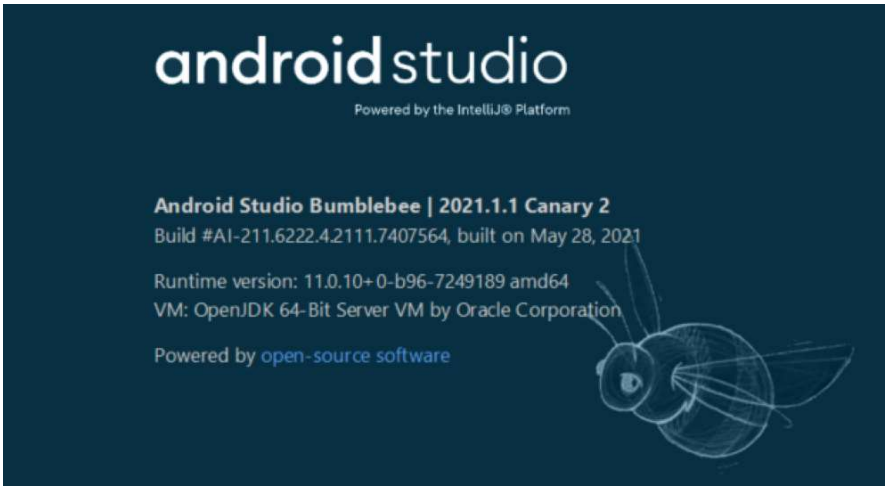



*Imagen 12. Android Canary*

Android Studio Canary © es una versión de Android Studio que ofrece las mismas características de la versión estable, al igual que nuevas características desarrolladas por el equipo de Google. Debido a la naturaleza de esta versión, es posible que algunas actualizaciones puedan introducir errores, debido a ello las soluciones a estos también son incluidas en las actualizaciones semanales que el canal Canary ofrece.

En la actualidad Google y otros fabricantes ofrecen diferentes imágenes listas para descargar, instalar y usar en Android Studio. Para este sección se mostrará como ejemplo la imagen compilada que Google ofrece. Para poder utilizarlo será necesario abrir el menú de dispositivos en Android Studio.

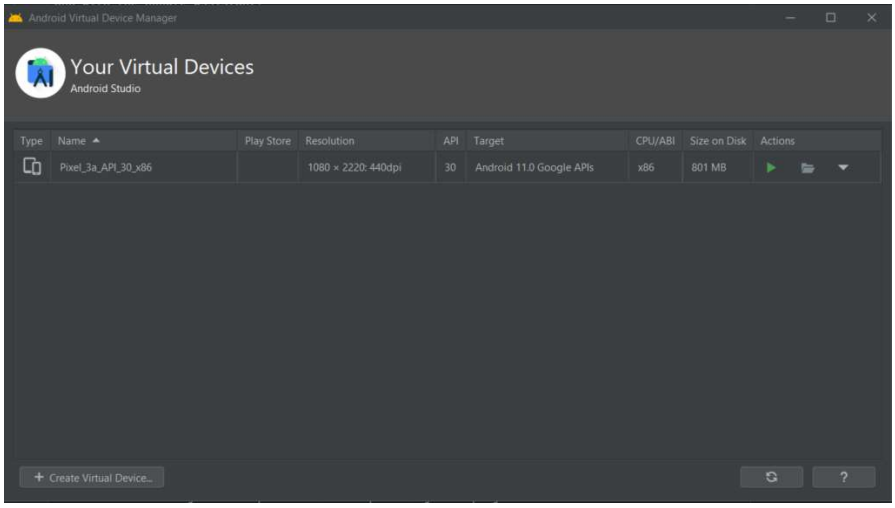



*Imagen 13. Vista Android Studio Canary*

La Imagen 13 sólo muestra un dispositivo virtual correspondiente a un smartphone, así que utilizaremos la opción “Create Virtual Device” para crear un dispositivo Automotive. La Imagen 14 muestra el menú donde se podrá crear el nuevo dispositivo virtual, en la pestaña izquierda se selecciona “Automotive” y el perfil se hará disponible.

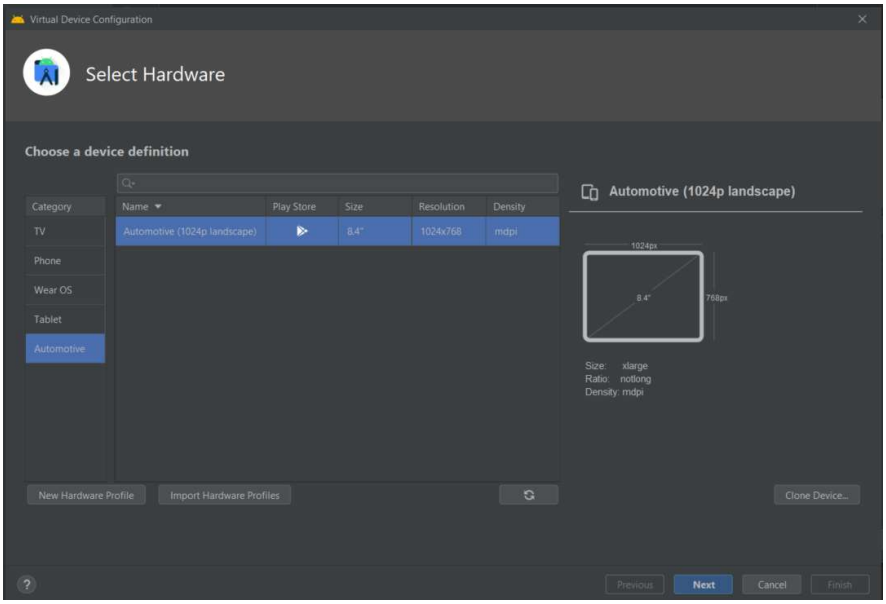



*Imagen 14. Dispositivos disponibles*

Una vez seleccionado el perfil se seleccionará también la imagen del sistema que se desea utilizar en el simulador, en el caso de este proyecto se utilizará Android Pie como se ve en la Imagen 15.

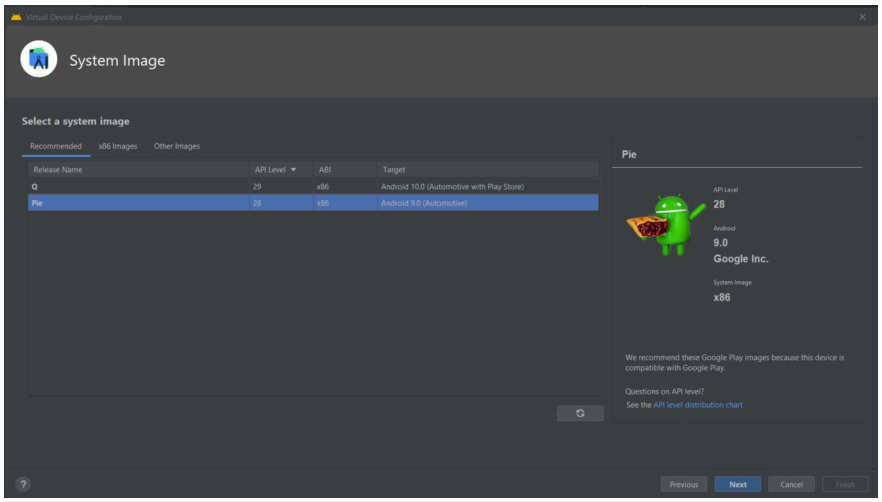



*Imagen 15. Imagen del Sistema*

Como último paso, sólo se verificará la configuración y de no necesitar cambios se seleccionará “finish” para continuar con la generación del dispositivo virtual. Esto se muestra en la Imagen 16.

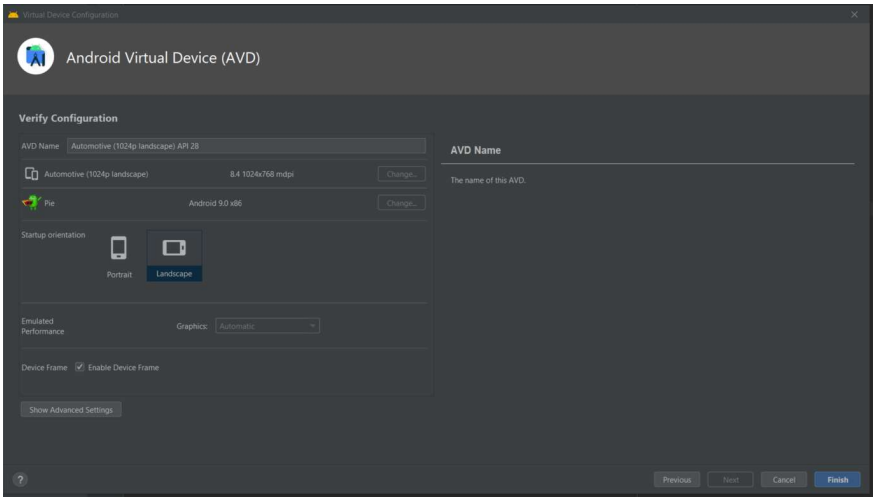



*Imagen 16. Verificar Configuración.*

Una vez listo el dispositivo virtual, este comenzará a ejecutarse y estará listo para ser usado, en la Imagen 17 se da un ejemplo de cómo se ve la GUI del simulador. Los pasos mencionados son relativamente sencillos, lo cual se debe a una ardua tarea por parte del equipo de Google por mantener accesible y amigables los recursos disponibles para desarrollo en Android. Google

describe con mayor detalle este proceso en su documentación oficial "Cómo probar apps de Android para vehículos" [17].


*Imagen 17. Vista de Automotive OS Pie 9*

En caso de querer ejecutar una aplicación desarrollada, el proceso es muy simple puesto que sólo se tiene que seleccionar la opción ejecutar en Android Studio, una vez seleccionada dicha opción la aplicación compilada comenzará a instalarse en el dispositivo virtual. Es importante recordar que Android Automotive OS tiene diferencias con respecto a la versión para celulares y una vez instalada la aplicación, esta no se ejecutara automáticamente, el usuario tiene que seleccionarla para que esta pueda ser ejecutada. La Imagen 18 muestra la pantalla de inicio usada en la aplicación desarrollada.

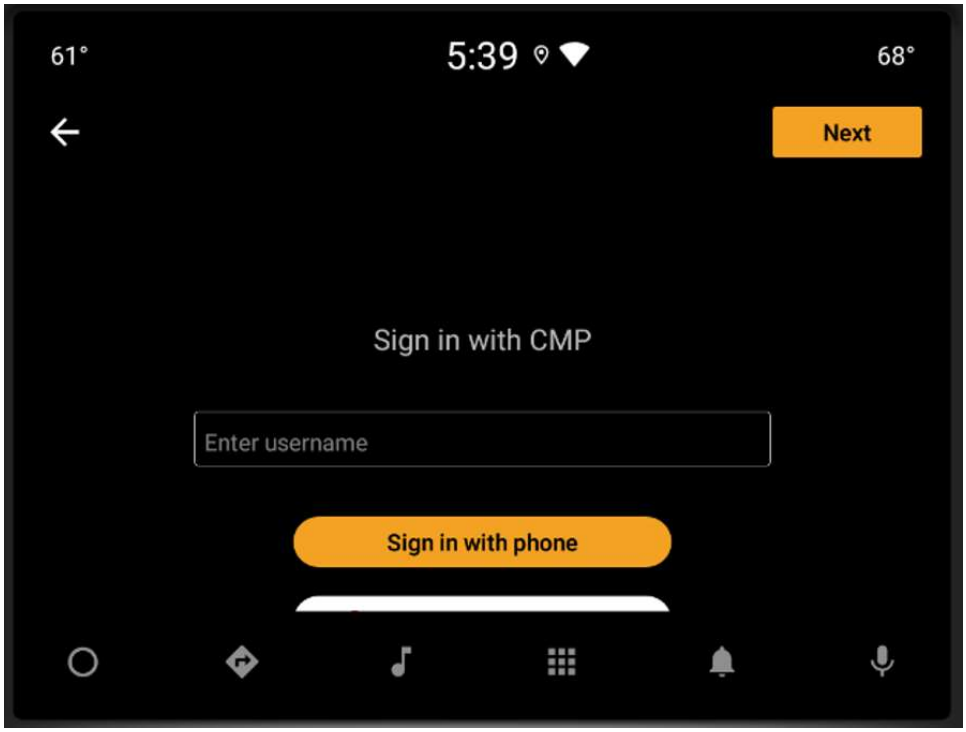



*Imagen 18. Vista de la Aplicación Media*

# 5. Arranque del sistema

Parte importante del esfuerzo inicial en el desarrollo de un proyecto embebido es el establecer e implementar la estrategia de arranque. Esta etapa crítica del sistema define el tiempo de respuesta desde que se inicializa el dispositivo, hasta el momento en que está listo para ejecutar sus tareas. Por lo general, criterios o requerimientos de estos tiempos son establecidos con base en las funciones críticas que se deben cumplir.

Iniciando el desarrollo desde cero de un sistema, incluyendo la definición e implementación de la estrategia de arranque, suele ser un lujo que la industria automotriz no puede permitirse. Incluso el reúso, tras la adaptación y actualización, de alguna generación anterior de la misma familia de proyectos o plataforma, implica un esfuerzo no trivial que tiende a demorar el trabajo en otras características requeridas para el sistema y que dependen grandemente de la capacidad de del dispositivo para estar listo y usar los recursos de hardware necesarios, así como servicios y elementos de software.

En el caso de los sistemas puramente Linux, éstos cuentan con una estrategia ya definida que suele agilizar esta etapa inicial de desarrollo. Sin embargo, siguen siendo necesarias adecuaciones y optimizaciones que permitan cumplir con los requerimientos establecidos en un proyecto.

En este capítulo se analizarán los procesos y secuencias involucrados en el arranque e inicialización del sistema para poder establecer acciones de configuración y optimización que permitan cumplir expectativas de disponibilidad tanto de funciones criticas como de aplicaciones de gran interés para la experiencia del usuario. Por lo general, el inicio del sistema incluye todos los pasos o tareas que se ejecutan desde que el sistema es arrancado hasta que se presenta el control de este al usuario. Dicha inicialización suele depender del estado inicial, ya sea desde un completo des energizado o apagado completo, a uno de latente en algún bajo consumo de potencia.

## 5.1. Generalidades

Android, incluyendo Automotive Android (AAOS), en realidad no es por sí mismo un sistema operativo creado desde cero, sino que está basado en Linux. AAOS usa el Kernel de Linux para brindar una base para el sistema como se muestra en la Imagen 19.

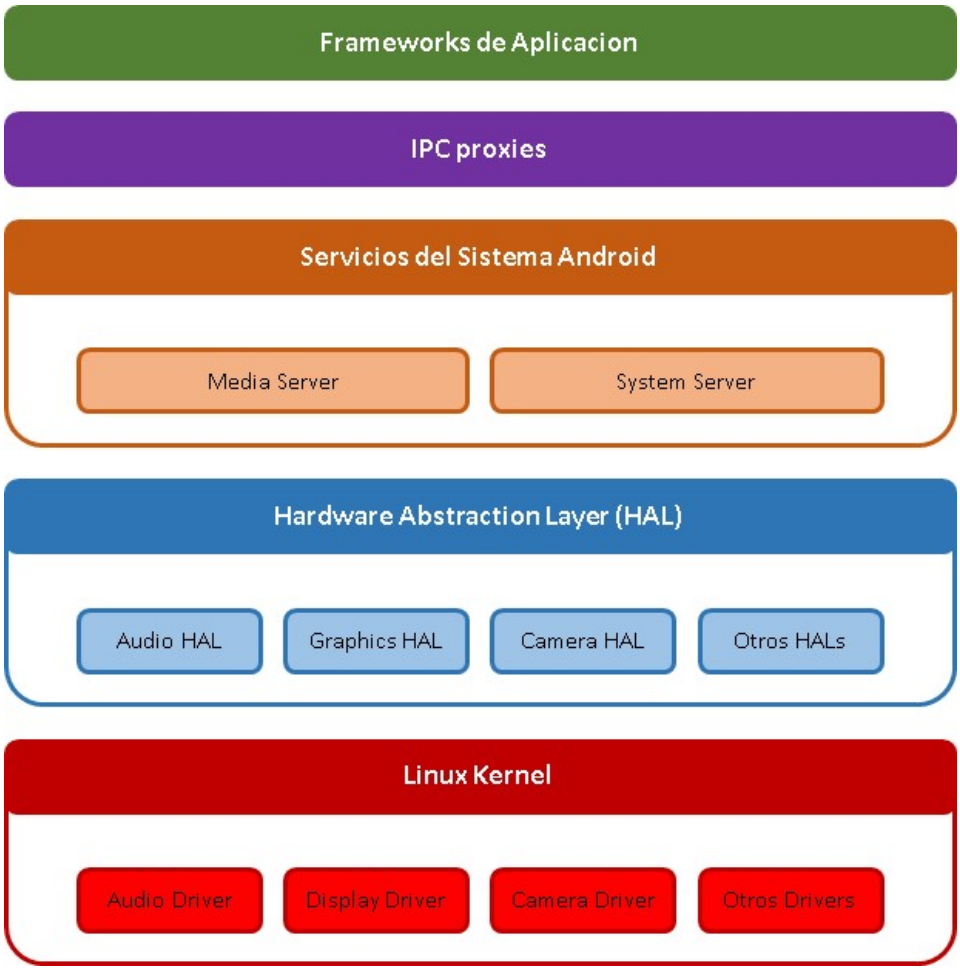



*Imagen 19. Arquitectura de Android*

En general, Android simplifica su estrategia al usar como base el kernel del sistema operativo Linux, agregar una Capa de Abstracción de Hardware (HAL), servicios específicos de Android, y un framework de manejo de aplicaciones. Debido a esto, hay una similitud entre el proceso de arranque de un sistema completamente Linux y uno Android. Resulta entonces importante señalar esta secuencia en Linux.

P. Raghavan, Amol Lad, Sriram Neelakandan mencionan [18, p. 41] que las etapas de la secuencia de inicialización en un sistema Linux embebido son como se describe en la Tabla 4.

| Secuencia | Etapa |
|---|---|
| 1 | Bootloader |
| 2 | Inicialización del Kernel |
| 3 | Inicialización del user-space |

*Tabla 4. Etapas de la secuencia de arranque en Linux*

#### 5.1.1.1 Bootloader

El bootloader es el software que se ejecuta en cuanto se enciende el dispositivo, ya sea desde un vector de reste especifico un una ROM dedicada. Sus funciones obligatorias son:

- Inicializar el Hardware: configurar el hardware y ejecutar un diagnóstico básico. Esto incluye establecer la velocidad del CPU e iniciar los controladores básicos para permitir la carga del Kernel, como es el caso del driver de la memoria flash.

- Cargar el Kernel: acceder o descargar el Kernel desde donde se encuentre almacenado y copiarlo en la región de memoria adecuada para su ejecución.

Los pasos que el bootloader sigue por lo general en procesadores ARM son (ver Imagen 20):

1. **Arranque "Booting":** Iniciando típicamente desde memoria flash, se configura la cache, se configuran registros básicos, se verifica la RAM y se corren verificaciones de hardware.
2. **Transferencia:** El bootloader se transfiere a la memoria RAM. Este paso puede incluir una etapa de compresión que suele emplearse para ahorrar espacio a un costo de la velocidad de arranque.

3. **Inicialización de Dispositivos:** Se inicializan los dispositivos básicos para poder después cargar el kernel, tales como tarjetas de red, puertos usb, etc. También se configura la consola en caso de que quiera hacerse disponible al usuario.
4. **Interfaz:** una interfaz de usuario es establecida para que, en caso de necesitarlo, el usuario seleccione que Kernel se usara.
5. **Descarga:** la imagen del kernel se descarga para su manipulación.
6. **Preparar kernel booting:** El kernel recibe parámetros de configuración o arranque en caso de estar disponibles.
7. **Kernel booting**: el control del sistema es transferido al kernel y empieza a ejecutarse.

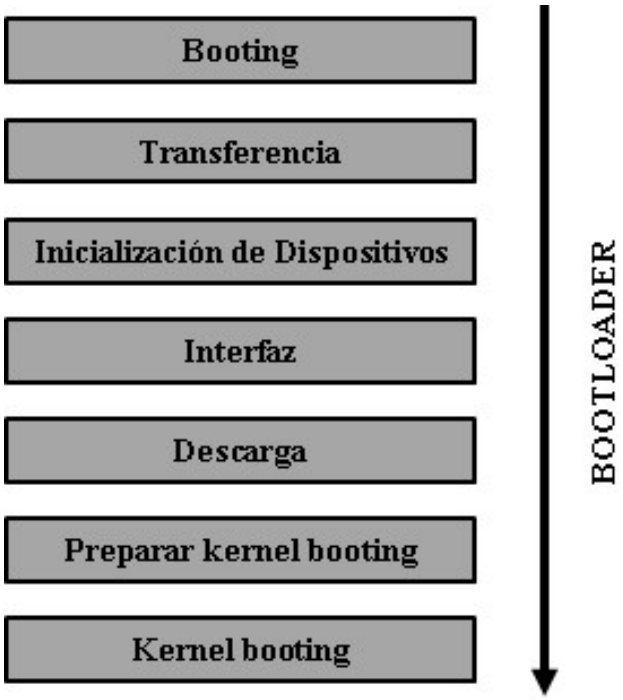



*Imagen 20. Secuencia genérica del bootloader*

#### 5.1.1.2 Inicializacion del kernel

Esta etapa realiza la operación específica del CPU/Plataforma, levanta los sub-sistemas del kernel, inicia el multitasking, monta el Sistema de archivos e inicia el user-space. Los pasos que suele seguir se muestran a continuación (Imagen 21).

1. **Iniciar CPU/Plataforma:** Se prepara el ambiente para ejecutar el primer programa, se configura la arquitectura del CPU, se inicializan las excepciones, se inicializan los procesos de interrupción, se inicializan timers, se inicializa la consola y se calibran los delay loops.
2. **Iniciar Subsistemas:** Se inicializa el scheduler, se inicializa el manejador de la memoria.
3. **Iniciar Drivers:** Los drivers son inicializados una vez que el manejador de memoria está listo.
4. **Montar sistema de archivos:** se monta el root file system que después permitirá iniciar el user-space
5. **Initcall:** se ejecutan todas las funciones etiquetadas como __init que están ligadas a la inicialización del kernel.
6. **Iniciar user-space:** el programa especial llamado init se ejecuta, dando por finalizada la inicialización del kernel y la operación del sistema cambia al user-space.

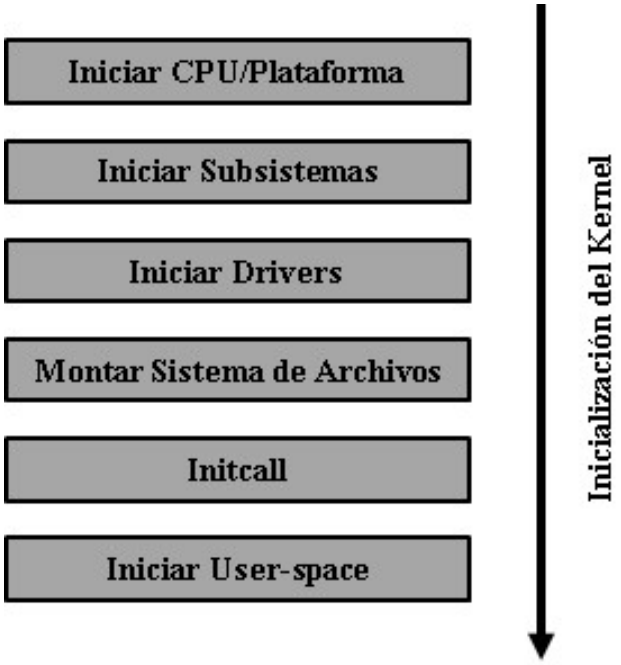



*Imagen 21. Inicializacion del Kernel*

#### 5.1.1.3 Inicializacion del user-space

La inicialización del user-space depende completamente de la distribución de Linux que se use y lo que se establezca en el programa init. En esta etapa se establece el estado del sistema o "run level" al pasarlo como argumento a init y puede ser tomado por default del archivo llamado inittab.

Una vez que el user-space está listo, los programas de usuario pueden empezar su ejecución. Aquí se establece una separación entre el Kernel-space y el user-space, que se puede describir con la Imagen 22 [19, p. 3].

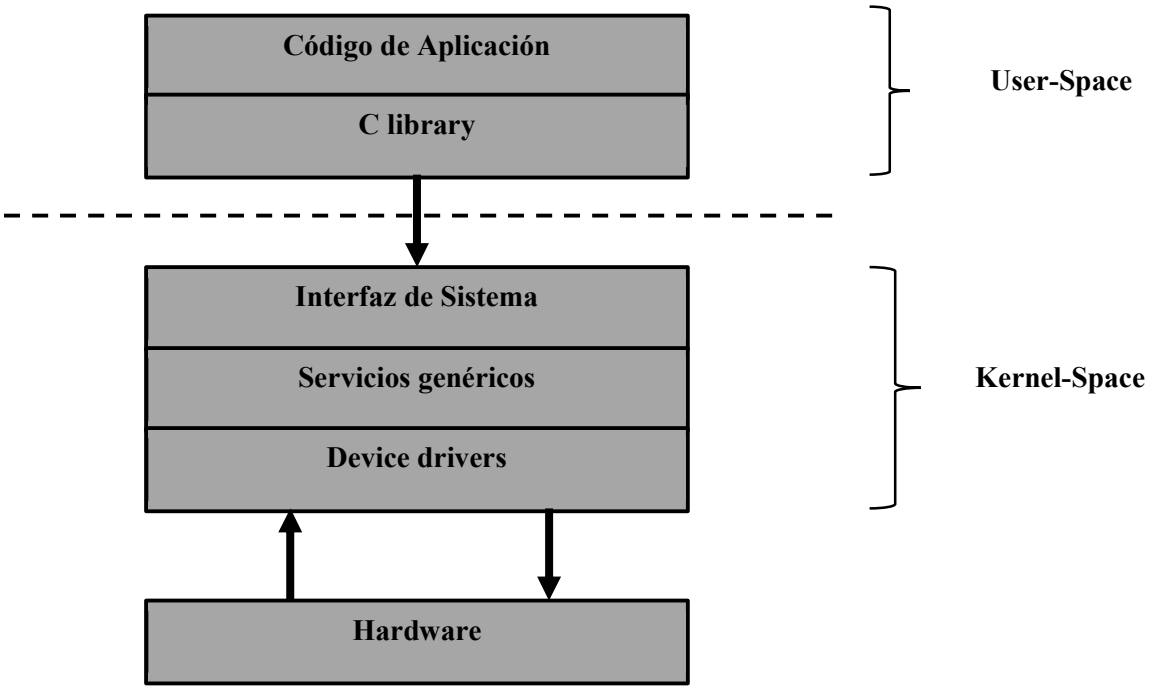



*Imagen 22. Kernel vs User Space*

Existe una discusión respecto si Android califica como una distribución de Linux precisamente por ser esta etapa en la que difiere el sistema, al igual que lo hacen típicamente las distribuciones de Linux.

## 5.2. Proceso de arranque en Android

El sistema operativo Android comparte similitudes con Linux en su secuencia de bootloader e inicialización del kernel (ver Imagen 23), sin embargo, difiere radicalmente en la etapa de inicialización del user-space. La principal diferencia se encuentra en la inicialización de la máquina virtual (Art reemplaza a Dalvik a partir de Android Lollipop) y el proceso llamado Zygote que permite un uso más eficiente del user-space al permitir a las aplicaciones compartir recursos.

La Imagen 23 muestra un diagrama de la comunidad de NXP [20] que describe el proceso de arranque en Android:

1. BOOT ROM: Al energizar el sistema, el código en la memoria ROM "boot" inicia su ejecución y carga en memoria el bootloader.

2. BOOTLOADER: En el bootloader se detecta la RAM y se carga un programa que permite ejecutar el kernel para inicializar los stacks de memoria a cero y ejecutar main.c, que a su vez inicializa los dispositivos de HW básicos.

3. Kernel: La inicialización es similar a un sistema Linux. Al comenzar su ejecución configura la memoria, el scheduler, carga drivers y busca el init. Pero, de acuerdo con [21], Android ha aplicado los siguientes cambios, entre otros, que principalmente lo diferencian:

    - Binder: un mecanismo de comunicación entre procesos (IPC) y sistema de invocación remota que está disponible en el kernel.
    - Ashmem: un alojador de memoria compartida que mejora el soporte memoria virtual en dispositivos con poca memoria.
    - Pmem: un alojador de memoria de procesos diseñado para manejar memoria físicamente contigua en dispositivos con recursos limitados.
    - Logger: es una implementación que da soporte a logcat al nivel de kernel, y que esta optimizado en escritura para una mayor velocidad.

- Wakelocks: archivos de administración de potencia que permite evitar que el dispositivo entre en bajo consumo de potencia para mejorar su respuesta.
- Oom handling: eliminación de procesos conforme se reduce la memoria, de acuerdo a reglas que pueden establecerse desde user_space
- Alarm timer: una implementación que da soporte al alarm manager de Android al nivel de kernel. En combinación con wakelock, permite al user_space indicar al kernel cuando le gustaría despertarse, independientemente del estado de potencia.
- Ram console: permite guardar mensajes del kernel a ram para poder ser consultados en caso de un kernel panic

4. Init: El proceso inicial a todos los procesos "init" es ejecutado. Similarmente a Linux, esta etapa monta directorios del file system y ejecuta init.rc. El lenguaje específico para definir init.rc se basa en cuatro clases de enunciados:
    - Acciones: secuencias de comandos por disparador "trigger" de eventos.
    - Disparadores: para identificar la ocurrencia de eventos que disparan acciones.
    - Servicios: software que se inicia o reinicia al salir
    - Opciones: modificadores al cómo y cuándo init ejecuta algún servicio

5. Zygote and Dalvik: se inicia el Zygote, que es el sistema que permite compartir código entre instancias de las Dalvik VM, las instancias en donde se ejecutan las aplicaciones. Zygote precarga e inicializa las librerías centrales. Aspectos del manejo de memoria, involucrando ambos, son detallados en 6 Manejo de Memoria RAM.

6. Servidor del Sistema: Zygote inicia todos los servicios necesarios al arrancar el servidor del sistema en cuyo contexto todos los servicios se ejecutarán, primero los servicios nativos y posteriormente todos los demás de acuerdo con su orden de inicio.

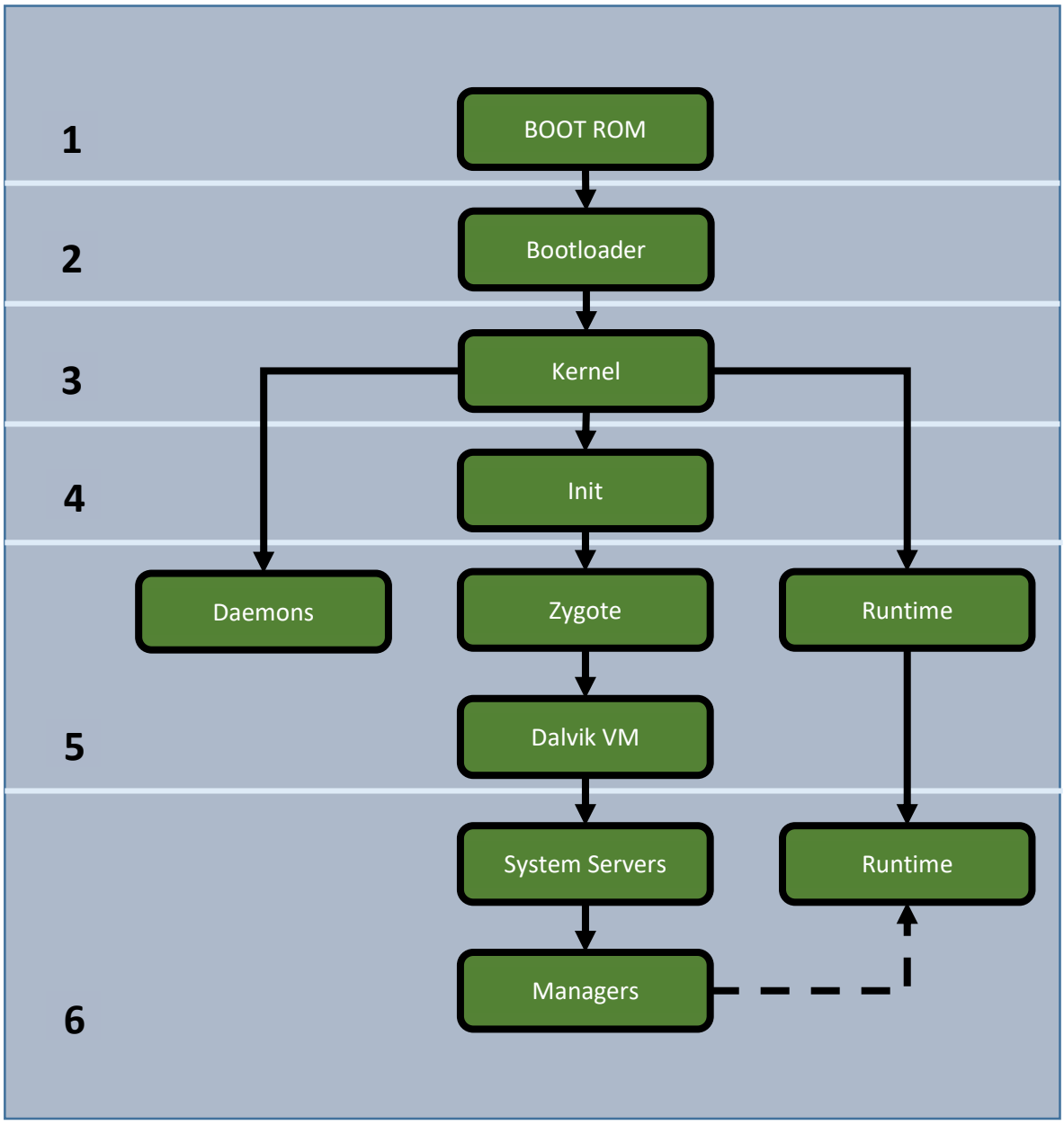



*Imagen 23. Proceso de arranque del sistema en Android*

Por último, la acción estándar "ACTION_BOOT_COMPLETED" es transmitida en modo broadcast, una de las pocas que permanecen después de actualizaciones de comportamiento con Android 8.0

## 5.3. Herramientas de “profiling”

Existen diferentes herramientas disponibles para hacer un perfil o “profiling” del arranque del sistema. Los más comunes son:

- Message Logging: esto es la captura del tiempo entre mensajes del sistema. Algunas herramientas disponibles son las siguientes:
    - Grabserial: permite la captura en la salida estándar “standard output” de los tiempos “timestamp” entre cada mensaje de sistema a través de puerto serial.
    - printk-times: es una opción del kernel (CONFIG_PRINTK_TIME) que agrega un timestamp a cada mensaje del kernel con printk. Permite recuperar la duración entre dichos mensajes, ya sea por consola o con dmesg. También hace posible controlar niveles de verbosidad o de log.

- Android logging system: es un conjunto de buffers circulares mantenidos por el proceso de sistema “logd” y que capturan logs de aplicaciones, de eventos, de sistema y de fallos, entre otros.
    - Logcat: es la herramienta que permite obtener los mensajes del sistema. Brinda diferentes opciones de formato y filtrado para los buffers de log que se desean obtener.

- Strace: permite mostrar las llamadas de sistema para un proceso, incluyendo aquellos de inicialización del sistema.

- Bootchart: Es una herramienta diseñada para capturar datos de arranque y poder desplegarlos de manera gráfica. En Android, esta funcionalidad está integrada en el programa “init”, aunque, de acuerdo con [22], su habilitación suele asociarse a un incremento en tiempo de arranque de alrededor de un 7%.

### 5.3.1 Perfil de arranque de sistema (boot “profiling”)

Una vez cargadas las imágenes del sistema como se describió en la sección 4.2.1, se puede habilitar alguna de las herramientas para perfilar el comportamiento durante el inicio del sistema.

En este caso, usamos bootchart para obtener una representación gráfica de dicho proceso. Para el sistema, basta con la creación del archivo “/data/bootchart/enabled” para que habilitemos su uso, lo cual se logra con los siguientes comandos desde la consola conectada con adb, desde el folder emmc_files:

```
$sudo adb shell ‘touch /data/bootchart/enabled’

$sudo adb reboot
```

Para recuperar el grafico, una vez que se inicializó el sistema, hay que ejecutar el script que adquiere los datos y genera la gráfica. Desde el área de trabajo, dentro del folder emmc_files, hay que ejecutar:

```
$sudo ./board-support/aosp-9.0/system/core/init/grab-bootchart.sh
```

## 5.4. Ajustes y Optimización

### 5.4.1 Guías y Recomendaciones

Android cuenta con información disponible en su sitio "Optimización de los tiempos de arranque" [23] para que desarrolladores puedan hacer ajustes en los tiempos de arranque del sistema. Entre las opciones recomendadas se encuentran:

- Ajustar la carga de inicio del sistema
- Ajustar la eficiencia de Entrada/Salida (E/S)
- Optimizar init.rc
- Optimizar las animaciones de arranque
- Ajustar parámetros de seguridad de SELinux

Una guía bastante comprensible para mejorar tiempos de arranque en un dispositivo Android similar (versión de Android OS "Marshmallow"), apoyándose en recursos de la comunidad de desarrollo de open source, se encuentra disponible en el reporte de aplicación "Android Boot Optimization on DRA7xx Devices" [24].

Una guía más extensa, también disponible por el fabricante del chip y nuevamente apoyándose en la comunidad de desarrollo Open Source, pero orientada a sistemas Linux, se encuentra disponible en el reporte de aplicación "Linux Boot Time Optimizations on DRA7xx Devices" [25].

Ambas guías son similares en las recomendaciones con boot (inicio del sistema) y recomendadas en los foros de la comunidad de desarrollo. Aunque el objetivo del reporte de aplicación en Android antes mencionado, es lograr la máxima eficiencia de arranque en un sistema que, aunque similar, difiere en las versiones de hardware y de software, para el objetivo de este proyecto basta con obtener una medida de la rapidez con que se pueden aplicar dichas optimizaciones. Para lo cual, el elemento de referencia será el arranque del sistema (boot).

### 5.4.1 Ejecución de los ajustes

Para agilizar la ejecución y prueba de los ajustes se crearon un numero de herramientas de scripting que permiten agilizar la compilación y carga de las imágenes de prueba, mismas que se encuentran disponibles en ITESO_MDE_Project:

```
alejandro@ubuntu:~/ti-processor-sdk-android-am57xx-evm-06.03.00.106$ ls -lart |
grep -i mde_
-rwxrwxr-x  1 alejandro alejandro    746 Nov  2  2021 mde_make_all.sh
-rwxrwxr-x  1 alejandro alejandro   1320 Nov  6  2021 mde_emmcfiles.sh
```

- Mde_make_all.sh: Ejecuta los siguientes pasos:
  - Construir U-boot
  - Construir el Kernel
  - Construir Androird file system para AM5, aplicando los parches de problemas en dependencias en apache-xml que son conocidos para esta versión por la comunidad de desarrollo.

- Mde_emmcfiles.sh: crea rápidamente el folder emmc_files que se usa para crear el sistema de archivos en la tarjeta de memoria o a cargar con fastboot.

Para cargas consecutivas de las imágenes no es necesario crear nuevamente la sd card. Basta con ejecutar los siguientes comandos:

1. Reiniciar el sistema con la tarjeta de memoria insertada.

2. Al iniciar, en la terminal serial, presionar cualquier tecla en la consola **inmediatamente**.

3. En la terminal, ejecute el siguiente commando para enviar el dispositivo a fastboot

```
$fastboot 0
```

4. En la consola, cambie al folder con los archivos nuevos de la emmc

```
$cd emmc_files
```

5. Ejecute fastboot para cargar los nuevos binarios y reiniciar.

```
$sudo ./fastboot.sh; sudo ./fastboot reboot
```

La imagen 24 muestra el resultado obtenido para el “profiling” del proceso de arranque del sistema, mismo que se encuentra disponible en ITESO_MDE_Project.


Boot chart for Android (1970-01-01 00:00:08)
uname: Linux 4.19.98-g8a8fd7ef1c #3 SMP PREEMPT Sun Oct 31 16:02:28 PDT 2021 armv7l
release: Android/beagle_x15_auto/beagle_x15:9/PPR1.181005.003/alejandro11041854:userdebug/test-keys
CPU: armv7l
kernel options: androidboot.selinux=permissive printk.devkmsg=on cma=256m vmalloc=400m androidboot.serialno=160120104a7c0922 console=ttyS2,115200 androidboot.console=ttyS2 androidboot.hardware=beagle_x15board
time: 00:57.60
CPU (user+sys)
I/O (wait)
Disk throughput
Disk utilization
0MB/s
Running (%cpu)
Unint.sleep (I/O)
Sleeping
Zombie
0s
5s
10s
15s
20s
25s
30s
35s
40s
45s
50s
55s
/init
/sbin/ueventd
/system/bin/logd
/system/bin/servicemanager
/system/bin/hwservicemanager
/vendor/bin/vndservicemanager
/vendor/bin/hw/android.hardware.boot@1.0-service
/vendor/bin/hw/android.hardware.keymaster@3.0-service
/system/bin/vold
sdcard
/system/bin/netd
/system/bin/iptables-restore
/system/bin/ip6tables-restore
/system/bin/iptables-restore
/system/bin/ip6tables-restore
zygote
/system/bin/patchoat
system_server
com.android.inputmethod.latin
com.android.car
com.android.systemui
webview_zygote
android.car.cluster.sample
android.ext.services
android.car.cluster.logginrenderer
com.android.phone
com.android.settings
/system/bin/hw/android.hidl.allocator@1.0-service
/system/bin/healthd
/vendor/bin/hw/android.hardware.audio@2.0-service
vehicle@2.0-ser
/vendor/bin/hw/android.hardware.cas@1.0-service
/vendor/bin/hw/android.hardware.configstore@1.1-service
/vendor/bin/hw/android.hardware.drm@1.0-service
/vendor/bin/hw/android.hardware.graphics.allocator@2.0-service
/vendor/bin/hw/android.hardware.graphics.composer@2.1-service
/vendor/bin/hw/android.hardware.memtrack@1.0-service
/vendor/bin/hw/android.hardware.usb@1.0-service
pvrsrvctl
crash_dump32
/system/bin/audioserver
/system/bin/com.android.car.powertestservice
vehicle@2.0-ser
/system/bin/com.android.car.procfsinspector
/system/bin/lmkd
/system/bin/surfaceflinger
/system/bin/thermalserviced
/system/bin/sh
/system/bin/adbd
/system/bin/cameraserver
/system/bin/drmserver
/system/bin/incidentd
/system/bin/installd
/system/bin/keystore
/system/bin/mediadrmserver
media.extractor
media.metrics
/system/bin/mediaserver
/system/bin/statsd
/system/bin/storaged
/system/bin/wificond
media.codec
/system/bin/gatekeeperd
/system/bin/perfprofd
/system/bin/tombstoned
/system/bin/bootanimation
/vendor/bin/hw/android.hardware.automotive.vehicle@2.0-service
kthreadd
rcu_gp
kworker/u4:0-pvr_fence_sync_workqueu
rcu_preempt
ksoftirqd/1
kworker/1:0-events
kworker/1:1-events
irq/71-48070000
kworker/u4:1-pvr_fence_sync_workqueu
kworker/0:2-events
kworker/1:3-pvr_timer
kworker/1:4-events_freezable
kworker/1:1H-kblockd
kworker/0:1H-kblockd
kworker/1:2H-kblockd
4b500000.aes-en
pvr_timer
PVR Linux Fence
4b700000.aes-en
pvr_fence_sync_
pvr_sync_workqu
irq/198-2-006f:
irq/197-mcp7941
480a5000.des-en
w1_bus_master1
ata_sff
scsi_eh_0
scsi_tmf_0
kworker/0:3-events_power_efficient
kworker/0:2H-mmc_complete
jbd2/mmcblk1p9-
ext4-rsv-conver
pvr_workqueue
pvr_fence_sync_
kworker/u4:2-pvr_fence_sync_workqueu
pvr_fence_sync_
irq/179-dwc3
pvr_fence_sync_
kworker/u4:3-pvr_fence_sync_workqueu
kworker/u4:4-pvr_fence_sync_workqueu
ptp0
pvr_fence_sync_
kworker/u4:5-pvr_fence_sync_workqueu
pvr_fence_sync_
kworker/u4:6-pvr_fence_sync_workqueu
http://code.google.com/p/pybootchartgui

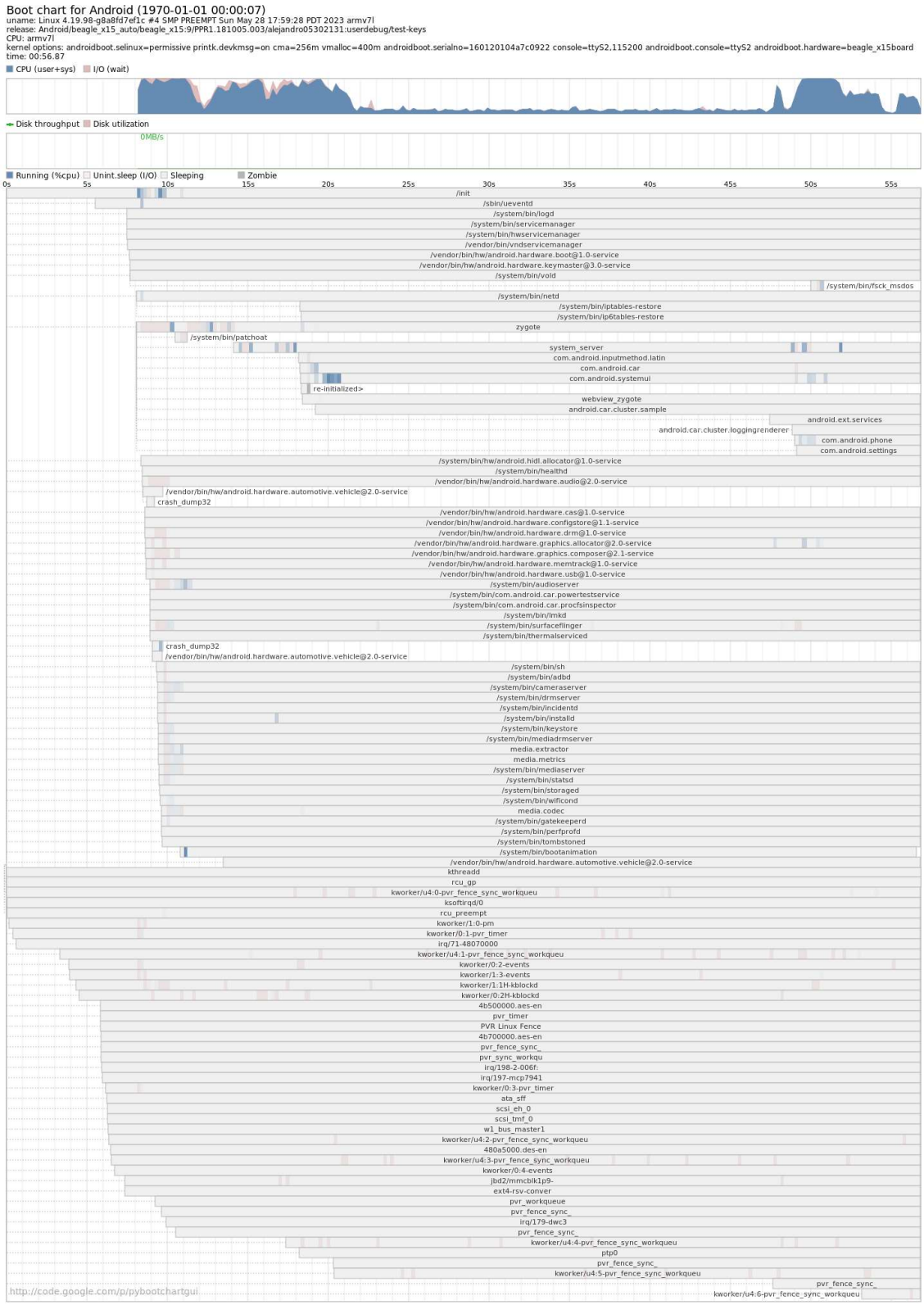



*Imagen 24. Android booting process before and after minor boot update*

## 5.5. Resultados

En resumen, el beneficio del uso de la plataforma Android Automotive para hacer un rápido ajuste en el desempeño de arranque consistió en el tiempo efectivo dedicado a las siguientes actividades:

- **Selección y habilitación de las herramientas** de medición del performance ("profiling").

El tiempo empleado en habilitar y generar una primera medida del desempeño del arranque del sistema, tras seleccionar la herramienta de "profiling" (bootchart) no excedió a un día. La herramienta se encuentra disponible en el sistema en el ambiente de desarrollo y existen numerosas referencias a su uso. Está completamente integrada en el proyecto de software desde su inicio.

- **Identificación de la estrategia** de ajustes de performance en Android:

Una guía directamente disponible en el sitio para desarrolladores de Android da un punto de inicio inmediato para plantear opciones de mejora, dependiendo del objetivo de performance buscado. Existen también ejemplos documentados de referencias "benchmark" para versiones de software y hardware específicos, como fue en este caso el "Jacinto board".

- **Implementación** de los ajustes:

La mayoría de los cambios necesarios para una rápida agilización del arranque, ya se encontraban disponibles en la comunidad de desarrollo en forma de parches o de ramas (branches) de mejora al performance. Inicialmente se intentó adquirir dichos cambios en el proyecto, pero esto resultó complicado con el uso esperado de gitlab. Los cambios se aplicaron localmente, y se encontró lo siguiente:

1. La comunidad recomienda cambios reusables, que sin embargo resultan en una mera referencia de implementación. Los cambios a los que se apuntan realmente no se pueden integrar en cualquier proyecto.

2. Existe la necesidad de evaluar el historial y contexto completo del cambio sugerido, así como las dependencias sore las cuales se implementó.

3. Al hacer ajustes en el software o configuración de etapas más tempranas de ejecución en el sistema, resultó necesario aplicar alguna otra adecuación en la capa posterior. Por ejemplo, entre u-boot y el Kernel.

4. El cambio aplicado fue trivial en este caso, el más simple:

```
#if 0
             /* Prepare console output */
             preloader_console_init();
#endif
```

- **Actualización y medición** de resultados

En sí el cambio evaluado fue la reducción de mensajes en bootloader, que esperaba un beneficio en tiempo de arranque de hasta 2 segundos (en Android Marshmellow y jacinto board), pero el bootchard en el sistema encontró sólo una mejoría de alrededor de medio segundo.

# 6. Manejo de Memoria RAM

El objetivo de este capítulo es poder analizar los procesos que Android utiliza para el manejo de la memoria RAM y para cumplir dicho objetivo es importante tener en cuenta los conceptos básicos de la memoria RAM en Android, así como también sus dos manejadores de memoria Dalvik y ART (Android Runtime).

Dalvik fue el manejador principal de la memoria y pronto este fue reemplazado por ART en Android Lollipop. Una de las mejoras más importantes fue la Recolección de Basura o "Garbage Collection", el cual es el proceso donde se limpian los objetos sin uso dentro de la ejecución de la aplicación. A pesar de que ART se lanzó para sustituir a Dalvik, este sigue ejecutándose como parte de los procesos de ART.

Debido a lo anterior, al establecer una estrategia de uso de RAM, lo ideal es tomar en cuenta la versión de Android que se usara puesto que el manejador de memoria varía entre las diferentes versiones. Para Dispositivos que ejecutan KitKat o versiones anteriores, la recolección de basura es simple "Marcar y Barrer". Los objetos en desuso se identifican y se remueven, pero los demás objetos se mantienen. Lo anterior se muestra en el primer y segundo renglón de la Imagen 25. Cuando el Garbage Collector se ejecuta, la memoria asignada (mostrada en azul) remueve los objetos sin referencias, dejando espacios de memoria libre (el mismo tamaño de los objetos removidos) en el espacio asignado para la aplicación. Si el dispositivo tiene un heap (memoria disponible) pequeño, o hay muchas recolecciones pequeñas, la memoria puede indicar que tiene 20MB de memorial libre, pero no te indica el bloque de memoria más grande con RAM libre el cual podría tener un 1MB. Lo cual sería un problema si estas intentando reservar 4MB puesto que no hay espacios libres de más de 1MB.


*Imagen 25. Las partes en azul indican la memoria en uso, y las partes en blanco el espacio libre.*

Un punto importante del manifiesto de ART es: “El recolector de basura debe ayudar, no lo contrario”. Esta nueva versión del Garbage Collector pausa la ejecución de las aplicaciones por recolección, la cual ocurre menos frecuentemente que las versiones anteriores y cuando se ejecuta, es significativamente rápido. Además, en ART, los objetos grandes tienen su propio heap ayudando a simplificar su administración de memoria y acelerando la recolección de objetos grandes.

## 6.1. Conceptos Básicos

### 6.1.1 Memoria Compartida y Privada

Dentro de Android, hay diferentes Clases, Assets (recursos) y librerías nativas que son utilizadas por todas las aplicaciones dentro del dispositivo. Si cada aplicación tuviera que mantener lo ya mencionado en memoria, sólo pocas aplicaciones podrían ejecutarse paralelamente. Para economizar, Android usa memoria compartida.

La memoria privada se utiliza en un proceso y debido a esto el 100% de la memoria asignada corresponde solamente a una aplicación. Lo anterior no aplica para la memoria compartida, ya que para esta es necesario tomar un promedio de la usada por todos los procesos.

### 6.1.2 Memoria Limpia y Sucia

La memoria sucia es memoria que sólo es almacenada en RAM; si se eliminara, la aplicación tendría la necesidad de ejecutarse otra vez para obtener los datos de vuelta. La *memoria limpia* consiste en elementos de la RAM también almacenados dentro de la memoria no volátil. Es por ello por lo que, si uno de estos datos es eliminado, este podría ser fácilmente cargado nuevamente.

En los dispositivos que utilizan Dalvik, el tipo de memoria más común es la memoria sucia y privada (esta memoria sólo es usada por una aplicación y sólo almacenada en RAM).

Los dispositivos Android contienen 2 tipos de memoria (Imagen 26):

- **RAM:** es el tipo de memoria más rápida, pero suele tener un tamaño limitado. Los dispositivos de alta gama suelen tener la mayor cantidad de RAM.
- **zRAM:** es una partición de la RAM utilizada para el espacio de intercambio. Cuando los objetos se colocan en zRAM, se comprimen todos los elementos; luego, se descomprimen cuando se les saca de allí. El tamaño de esta parte de la RAM aumenta o disminuye cuando se agregan paginas a zRAM o que se quitan de esta. Los fabricantes de dispositivos pueden establecer el tamaño máximo.

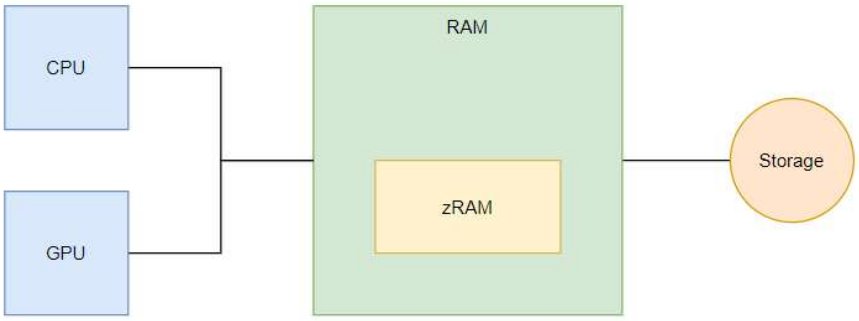



*Imagen 26. Esquema de RAM y zRAM*

Tanto Android Runtime como Dalvik, usan funciones de paginación y mapeo de memoria para administrar la RAM. Esto significa que cualquier memoria que modifique una aplicación, ya sea asignando objetos nuevos o tocando páginas con mapeo de memoria, permanece en la memoria RAM y no se puede transferir a un almacenamiento auxiliar. La única forma de liberar memoria de una aplicación es liberar las referencias a objetos que la aplicación contienen a fin de que la memoria esté disponible para el recolector de elementos no utilizados (Garbage collector). Hay una excepción: cualquier archivo que se mapee sin modificación, como el código, se puede quitar de la RAM si el sistema desea usar esa memoria en otro lugar.

Las páginas de memoria usadas para administrar RAM tienen por lo general 4KB de memoria. Estas pueden tener dos estados principalmente “libres y usadas”. Las libres como su nombre lo

indican son memoria RAM sin usar y las usadas son RAM que el sistema utiliza de manera activa y se agrupan en las siguientes categorías:

- Almacenamiento en cache
    - Privada
        - Limpia
        - Sucia
    - Compartida
        - Limpia
        - Sucia
- Anónima
    - Sucia

Las páginas limpias contienen una copia exacta de un archivo que existe en el almacenamiento. Una página limpia se convierte en una página sucia cuando ya no contiene una copia exacta del archivo. Es posible borrar las páginas limpias porque siempre se pueden volver a generar con los datos de almacenamiento. En cambio, no se pueden borrar las páginas sucias, ya que se perderían los datos.

### 6.1.3 Limpieza de la Memoria (Recolector de Basura)

En general, una vez que un objeto no tiene referencias válidas en la aplicación, esta puede ser eliminada durante la recolección de basura. El recolector de basura comienza con los objetos raíces (objetos que él sabe están vivos y son usados por algún proceso) y sigue cada referencia asociada buscando conexiones. Si un objeto no está ubicado en la lista de referencias validas, significa que este ya no es usado, y puede ser limpiado. De tal forma que, la memoria asignada a dicho objeto puede ser reusada. En la Imagen 27 los objetos sin referencias activas (flechas) están coloreados en rojo, y serán removidas cuando el evento de recolección de basura se ejecute.

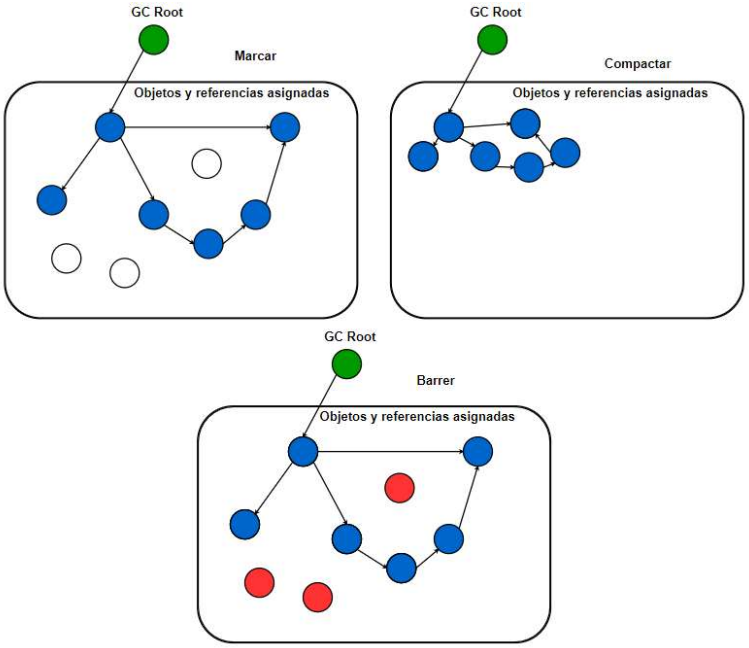



*Imagen 27. Referencias seguidas por la recolección de basura marcando objetos activos (azul), y recolectando los objetos que no tienen una referencia actual (rojo).*

La plataforma de Android se basa en la premisa de que la memoria disponible es memoria desperdiciada, de modo que intenta utilizar toda la memoria disponible en todo momento. Por ejemplo, el sistema guarda las aplicaciones en la memoria después de que se cierran para que el usuario pueda volver a abrirlas con rapidez. Por esta razón, a menudo, los dispositivos Android funcionan con muy poca memoria libre. La administración de la memoria es vital para asignar memoria de forma adecuada a los procesos importantes del sistema a las diferentes aplicaciones de usuario.

### 6.1.4 Administración de Memoria Baja

La memoria RAM es un recurso valioso en cualquier entorno de desarrollo de software, pero es más valiosa en un sistema operativo donde la memoria física puede llegar a tener restricciones como en el caso de los dispositivos móviles o sistemas automotrices. Tanto Android Runtime como la máquina virtual Dalvik realizan de manera rutinaria la recolección de elementos

no utilizados, esto no significa que se puedan ignorar en qué momento y lugar una aplicación asigna y libera memoria.

Android tiene diferentes mecanismos que ayudan a administrar de manera eficiente la memoria en diferentes entornos de hardware, uno de los más importantes para controlar situaciones donde la memoria es reducida es el “Kernel swap daemon” y “Low-memroy killer”.

**Kernel swap daemon** es parte del kernel de linux y convierte la memoria usada en memoria libre. Se activa cuando queda poca memoria libre en el dispositivo. El kernel de linux mantiene umbrales de memoria libre mínimos y máximos. Cuando la memoria libre cae por debajo del umbral mínimo, este comienza a reclamar memoria. Una vez que la memoria libre alcanza el umbral máximo este mecanismo deja de reclamarla.

El proceso es reclamar paginas limpias borrándolas porque están respaldadas por el almacenamiento y no se modificaron. Si un proceso intenta abordar una página limpia que se borró, el sistema copia la página del almacenamiento y la coloca en la memoria RAM. Esta operación también se le conoce como paginación por demanda. De acuerdo con la Imagen 28.

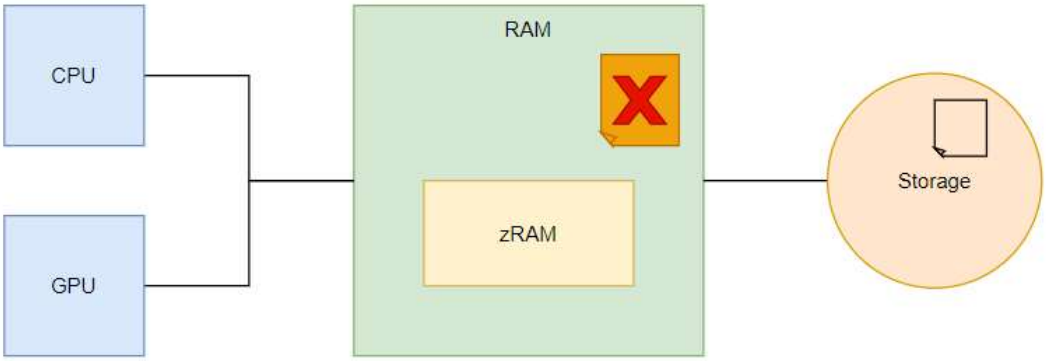



*Imagen 28. Pagina limpia, respaldada por almacenamiento, borrada.*

El kernel swap daemon puede mover paginas sucias privadas en cache y paginas sucias anónimas a zRAM, donde se comprimen. Esto hace que se libere memoria disponible en RAM (paginas libres). Si un proceso intenta tocar una página sucia en zRAM, se descomprime la página y vuelve a la RAM. Si se cierra el proceso asociado con una página comprimida, este se borra de

zRAM. Si la cantidad de memoria libre es inferior a un umbral determinado, el sistema comienza a cerrar procesos. De acuerdo con la Imagen 29.

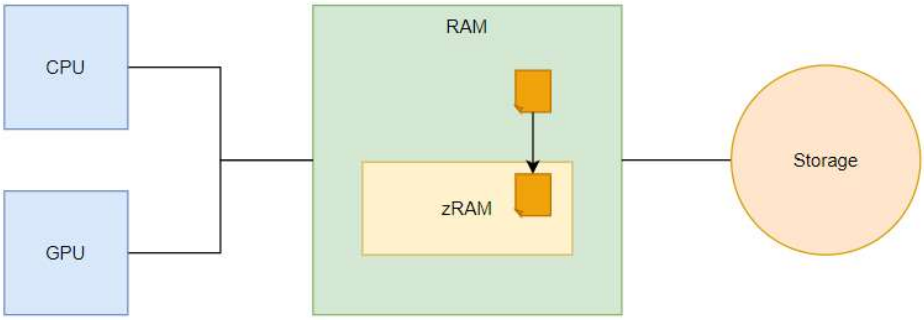



*Imagen 29. Pagina sucia trasladada a zRAM y comprimida.*

El **Low Memory Killer (LMK)** es otra característica dentro de Android y se utiliza cuando el kernel swap daemon no puede liberar suficiente memoria para el sistema. Cuando condiciones de memoria baja son identificadas, el sistema notifica a una aplicación que se está agotando la memoria y que debería reducir sus asignaciones. Si esto no es suficiente, el Kernel comienza a cerrar procesos para liberar memoria.

Para decidir qué proceso finalizar, LMK usa una ponderación de "memoria insuficiente" para priorizar los procesos en ejecución. Primero se deben cerrar las aplicaciones en segundo plano y después los procesos del sistema (los últimos que se cierran). En la siguiente lista se muestran las categorías de ponderación de procesos y aplicaciones en orden descendente; primero se cerrarán los procesos de la categoría de mayor ponderación.

1. Background Apps
2. Previous App
3. Home App
4. Services
5. Perceptible Apps
6. Foreground App
7. Persistent
8. System

9. Native

## 6.2. Memory Profiler

Solucionar problemas relacionados con el uso de memoria pueden llegar a ser complicados y el utilizar herramientas enfocadas a la memoria podrían mejorar los tiempos de solución del problema. Android Studio incluye una herramienta llamada “Memory Profiler”, en el cual se podrá observar el uso de la memoria RAM de los procesos seleccionados o del sistema completo, la Imagen 30.

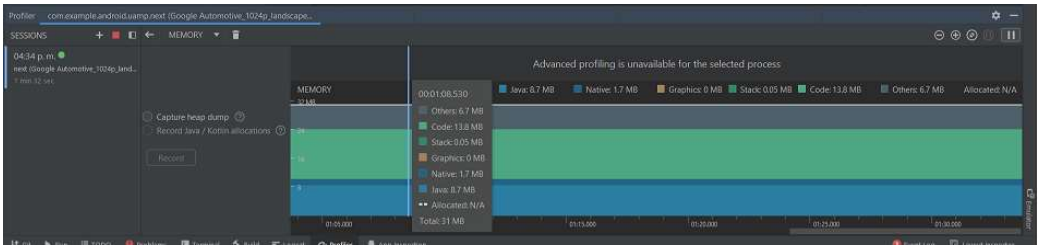



*Imagen 30.Memory Profiler incluido en Android Studio*

Esta herramienta puede ser utilizada con el dispositivo virtual o físico, lo cual es muy útil al momento de comparar comportamientos.

El Memory Profiler puede ser usado para:

- Buscar patrones de uso de memoria no deseados.
- Hacer grabaciones del uso del heap, para buscar objetos en la memoria en lapsos de tiempo grabados.
- Hacer grabaciones de las asignaciones de memoria durante la interacción del usuario para identificar exactamente donde se ejecuta su código.

La Imagen 31 muestra una vista del Memory Profiler. Se puede observar la línea de tiempo y el incremento en el uso de la memoria correspondiente a la ejecución de la aplicación de media utilizada en este proyecto.

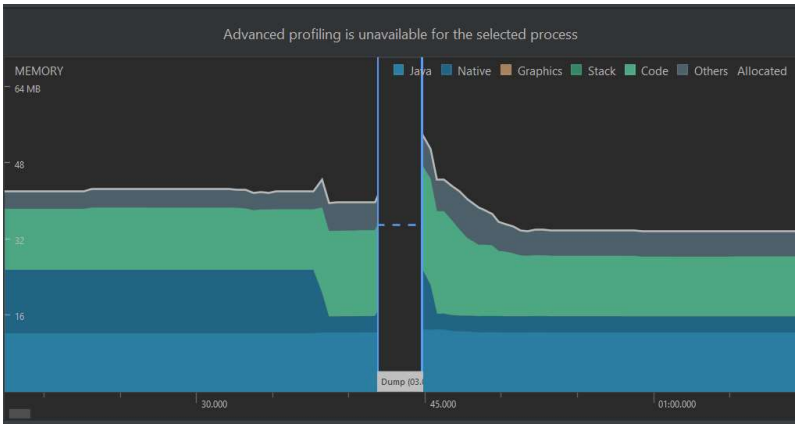



*Imagen 31. Línea de tiempo en el Memory Profiler*

## 6.3. Aplicación de Media

Se desarrolló una aplicación capaz de reproducir contenido de audio para tener un entorno donde se pueda analizar el uso de la memoria RAM, así como también, tener una comparación con respecto a el tiempo de desarrollo invertido. La información obtenida para el desarrollo de esta aplicación, así como la arquitectura base, fue provista por Google dentro de su documentación. Esta aplicación es capaz de manejar listas de canciones, manejar álbumes por autores, y reproducir contenido de audio.

### 6.3.1 Arquitectura

El desarrollo de la arquitectura está basado en un patrón de diseño llamado Cliente/Servidor como se muestra en la Imagen 32, el cual ha sido usado de manera tradicional para este tipo de aplicaciones. El cliente principal será un “Activity” dentro de la aplicación que incluye a los módulos de software Media Browser, Media Controller y la interfaz de usuario (UI).

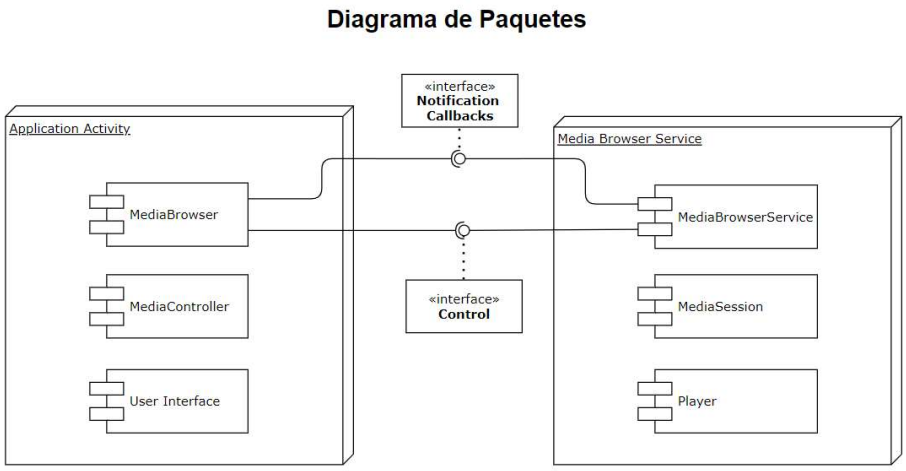



*Imagen 32. Vista de los paquetes usados en la aplicación.*

La clase MediaBroserService que se encuentra dentro del paquete servidor, permite a otras aplicaciones además de la principal poder encontrar la aplicación de medios y solicitarle reproducción de contenido.

El módulo MediaBroserService proveerá dos funcionalidades principales:

- El objetivo de usar el servicio MediaBrowserService será que otros componentes y aplicaciones que implementen un módulo "Media Browser" podrán descubrir el servicio, crear su propio controlador de media, y compartir funcionalidades con respecto a la sesión. De esta manera otro tipo dispositivos como WearOS y otras aplicaciones pueden obtener acceso a la aplicación.
- El módulo antes mencionado, también provee APIs para "Browsing". El uso de ésta es completamente opcional, y permite a los clientes hacer listas de servicio y crear representaciones de ellas de manera jerárquica, las cuales se podrían usar como listas de reproducción, librería de contenidos y otro tipo de clasificaciones.

La información relacionada a esta clase puede ser encontrada en "MediaBrowserService" [26]

Para el caso de este proyecto usaremos la clase "MediaBrowserService" y la llamaremos "MusicService" (ver Imagen 33. Esta imagen muestra una vista detallada de la clase MusicService.), en donde MusicService es responsable de:

- Instanciar y ejecutar el reproductor de audio (ExoPlayer).
- Crear y mantener la sesión de media.
- Mantener las notificaciones que provienen desde el reproductor de audio.
- Crear la lista de reproducción.

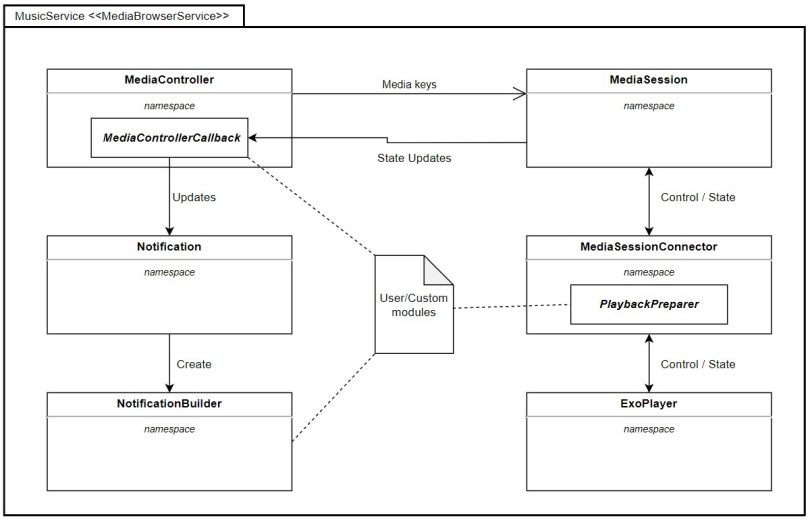



*Imagen 33. Esta imagen muestra una vista detallada de la clase MusicService.*

Como parte de las funcionalidades de la clase MusicService, la instancia de la clase MediaSessionCompact llamada "MediaSession", mantiene la sesión de reproducción de media que está en ejecución, esta clase provee varios mecanismos para controlar la reproducción, recibir estados, así como también actualizaciones de los metadatos del archivo en reproducción. La principal motivación de esta clase es proveer una manera universal de interactuar con un reproductor de audio o video. Esta clase informa a Android que un archivo de media se está reproduciendo en una aplicación y delega los controles de la reproducción a la aplicación. Esta clase fue desarrollada por Google y sus servicios se usarán dentro de este proyecto.

Para asegurarnos que la aplicación tiene acceso a los recursos de audio, un token deberá ser otorgado por medio de la clase MediaControllerCompact, dicho token nos dará acceso a la reproducción de los archivos, así como también tener control del reproductor. Hay diferentes

callbacks que se pueden implementar para recibir metadatos, así como también eventos, las cuales pueden ser definidos durante la instanciación de esta clase. La Imagen 34 muestra la interacción entre los diferentes componentes.

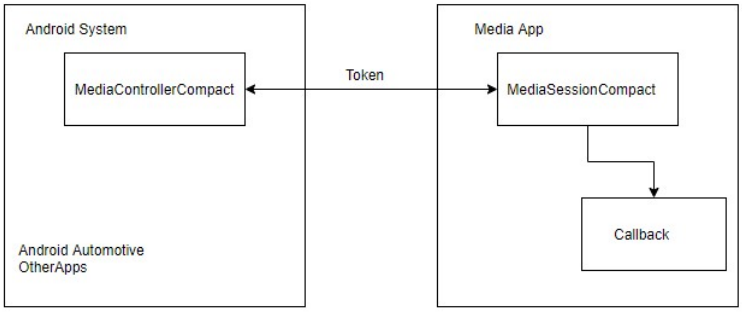



*Imagen 34. Token de acceso*

La comunicación entre la sesión de media y ExoPlayer se lleva a cabo usando la clase "MediaSessionConector" [27] que provee ExoPlayer como parte de las interfases disponibles y es totalmente compatible con la clase "MediaSessionCompact" que se mencionó anteriormente. Esta clase es un conector que escucha todos los eventos realizados por el controlador de la sesión de media y los implementa llamando de manera apropiada las interfaces del reproductor.

Otra clase importante durante la implementación de aplicaciones de reproducción de medios es la clase "MediaControllerCompact" la cual es parte de las librerías de media provistas por Google. Esta clase es un puente para comunicar la aplicación con la sesión de media puesto que recibe eventos de los botones y los pasa a la sesión de media, así como también los estados y las actualizaciones de los metadatos a través de callbacks (la interacción se puede observar en la Imagen 35. Dichas callbacks son opcionales dependiendo del caso de uso de la aplicación, también las callbacks pueden ser cambiadas durante ejecución en caso la aplicación soporte más de un reproductor de medios.

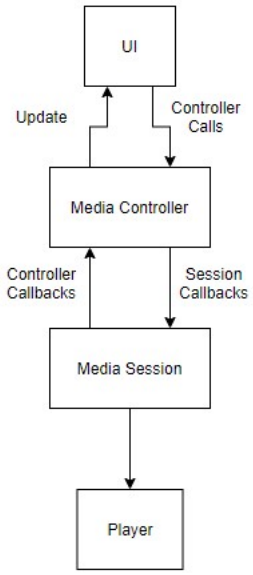



*Imagen 35. Comunicación entre la sesión de media y el controlador.*

La comunicación entre las clases MediaController y MediaBrowser se hizo mediante otra clase llamada “MusicServiceConnection” la cual implementa un patrón singleton para conectarse con la clase “MusicService” (la cual mantiene la sesión de media). Esta clase es responsable de diferentes actividades, pero una de las principales será permitir a la interfaz de usuario recibir la lista de álbumes y canciones mediante una suscripción a dichos servicios.

La ventaja de usar Android y sus librerías de Media, son la gran cantidad de APIs disponibles y la flexibilidad de uso que hay, pues para levantar una sesión de reproducción de multimedia, sólo es necesario un número reducido de llamadas a las clases de la librería Media-Compact.

### 6.3.2 Exoplayer

ExoPlayer es una aplicación de reproducción de multimedia para Android, la cual provee una alternativa más para las APIs del reproductor de media de Android en reproducción de audio y video, los cuales pueden ser archivos locales o remotos por medio de internet.

Exoplayer tiene diferentes ventajas sobre el reproductor de media de Android, algunas de sus ventajas son:

- Menor variación de comportamiento entre diferentes dispositivos.
- A diferencia del reproductor de medios de Android, se puede actualizar el reproductor junto con su aplicación. Debido a que ExoPlayer es una biblioteca que está incluida en el instalador de la aplicación, el desarrollador tiene control sobre la versión que usa y puede actualizar fácilmente a una versión más nueva como parte de una actualización regular de la aplicación.
- Capacidad de personalizar y adaptar el reproductor al caso de uso necesario. ExoPlayer está diseñado específicamente con esta mentalidad y permite reemplazar muchos componentes con implementaciones personalizadas.

La principal motivación de usar ExoPlayer fue su accesibilidad, así como también, la capacidad de mantener control sobre la versión usada. Pues es importante asegurar el funcionamiento y compatibilidad de la aplicación, como antes se mencionó en secciones previas, el tener un error que pueda ser observado por el usuario final, podría hacer disminuir la confianza que los usuarios tienen, así como también que tomen la decisión de usar otras opciones.

## 6.4. Uso actual de RAM

Esta sección está enfocada en presentar y analizar el uso actual de RAM consumido por la aplicación de media, para obtener dicha información se consideraron algunos casos de uso los cuales se consideran como los más significativos o los que nos podrían dar más información acerca de la aplicación. Los casos de uso que se revisaran son los siguientes:

1. Cuando se inicia la reproducción de música.
2. Durante la reproducción de música.
3. Cuando se pausa la reproducción de música.

Para obtener la información presentada, así como también analizar los datos se analizaron las siguientes herramientas:

- **Android Studio Profiler:** La información relacionada a Android Studio Profiler es mencionada en la sección 6.2.
- **MAT Tool (Eclipse Memory Analyzer):** es un analizador para el heap en java que ayuda a encontrar memory leaks y reducir el consumo de memoria. Este es de uso libre y disponible en https://www.eclipse.org/mat/ .

Para explicar un poco más del proceso del cómo se están revisando los datos obtenidos, tomaremos como ejemplo un estado “IDLE” de la aplicación, donde la aplicación ya ha iniciado y se encuentra activa, pero no se ha iniciado la reproducción de ningún contenido. La Imagen 36 muestra en una línea de tiempo la información disponible en el profiler de Android Studio, la cual corresponde a diferentes áreas de relevancia en el dispositivo.

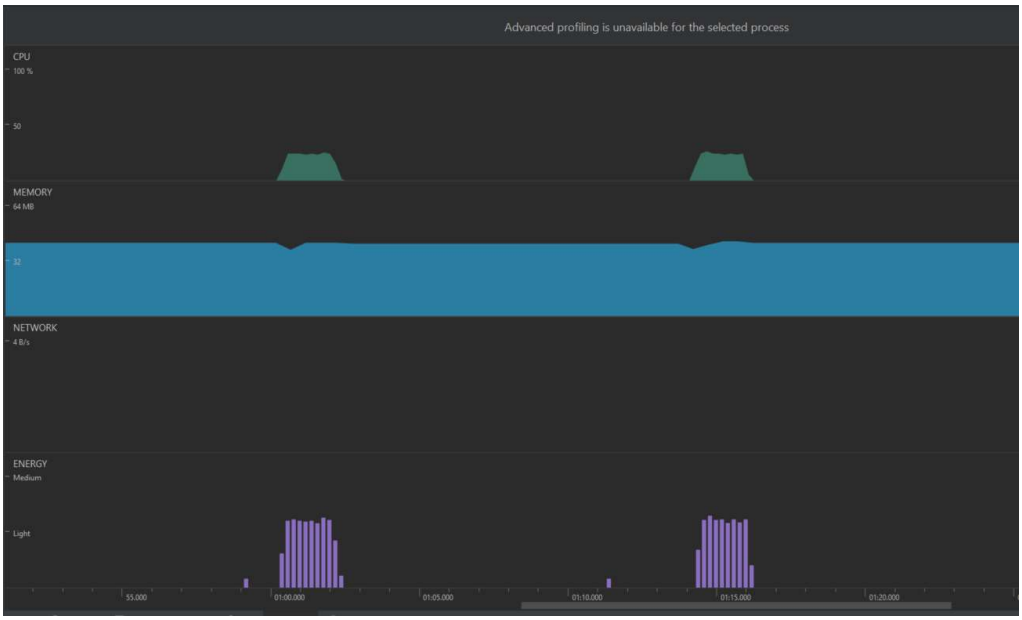



*Imagen 36. Vista de Android Studio profiler*

Si bien, Android Studio Profiler provee la información de CPU, Network, Batería y otras, la información relevante para este trabajo es la relacionada a la memoria consumida por la aplicación. Y para esto, el profiler da la opción de enfocarse sólo en la memoria.

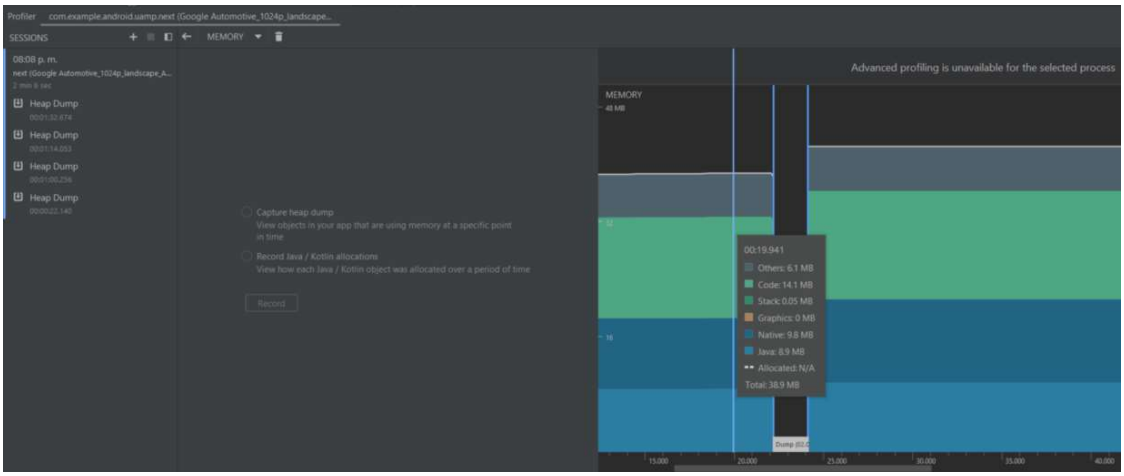



*Imagen 37. Vista del memory profiler*

Los datos que muestra el profiler en la Imagen 37, corresponden a diferentes objetos asociados al proceso de la aplicación de media, donde también se incluyen objetos que son compartidos por otras aplicaciones como pueden ser librerías u objetos relacionados al sistema, es

por ello por lo que el profiler desglosa la memoria en diferentes elementos, así como también la distingue con colores. La imagen de Imagen 37 muestra que la memoria usada en total es de 38.9 MB en el momento que hicimos la obtención de datos, pero si bien mucha de esta memoria puede ser descartada ya que corresponde a otros elementos como se mencionó antes, debido a ellos tomaremos sólo la memoria de Stack y Code que corresponde al heap de la aplicación. En total tomando en cuenta el stack y heap de la aplicación podríamos decir que está consumiendo un total de 14.15 MB sin elementos compartidos. En el caso de querer saber datos específicos de los objetos de estas regiones, el profiler de Android se puede configurar de la manera en que se muestra en la Imagen 38.Vista detallada del heapImagen 38.

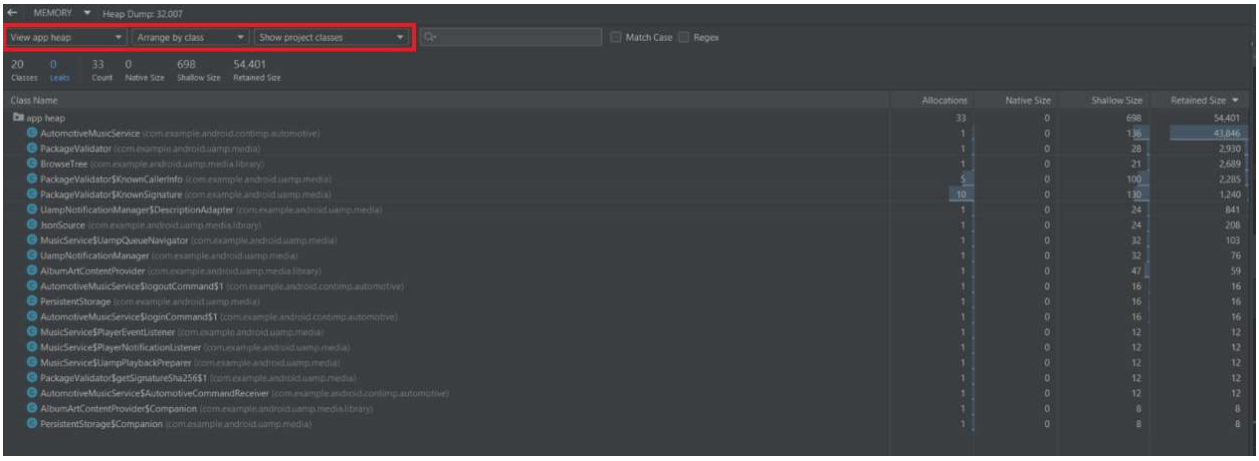



*Imagen 38.Vista detallada del heap*

La ventaja de usar el analizador de memoria en Java de eclipse es que este puede realizar análisis sobre los datos obtenidos para identificar fugas de memoria o posibles problemas. En la Imagen 39 se muestra el resultado del análisis en el modo IDLE.

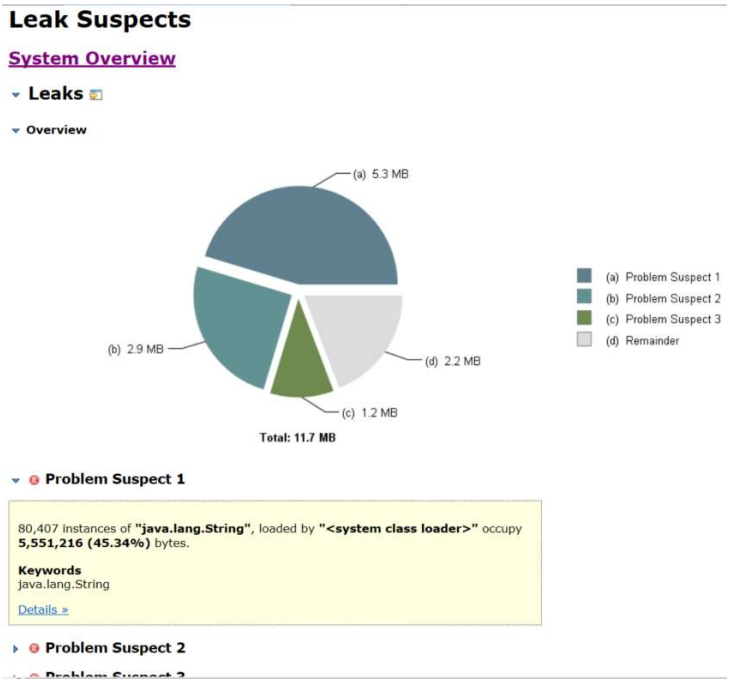



*Imagen 39. Reporte Leak suspects en Mat Tool*

Se puede observar que los resultados arrojan que hay 3 posibles objetos que podrían generar fugas de memoria debido al tamaño que tienen las instancias de este objeto. En este ejemplo los 3 posibles problemas muestran sospechas en clases que corresponden a tipos de datos usados en Java, no se consideran como problemas al valorarlos.

### 6.4.1 Casos de uso y sus mediciones

#### 6.4.1.1 Cuando se inicia la reproducción de música.

- Dependencias:
  - Conexión a internet.
- Precondición:
  - Aplicación inicializada y seleccionada.
- Descripción:
  - En este caso de uso el objetivo principal es tomar mediciones de la memoria RAM mientras el reproductor de media carga los datos correspondientes a la canción seleccionada.
- Procedimiento:
  - Iniciar CMP.
  - Entrar al menú álbumes.
  - Seleccionar un álbum.

- Seleccionar una canción.
- Mientras la canción está cargando, comenzar a grabar con memory profiler.

Como resultado de las mediciones obtenidas, el uso total de memoria fue de 47 MB totales. La Imagen 40, Imagen 41 e Imagen 42 muestran los resultados obtenidos.

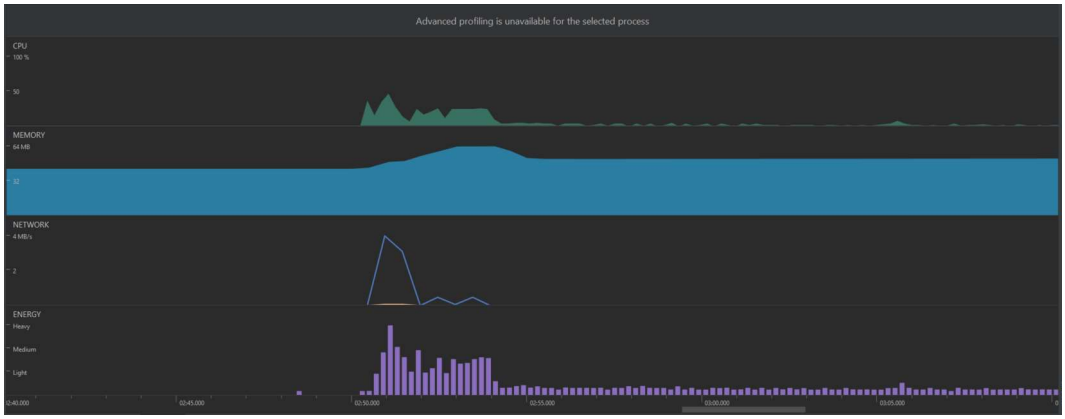



*Imagen 40. Snapshot de la grabación de recursos utilizados.*

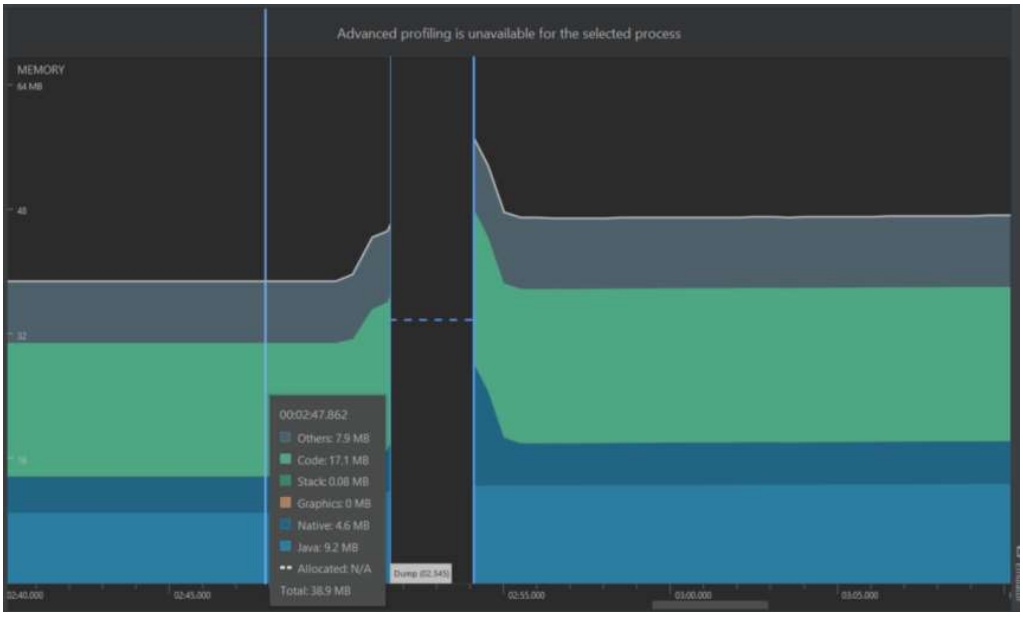



*Imagen 41. Snapshot del uso de memoria mostrando el consumo antes de iniciar la reproducción de audio.*

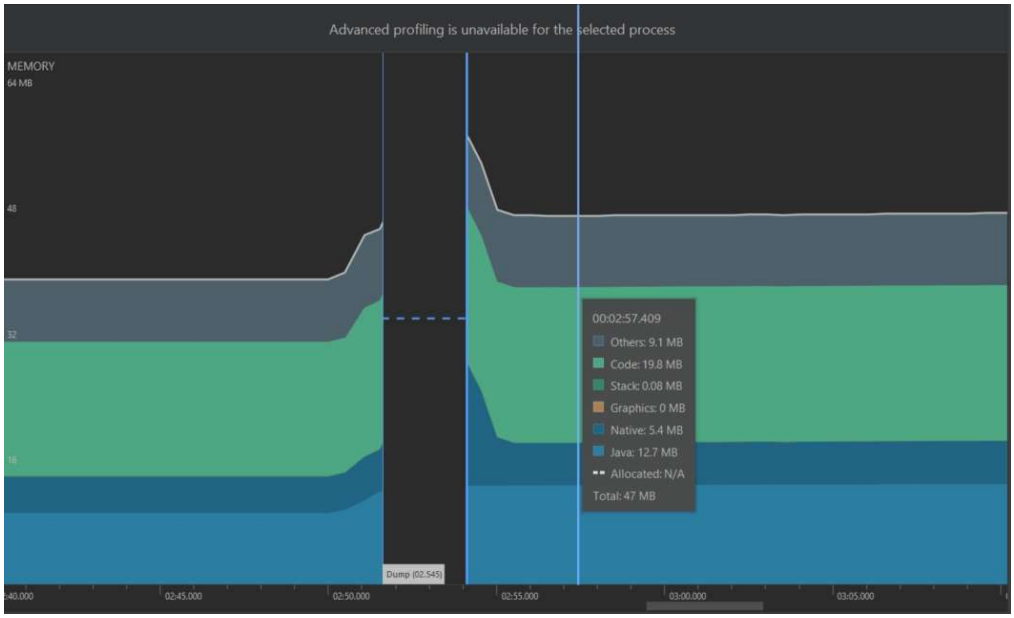



*Imagen 42. Snapshot del uso de memoria mostrando el consumo después de iniciar la reproducción de audio.*

#### 6.4.1.2 Durante la reproducción de música.

- Dependencias:
  - Conexión a internet.
- Precondicion:
  - Aplicación inicializada y seleccionada.
- Descripcion:
  - En este caso de uso el objetivo principal es tomar mediciones de la memoria RAM mientras el reproductor está reproduciendo un archivo de audio de manera remota.
- Procedimiento:
  - Iniciar CMP.
  - Entrar al menu de albumes.
  - Seleccionar un album.
  - Seleccionar una cancion.
  - Esperar a que la cancion comienze a reproducirse.
  - Comenzar a grabar con el memory profiler.

Como resultado de las mediciones obtenidas, el uso total de memoria fue de 40.1MB totales. La Imagen 43 e Imagen 44 muestran los resultados obtenidos.

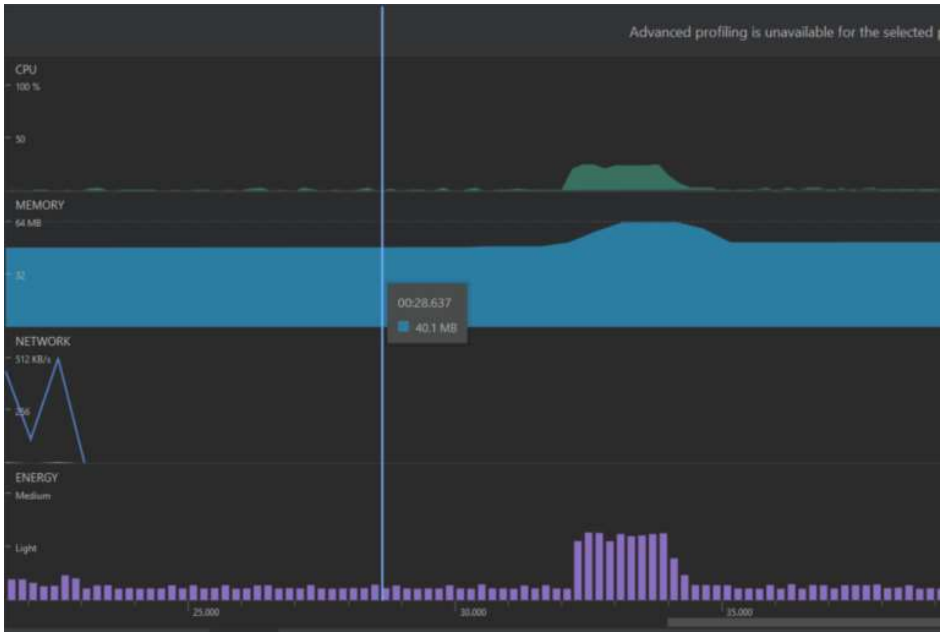



*Imagen 43. Snapshot de la grabación de recursos utilizados.*


*Imagen 44. Snapshot del uso de memoria mostrando el consumo durante la reproducción de audio*

#### 6.4.1.3 Cuando se pausa la reproducción de música.

- Dependencias:
  - Conexión a internet.
- Precondición:
  - Aplicación inicializada y seleccionada.
- Descripción:
  - En este caso de uso el objetivo principal es tomar mediciones de la memoria RAM después de que el reproductor haya pausado la reproducción de un archivo de audio.
- Procedimiento:
  - Iniciar CMP.
  - Entrar al menú de álbumes.
  - Seleccionar un álbum.
  - Seleccionar una canción.
  - Esperar a que la canción comience a reproducirse.
  - Esperar a que la canción alcance la mitad del tiempo de reproducción.
  - Pausar la reproducción actual.
  - Comenzar a grabar con el “memory profiler”.

Como resultado de las mediciones obtenidas, el uso total de memoria fue de 48 MB totales. La Imagen 45, Imagen 46 e Imagen 47 los resultados obtenidos.

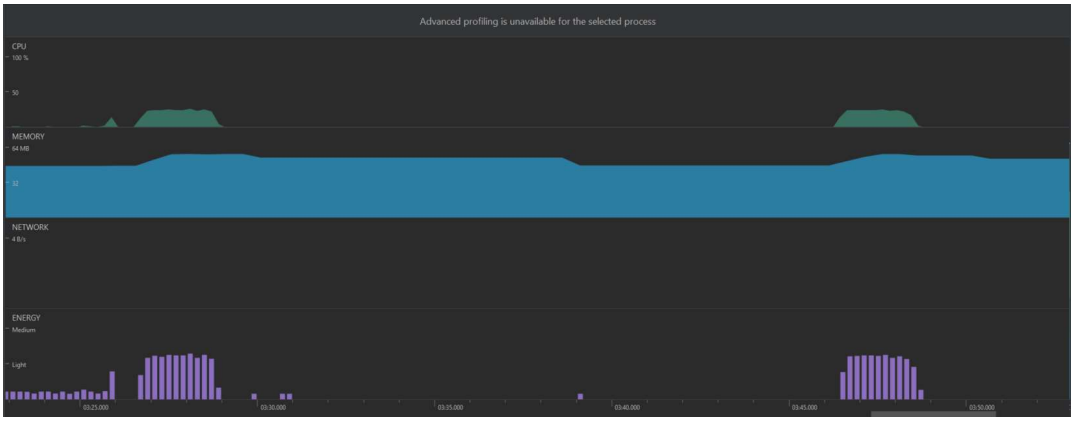



*Imagen 45. Captura de pantalla de la grabación de recursos utilizados.*

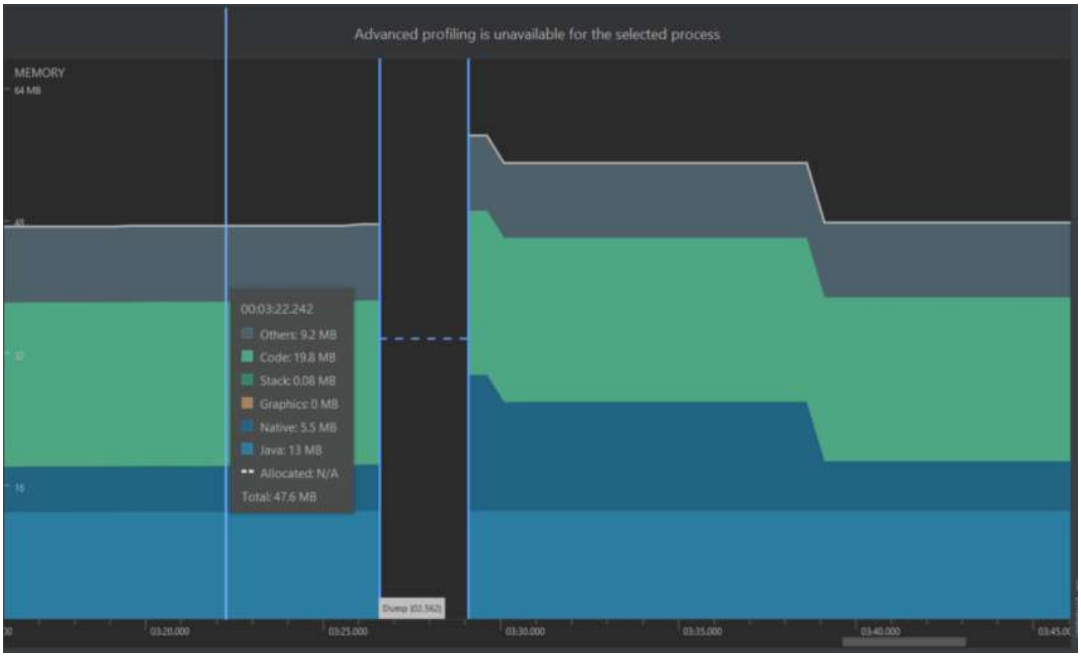



*Imagen 46. Captura de pantalla del uso de memoria mostrando el consumo antes de pausar la reproducción de audio*

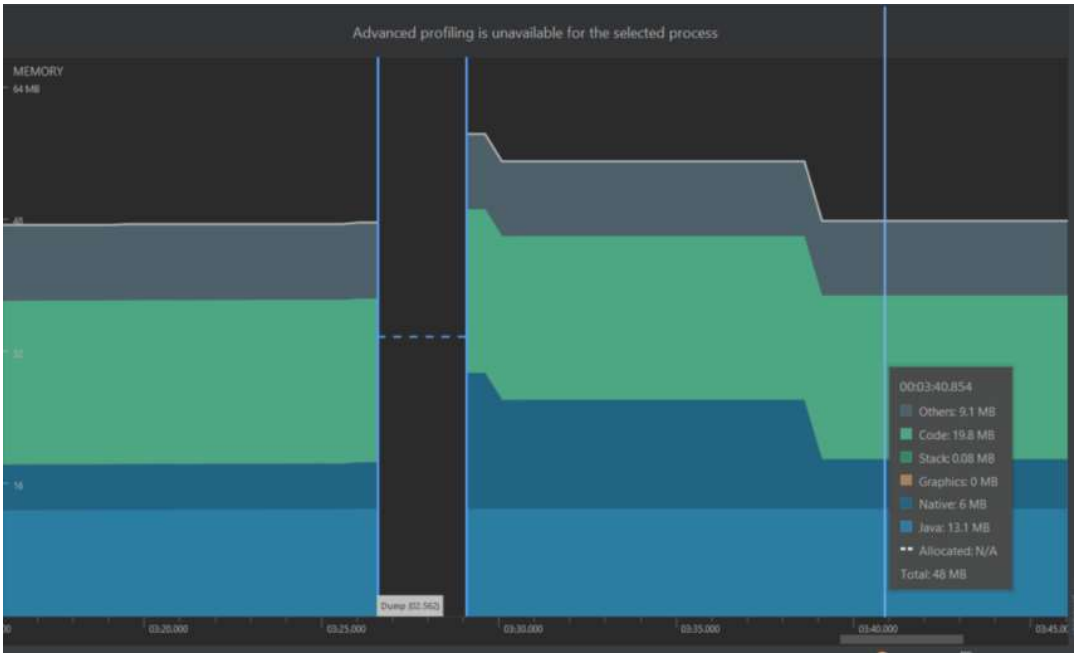



*Imagen 47. Captura de pantalla del uso de memoria mostrando el consumo después de pausar la reproducción de audio*

## 6.5. Análisis de los resultados obtenidos.

El objetivo de esta sección es analizar los datos resultantes durante la ejecución de los casos de uso mencionados en la sección anterior. Con los datos obtenidos se buscarán posibles fugas de memoria o fugas existentes, así como también, se buscarán los objetos que más espacio ocupan para ver la posibilidad de reducir el uso de la memoria, pues cuando se trata de la administración de memoria, la primera regla es minimizar siempre la cantidad de información que se almacena. Al reducir esto, es menos probable que se experimenten errores relacionados con la memoria y, con menos objetos en la memoria, se reciclan menos objetos, lo que conlleva a una recolección de basura más rápida. Las pérdidas de memoria son posibles en las aplicaciones cuando objetos se dejan innecesariamente en la memoria. Esto puede suceder debido a referencias accidentales u otros vínculos entre actividades que impiden que el garbage collector elimine el objeto.

Estas referencias accidentales pueden provocar problemas de falta de memoria en dispositivos con poca memoria, por lo que es fundamental se rastreen y resuelvan. Si se encuentran problemas de memoria en la aplicación, es posible que se tenga una fuga de memoria y se necesitaría investigar más para descubrir y eliminar la fuga de memoria a la que también se le conoce como “memory leak”.

Como primer paso, los archivos dump obtenidos y exportados desde Android Studio, se convirtieron a un formato estándar que Mat Tool pudiera entender pues Android tiene un formato especifico. La extensión de los archivos usado es la *. hprof y la conversión se hizo con la herramienta hprof-conv que es parte de las herramientas incluidas en el SDK de Android.

Mat Tool es capaz de generar diferentes reportes, el primer reporte generado fue el llamado “Leak Suspects” el cual como su nombre lo dice, reporta las sospechas de fugas en la memoria. En los 3 casos de uso descritos en la sección pasada se pudieron observar sospechas de fugas, la mayoría de estas sospechas están relacionadas a clases de tipos de datos utilizados en Java y es entendible puesto que son tipos muy comunes pero en el caso de cuando se inicia la reproducción de audio se logra ver una clase que consume 13.68% de la memoria y dado que el consumo de RAM es mayor en el caso donde se pausa la reproducción de audio, es posible que haya una clase

que no es des referenciada apropiadamente. La Imagen 48, Imagen 49 e Imagen 50 muestran los resultados del reporte leak suspects.

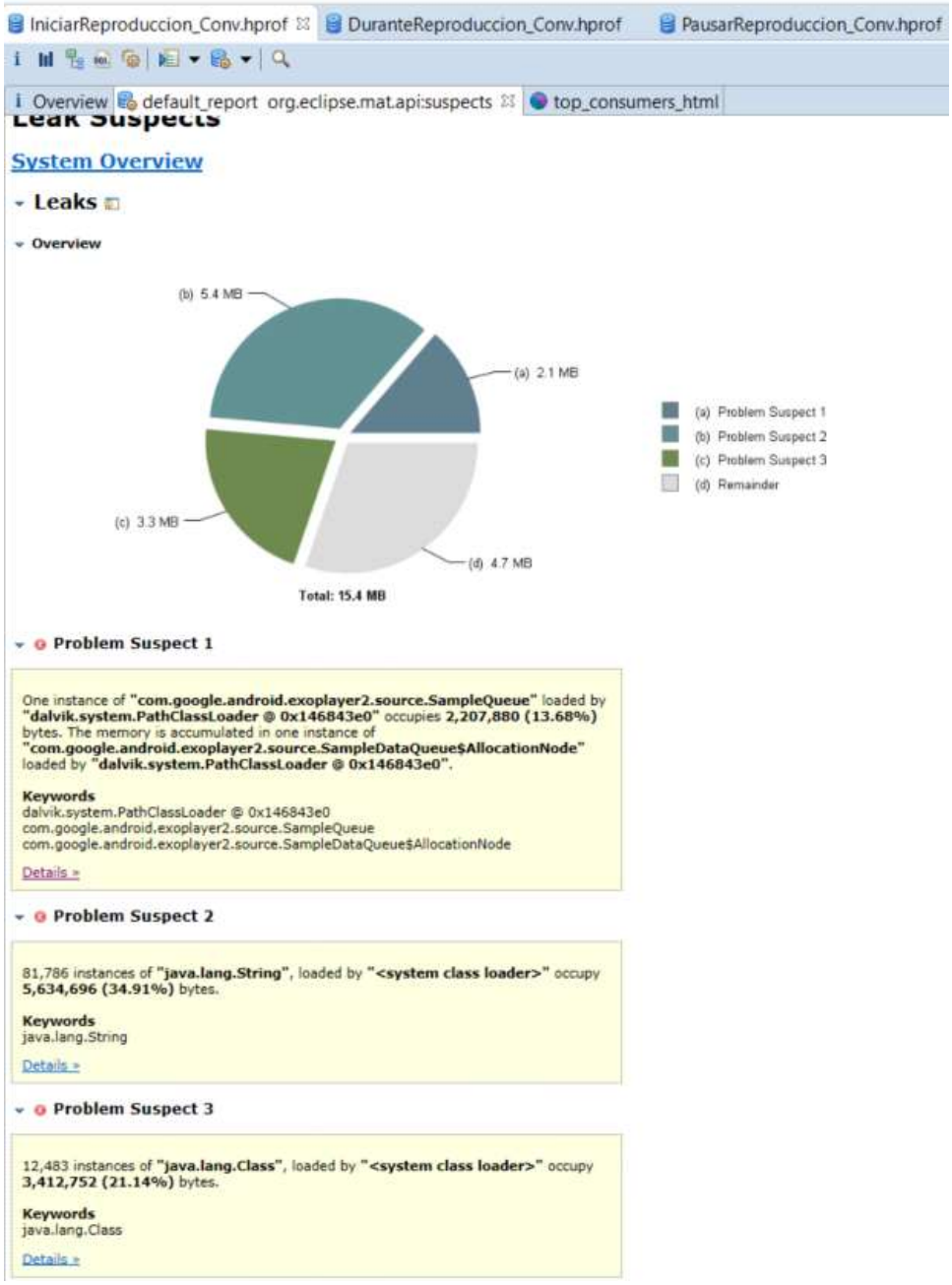



*Imagen 48. Reporte Leak Suspects del caso de uso cuando se inicia la reproducción de música*

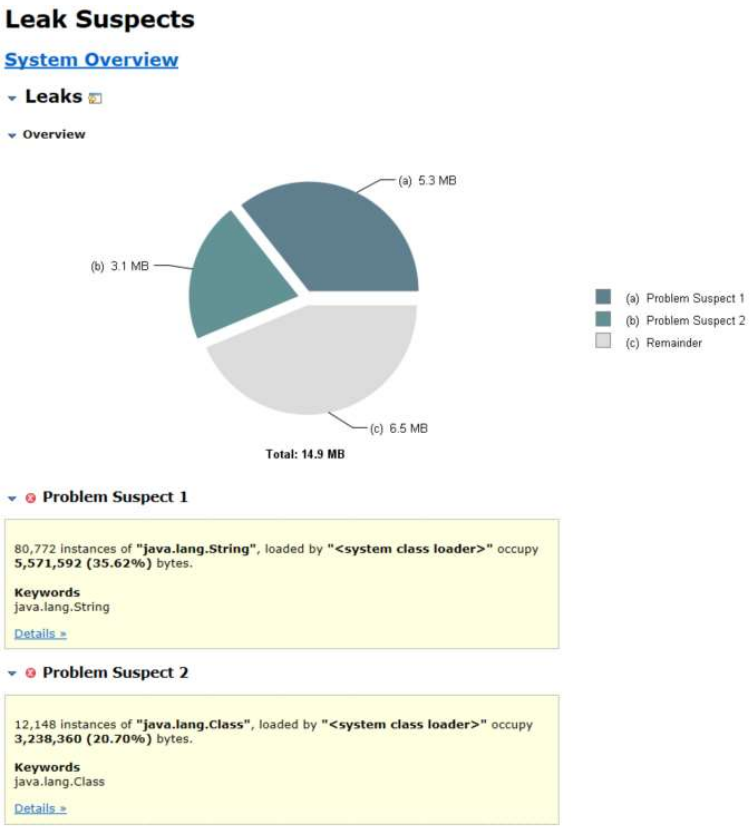



*Imagen 49. Reporte Leak Suspects del caso de uso durante la reproducción música*

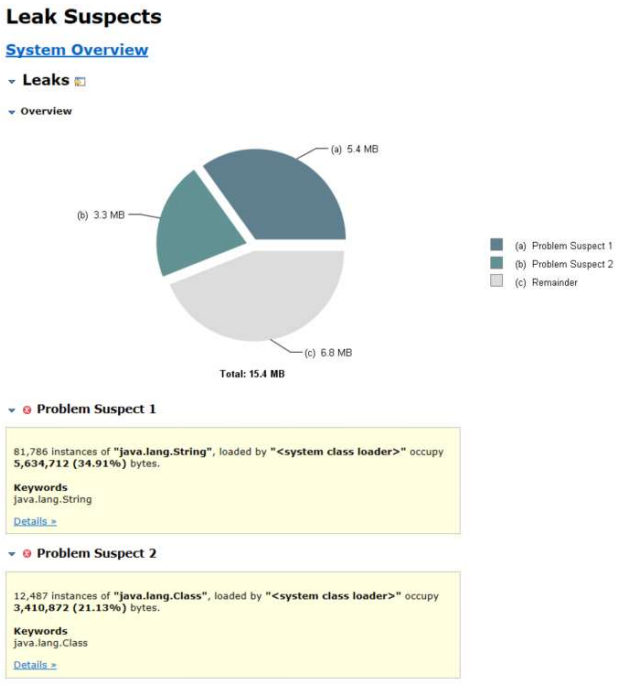



*Imagen 50. Reporte Leak Suspects del caso de uso cuando se pausa la reproducción de música*

En la Imagen 48 se puede observar una sospecha que hace referencia una clase que consume 13.68% del uso total, dicha clase se llama SampleQueue y la cual es parte del código de ExoPlayer. Esta es una clase que encola los archivos de multimedia, lo que podría explicar el espacio utilizado por esta clase cuando se inicia la reproducción de audio. Otro tipo de reporte llamado “Top Consumers” el cual, como su nombre lo indica, muestra los objetos que consumen más memoria y muestra la clase SampleQueue como la que tiene mayor uso de memoria en 2 de los 3 casos de uso. En la Imagen 51, Imagen 52 e Imagen 53 se muestran los “Top Consumers”.

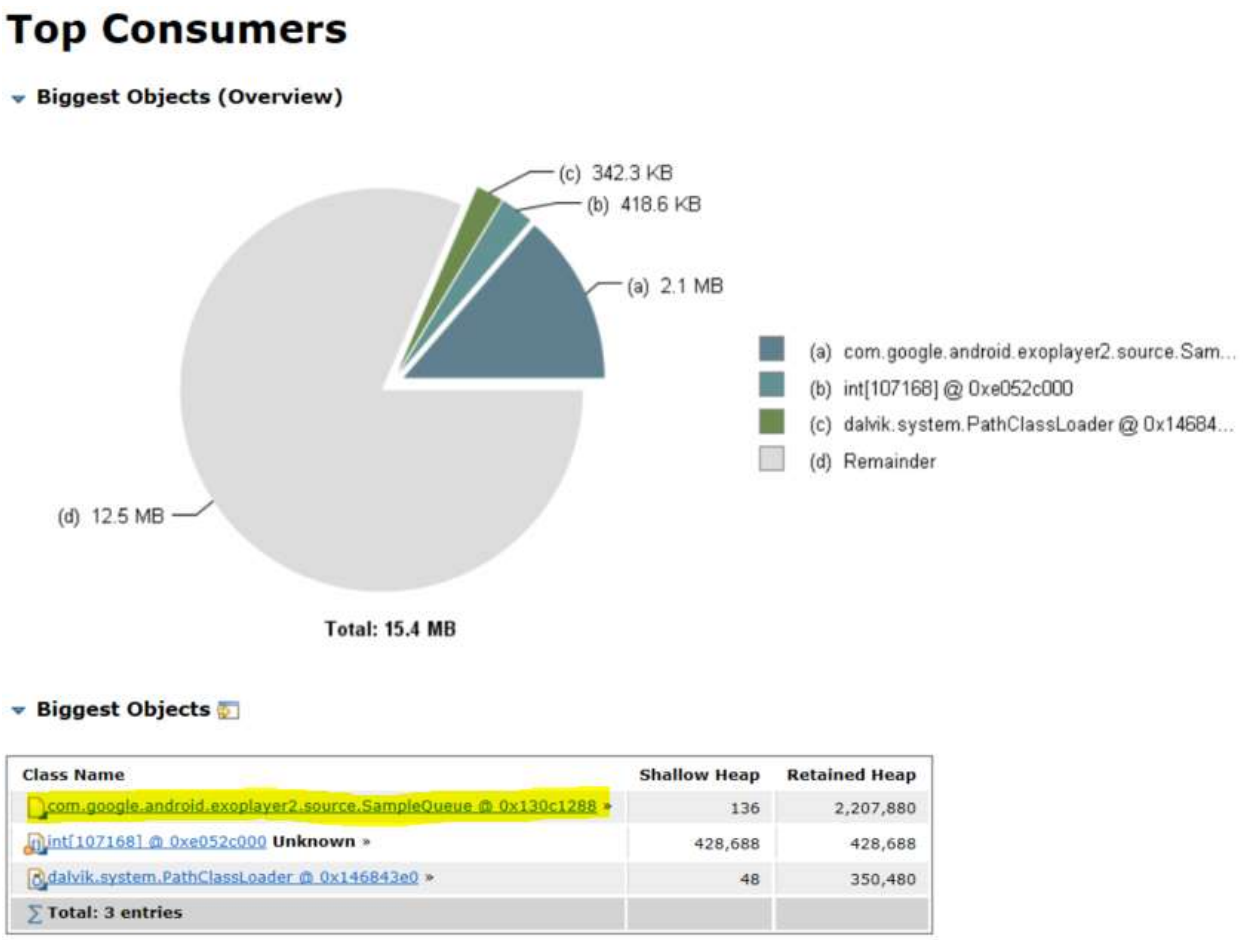
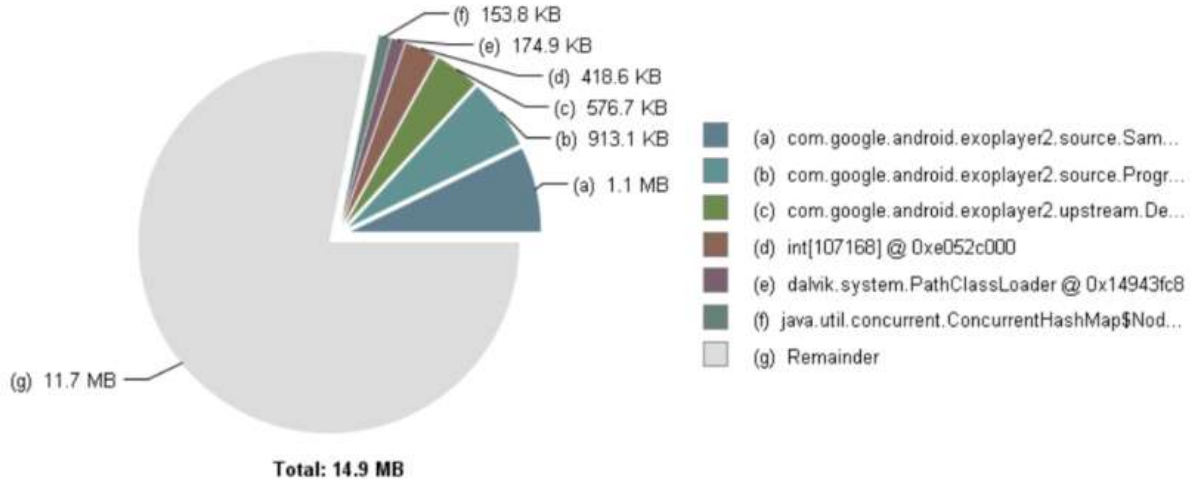



*Imagen 51. Top Consumers del caso de uso cuando se inicia la reproducción de música*

# Top Consumers

**Biggest Objects (Overview)**



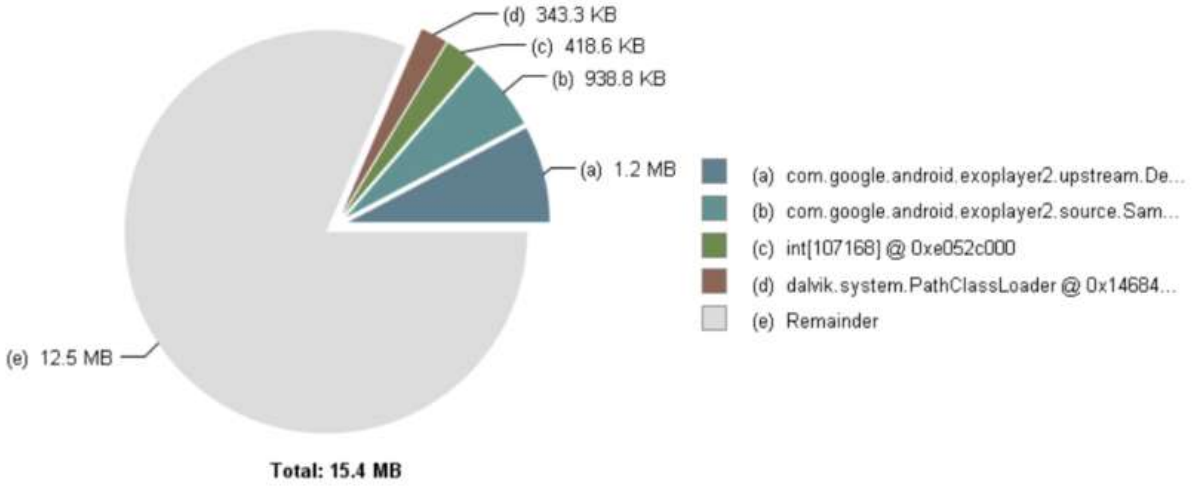

**Biggest Objects**

| Class Name | Shallow Heap | Retained Heap |
|---|---|---|
| com.google.android.exoplayer2.source.SampleQueue @ 0x12d401e0 » | 136 | 1,122,152 |
| com.google.android.exoplayer2.source.ProgressiveMediaPeriod @ 0x12fc0758 » | 152 | 935,040 |
| com.google.android.exoplayer2.upstream.DefaultAllocator @ 0x12e50fe8 » | 40 | 590,584 |
| int[107168] @ 0xe052c000 **Unknown** » | 428,688 | 428,688 |
| dalvik.system.PathClassLoader @ 0x14943fc8 » | 48 | 179,080 |
| java.util.concurrent.ConcurrentHashMap$Node[2048] @ 0x740dcfb8 **Unknown** » | 8,208 | 157,456 |
| **Total: 6 entries** | | |

*Imagen 52. Top Consumers del caso de uso durante la reproducción música*

## Top Consumers

**Biggest Objects (Overview)**




**Biggest Objects**

| Class Name | Shallow Heap | Retained Heap |
|---|---|---|
| com.google.android.exoplayer2.upstream.DefaultAllocator @ 0x146b06d0 » | 40 | 1,246,264 |
| com.google.android.exoplayer2.source.SampleQueue @ 0x130c1288 » | 136 | 961,328 |
| int[107168] @ 0xe052c000 **Unknown** » | 428,688 | 428,688 |
| dalvik.system.PathClassLoader @ 0x146843e0 » | 48 | 351,528 |
| **Total: 4 entries** | | |

*Imagen 53. Reporte Top Consumers del caso de uso cuando se pausa la reproducción de música*

En la industria de software automotriz, el uso de RAM tiende a ser el menos posible debido a que las memorias tienen un alto costo. En el caso muy específico de Info-entretenimiento las memorias suelen ser de mayor capacidad, puesto que la capacidad de procesamiento de audio, gráficos y otras características propias de esos sistemas consumen más RAM que otros sub-sistemas de un automóvil. En el caso muy específico de este proyecto, la tarjeta que se utilizó cuenta con una memoria DDR3 RAM de 2GB, lo cual es algo normal en sistemas de Info-entretenimiento. En el caso de la aplicación de media desarrollada el consumo de RAM total es de 15.4 MB en base a la Imagen 39, lo cual resulta en que la aplicación de media tiene un consumo de menos del 1% de la memoria total. Y hablando un poco al respecto del tiempo que tomó, éste se puede observar en los Apéndices, y comparado con el Epic-xxx40 de la Tabla 1, los desarrollos

de media fueron ligeramente más bajos en este proyecto pues el Epic-xxx40 se desarrolló en 2.9 meses y la aplicación de media en 2.25 meses.

# 7. Verificación y Validación

## 7.1. Metodología general de las pruebas y proceso de pruebas

El enfoque y objetivos generales de las pruebas tienen diferentes estados a medida que el proceso de desarrollo va arrojando resultados paulatinos debido a las salidas de las diferentes fases enmarcadas por los llamados “sprints”, bajo los esquemas de entregas de ciclo de desarrollo ágil. Si bien en la industria automotriz el imperante es el ciclo de desarrollo en V, este ciclo de desarrollo se muestra en la Imagen 54. Niveles de prueba en el ciclo V.

Para el propósito de estudio de este trabajo recepcional, la mayoría de los esfuerzos se llevan a cabo en la fase de sub-sistema y nivel de desarrollo de aplicación. Aquí valiendo la pena remarcar las exclusiones de desarrollo del sistema como el enfoque del desarrollo del hardware y mecánico como el gabinete donde está contenido.

Como se menciona, hay un desarrollo paulatino, el cual también va construyendo el sistema en diferentes niveles, que son un conjunto de actividades de pruebas gestionadas en el marco del nivel de derivación de requisitos que van cambiando en complejidad a medida que los requisitos de alto nivel provenientes de un cliente (ya sea interno o externo) hasta medidas. Aquí, el plan donde se administra define, detalla, gestiona y controla se llama plan de pruebas o “Test plan”, es decir, ningún aspecto del ciclo de prueba pasa inadvertido para este documento de planeación. Parte de esta gestión incluye ubicar en que fases y como se ubicarán los esfuerzos de prueba desde hacer una simple unidad de código hasta la simulación de un escenario real. Considerando la Imagen 55 los niveles de prueba se colocarían en la planeación dentro del plan de prueba. A continuación, se describen los niveles de pruebas en un esquema del ciclo en V la Imagen 54:

(1) *Nivel de Sistema.* Cuyo objetivo es probar los requisitos de la aplicación. Este nivel es el caso de estudio de este trabajo recepcional.

(2) *Nivel de Integración de HW-SW.* El principal objetivo son los casos de prueba de integración del Hardware o HAL con el sistema operativo. Este nivel esta fuera del alcance de este trabajo recepcional.
(3) Nivel de Disciplina. Serían los requisitos formales de una aplicación, específicos para Hardware o para Software. Este nivel esta fuera del alcance de este trabajo recepcional.
(4) Nivel de Unidad. Criterios cuantitativos de las funciones o métodos de SW (principalmente métricas de cobertura para pruebas unitarias). Este nivel esta fuera del alcance de este trabajo recepcional.

En el nivel de sistema, el cual es en el que se enfoca este estudio, habría una de las siguientes categorías que a la postre nos ayudaría a definir una métrica. Dentro de dichas secciones se pueden mencionar un número muy nutrido de propiedades, pero limitaremos ese número a las siguientes propiedades:

- ***Alcance de las pruebas***: Cuando alguna especificación de requisitos, hacia la cual hay trazabilidad a través del CDD de Android, define cuantos casos de prueba deberían ejecutarse. Por ejemplo, si es la primera ocasión en que el producto ejecuta las pruebas del CDD entonces es un alcance de pruebas completo o "*full test loop*". Si de ese conjunto de pruebas ya ejecutadas han tenido una actualización ya sea por parte del CDD de Android o los requisitos adaptados al producto, sólo se corre de forma parcial, lo cual mide lo que ha cambiado de una versión anterior a la nueva; se llama "delta" de pruebas. De dicha delta de requisitos hay dos vertientes: Las pruebas puntuales o "*retests*" y las pruebas regresivas "*regression tests*", las cuales a través de manera exploratoria y con base en algún modelo de arquitectura, ejecutan casos de prueba relacionados de manera indirecta a los cambios. Por ejemplo, si un caso de prueba relacionado a las lecturas de velocidad de un ECU de telemetría, en la iteración de pruebas más cercana a una liberación del SW se probará los casos de prueba para otros parámetros.

  Por último, pero no menos relevante sobre el tema de alcances de pruebas, están las pruebas primarias las cuales se ejecutan después de la liberación de alguna versión de SW para pruebas de caja negra. Hay dos conceptos en estados de prueba diferente: Las pruebas de

sanidad o “Sanity Test” y las pruebas exploratorias o “Smoke Test”. Los anteriores un tanto similares, pero la diferencia estriba que las pruebas exploratorias son de aceptación de una versión de SW y las pruebas de sanidad son las que se llevan a cabo antes de una fase de pruebas de regresión.

- **Asignaciones generales** como descripción general del “tester”, nivel de automatización (como de “tester” manual o “tester” automatizado), el calendario, y las técnicas de prueba. Estas son estrategias en los cuales se escribe un caso de prueba. Para este caso de estudio sólo se basa en aspectos funcionales para la cuestión automotriz, pero hay un universo mucho más extenso de especificaciones de prueba a considerar.

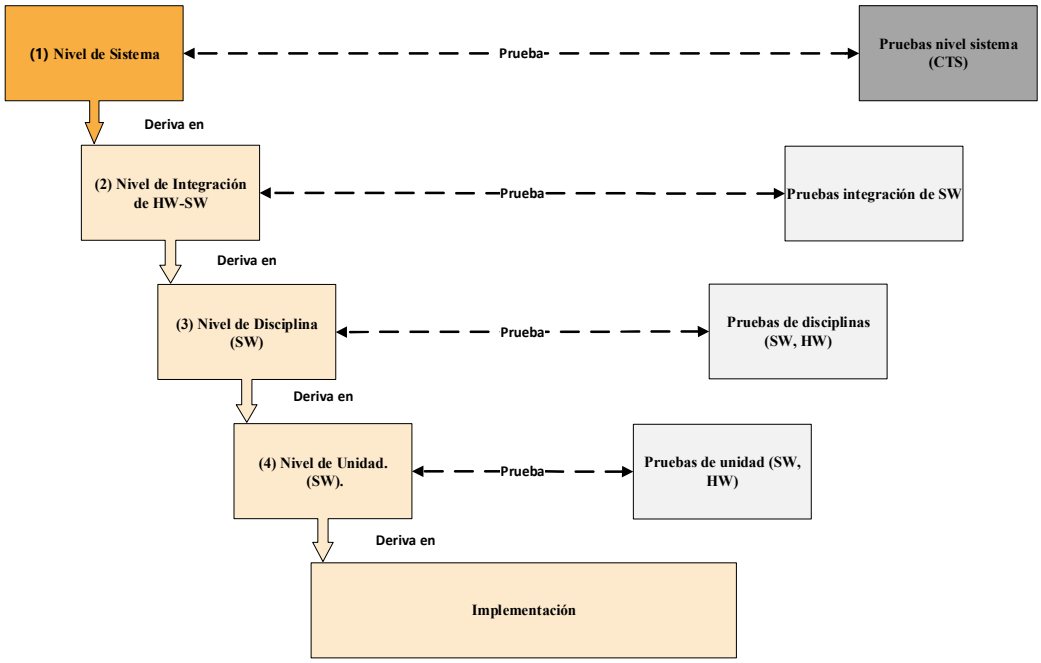



*Imagen 54. Niveles de prueba en el ciclo V. En donde la parte izq. Se aprecia la derivación de los requisitos a probar por los casos de prueba de la parte derecha.*

Si bien, la estructura de prueba para proyectos automotrices se describe en la Imagen 54, no se describe que es lo que ocurre con la parte de Android Automotriz. Los objetos de prueba se enmarcarían en cada uno de los bloques conceptuales del ciclo en V. En la sección 2.1 SAFe Essential se explica el manejo holístico del proyecto bajo el esquema SAFe ágil, cuya gestión de pruebas se ajusta conservando el aspecto de trazabilidad del modelo en V.

Es importante conocer ciertos conceptos para la correcta ubicación y manejo de recursos de SW, de HW y de pruebas en general que se enmarcan en el proceso tanto de desarrollo de sistemas automotrices en la industria, así como de Android Automotive. Por lo tanto, se integrarán ambas estructuras de prueba, siendo el "test framework" o estructura de pruebas de la industria automotriz en este estudio como se muestra en la Imagen 55, la cual muestra la "*Adaptación de la infraestructura de prueba tipo CTS*" de Android.

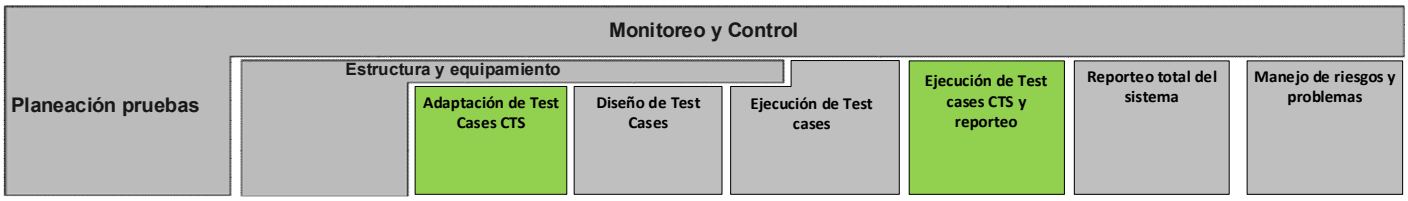



*Imagen 55. Proceso de pruebas incluyendo elementos del CTS en verde con la estructura de pruebas de la compañía Tier 1 en gris.*

## 7.2. Manejo de la especificación de requisitos o Documento de definición de compatibilidad de Android (CDD).

La empresa propietaria de todo el marco de desarrollo de Android Automotive, Google, ha especificado ciertos lineamientos, consideraciones y reglas a ser consideradas para que algún desarrollo pueda ser clasificado como un sistema, producto o dispositivo compatible o apegado con Android. Para ello cada ente que realice un producto que se jacte de ser compatible con Android significaría que siguieron las reglas referidas en el documento de definición de compatibilidad de Android, o por siglas en inglés "CDD" de "Android Compatibility Definition Document".

Cada versión nueva de Android está ligada a un CDD, y este nos otorgaría la política o "policy", cuyos objetivos son:

- Establecer políticas de codificación.
- Clarificar requisitos. Por ende, eliminar cualquier inconsistencia en los requisitos presentes.

Es decir, le dan al diseñador reglas para crear un producto apegado al mercado de Android tanto de aplicaciones como de su uso. Sin embargo, para desarrollar un producto basado en la arquitectura de Android, cualquier aspecto es relevante y debe considerarse. Y para estar seguro de que cada pequeña unidad desarrollada es compatible en cuanto al marco de la CDD, lo ideal es correr pruebas de forma frecuente y tener una estrategia de regresión adecuada. Esta es la manera que Google usa para que los desarrolladores estén apegados a las directrices de su solución.

## 7.3. Descripción de la especificación de prueba o CTS y configuración del banco de pruebas

Sabiendo la importancia del CDD, es necesario crear una aplicación compatible con las limitantes de los componentes de HW y las directrices creadas por el proveedor, Google. Para que todo funcione como se espera, la aplicación tiene que ser verificada. Para esto se tienen que ejecutar las pruebas dinámicas a través del Android "Compatibility Test Suite" o "CTS".

Al igual que con el CDD, las distribuciones de Android (incluida la versión automotriz) vendrá con un CTS especifico, en el cual podemos resaltar algunos componentes de acuerdo con la Imagen 56:

- **Secuenciadores de prueba**: Donde figuran el "TRADEFED" (Trade Federation), CTS (Compatibility Test Suite) y el VTS (Vendor test suite).
- **Casos de prueba o "Test cases":** En general, estas son las secuencias de prueba ejecutadas en el "Device under test (DUT)". En cuyo caso son por lo regular los JUnit test cases en Java y en paquetes de Android tipo de archivo *.apk,* de tal forma que pueden ser ejecutados en el DUT.
- **CTS Verifier:** La cual es una herramienta para pruebas manuales, conformada de la aplicación verificada ejecutada en el DUT y permite obtener los resultados de las pruebas.

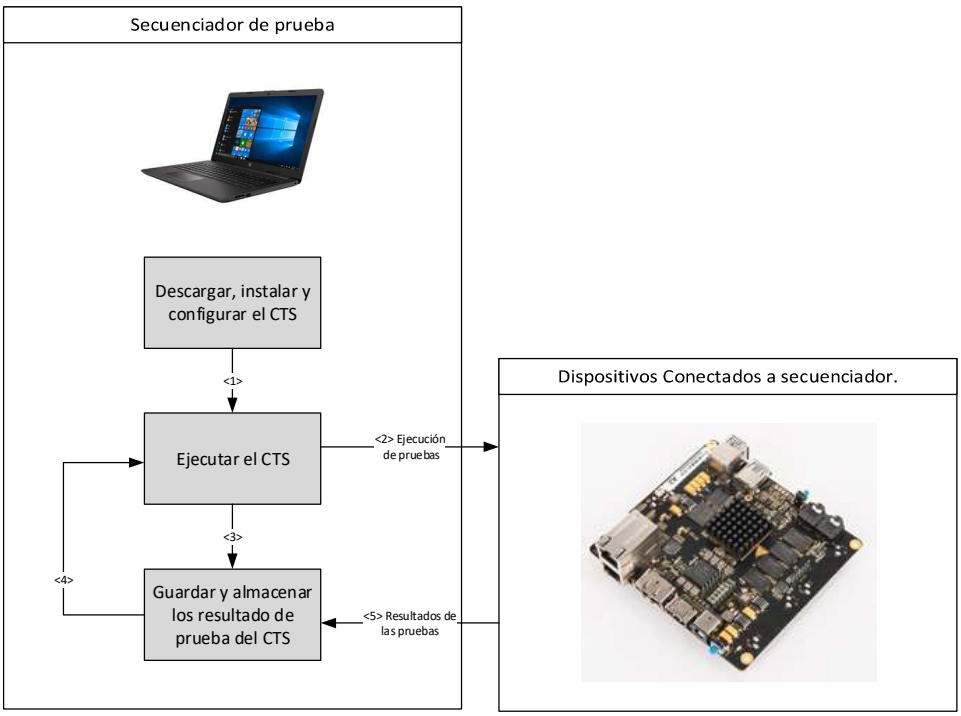



*Imagen 56. Los tipos de pruebas que provee el CTS son pruebas unitarias y pruebas funcionales.*

#### 7.3.1.1 Integration Test Level

Android "Integration Tests" puede comprender el "Testing" o pruebas "AOSP image porting", que no se abordarán en esta sección. Los resultados de la integración de la imagen y el arranque se muestran en la Imagen 9.

#### 7.3.1.2 SW Requirements Test Level

##### *7.3.1.2.1 Pruebas funcionales.*

En esta fase de pruebas se contemplan aspectos comportamentales que consisten en efectos a nivel usuario, llevado a cabo por el llamado y ejecución de las APIs.

##### *7.3.1.2.2 "Performance testing".*

Actualmente, este tipo de pruebas no se encuentra considerado en el CTS. El objetivo es poner bajo una demanda o llamado de servicios de forma periódica o espontánea y medir la experiencia del usuario en cuanto a satisfacción en la experiencia.

Aspectos de las pruebas manuales de performance que se pudieran considerar para un catálogo de pruebas más extenso:

- Percepción de desempeño.
- "Spinners: The Good and the Bad".
- "Animations to Mask Load Times".
- Obtener "White Lie" de actualizaciones instantáneas.
- Recomendaciones para mejorar "Perceived Performance".
- Medir el uso del CPU.
- Systrace para análisis de CPU.
- Traceview (Legacy Monitor DDMS y Android Studio).
- Y otras herramientas.

#### *7.3.1.2.3 Pruebas de robustez.*

En esta fase de pruebas se consideran aspectos de durabilidad bajo condiciones de estrés de usuarios por una demanda o llamada constantes de servicios o APIs.

### 7.3.2 Configuración de herramientas y banco de pruebas para ejecución automática del CTS.

Por ahora CTS sólo es soportado en Ubuntu. Para este caso de estudio se usará la versión de Ubuntu 20.04 LTS. Sin embargo, antes de ejecutar un determinado CTS, hay que estar seguros de que se tienen las versiones recientes del llamado "Android Debug Bridge" (adb) y el "Android Asset Packaging Tool" (aapt) llamando comandos en la terminal para conocer el estado de dichas herramientas.

#### *7.3.2.1.1 ADB (Android Debug Bridge)*

El "adb" es una potente herramienta, de donde se identifican dos casos cruciales en el uso de estas dentro de Android:

- El adb ejecutándose en un dispositivo.
- El adb ejecutándose en una PC.

Por otra parte, el "*adb*" se comunica con cualquier dispositivo Android sobre una conexión tipo USB. Esto, en una arquitectura de cliente/servidor donde se tiene la opción de tener varios servidores, teniendo el cliente comandos adecuados para poderse comunicar con varios dispositivos a la vez sin causar un conflicto entre ellos. La arquitectura de la red de la comunicación se ve como en la Imagen 57.

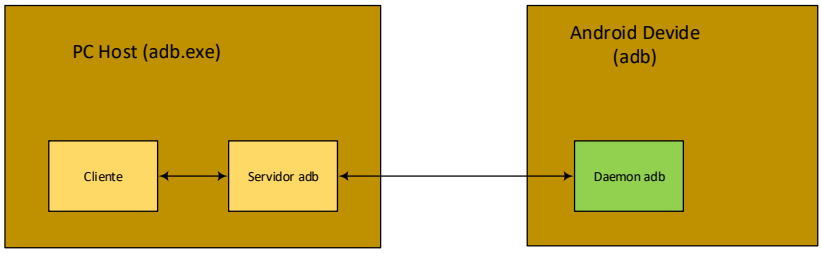



*Imagen 57. Arquitectura de comunicación con "adb".*

- **Servidor adb:** Es una aplicación que corre en segundo plano del sistema host y mantiene la comunicación con el dispositivo de forma latente.
- **Cliente:** Aplicación que corre en la terminal sobre la herramienta *Dalvik Debug Monitor Server (DDMS).*
- **Daemon adb:** Una aplicación que se ejecuta en el dispositivo en cuestión/o emulador.

#### 7.3.2.1.1.1 Características internas del adb.

El código fuente se encuentra en system/core/adb dentro de los siguientes archivos en el proyecto: adb.c, fdevent.c, transort.c, transport_local.c, tansport_usb.c, service.c, sockets.c, y util.c.

El comando en la terminal para invocar la versión de "adb" es:

```
$adb version
```

En la Imagen 58 se tiene la respuesta de la versión del "adb" en la consola de la terminal.


*Imagen 58.Versión de la "adb" instalada.*

Al habilitar las opciones de desarrollador un dispositivo moderno, este desplegará la notificación con el ID del dispositivo conectado a la estación de prueba como se muestra en la Imagen 59:

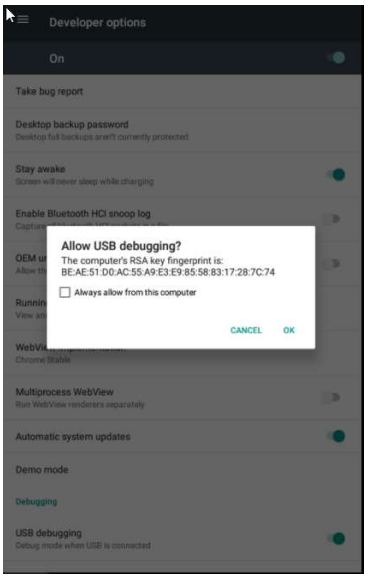



*Imagen 59. Ejemplo de RSA generado una vez conectando el dispositivo.*

Ahora haciendo uso del "*adb"* a través del comando "devices" en la terminal, se tiene su respectiva respuesta mostrada en la Imagen 60.

```
$adb devices -l
```


*Imagen 60. Aquí se tiene el ID del dispositivo junto con una descripción de la Tablet de ejemplo.*

De manera análoga, se ejecuta el comando para "aapt v" para su ejecución.

```
$aapt v
```

En la captura que corresponde a la Imagen 61 se tiene la respuesta de la versión del "aapt".


*Imagen 61. Versión de la aapt instalada.*

Ahora se ejecutará el CTS en un dispositivo ya en el mercado donde se asume se ha verificado y validado. En el caso de una prueba con una Tablet, donde se tiene en el instructivo del fabricante (Instructivo [28]) donde se tiene una **Tablet Android 7.0 Nougat,** es decir, un producto ya probado y cuyo objetivo es hacer una prueba de exploratoria del banco de pruebas descargado del CTS para demostrar cierta confiabilidad antes de ejecutar pruebas en el prototipo bajo estudio.

Dado que se tiene una versión ya en el mercado, se asume que se corrieron las pruebas para dicho dispositivo.

```
$export PATH=$PATH:/home/eduardo/Android/Sdk/build-tools/27.0.3
```

```
$java -version
```

Después se revisa la versión de Java ya instalada en el "tester" (PC) como se muestra en la Imagen 62


*Imagen 62. Respuestas para conocer versiones de Java disponible.*

En dado caso que no se tenga la versión instalada o que el SO apunte hacia otra instalación, el intento de ejecutar el CTS tendrá como respuesta lo que se muestra en la Imagen 63:


*Imagen 63. Respuesta en terminal a versión incorrecta de Java.*

Ahora, en un equipo o "tester" automático pudieran convivir diferentes versiones de Java. Todas las versiones existentes disponibles se pueden mostrar en la terminal ejecutando el comando de la lista de versiones como se muestra en la Imagen 64.

```
$sudo update-java-alternatives --list
```

```
(base) eduardo@eduardo-Inspiron-5758:~/Android/Sdk/cts/android-cts-7.0_r33-linux_x86-arm/android-cts/tools$ sudo update-java-alternatives --list
[sudo] password for eduardo:
java-1.11.0-openjdk-amd64      1111       /usr/lib/jvm/java-1.11.0-openjdk-amd64
java-1.8.0-openjdk-amd64       1081       /usr/lib/jvm/java-1.8.0-openjdk-amd64
```

*Imagen 64. Lista de versiones de Java disponibles.*

En este ejemplo, ya se tiene instalada la versión deseada para correr los casos de prueba de la versión Android 7.0 Nougat, A continuación, se tiene la ejecución en terminal en la Imagen 65.

```
$sudo update-alternatives --config java
```

```
(base) eduardo@eduardo-Inspiron-5758:~/Android/Sdk/cts/android-cts-7.0_r33-linux_x86-arm/android-cts/tools$ sudo update-java-alternatives --list
[sudo] password for eduardo:
java-1.11.0-openjdk-amd64      1111       /usr/lib/jvm/java-1.11.0-openjdk-amd64
java-1.8.0-openjdk-amd64       1081       /usr/lib/jvm/java-1.8.0-openjdk-amd64
(base) eduardo@eduardo-Inspiron-5758:~/Android/Sdk/cts/android-cts-7.0_r33-linux_x86-arm/android-cts/tools$ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Priority   Status
------------------------------------------------------------
* 0            /usr/lib/jvm/java-11-openjdk-amd64/bin/java      1111      auto mode
  1            /usr/lib/jvm/java-11-openjdk-amd64/bin/java      1111      manual mode
  2            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      manual mode

Press <enter> to keep the current choice[*], or type selection number: 2
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java to provide /usr/bin/java (java) in manual mode
(base) eduardo@eduardo-Inspiron-5758:~/Android/Sdk/cts/android-cts-7.0_r33-linux_x86-arm/android-cts/tools$ sudo update-java-alternatives --list
java-1.11.0-openjdk-amd64      1111       /usr/lib/jvm/java-1.11.0-openjdk-amd64
java-1.8.0-openjdk-amd64       1081       /usr/lib/jvm/java-1.8.0-openjdk-amd64
(base) eduardo@eduardo-Inspiron-5758:~/Android/Sdk/cts/android-cts-7.0_r33-linux_x86-arm/android-cts/tools$ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Priority   Status
------------------------------------------------------------
  0            /usr/lib/jvm/java-11-openjdk-amd64/bin/java      1111      auto mode
  1            /usr/lib/jvm/java-11-openjdk-amd64/bin/java      1111      manual mode
* 2            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java   1081      manual mode
```

*Imagen 65. Selector de versión predeterminada de Java.*

### *7.3.2.1.2 Android ABI*

Es necesario conocer el tipo de ABI de Android usada para el microcontrolador del DUT. En el caso de un dispositivo basado en el microcontrolador MediaTek MT8321, se ejecuta la siguiente instrucción, con la correspondiente salida en la terminal como en la Imagen 66 ejecutando el comando del "*adb Shell*" en la terminal.

```
$sudo adb shell getprop ro.product.cpu.abi
```

Teniendo la siguiente respuesta en la terminal de acuerdo con la Imagen 66.


*Imagen 66. Tipo de abi soportado para el procesador del DUT.*

Para el caso del ABI armeabi-v7a, este se encuentra descrito en las definiciones de procesadores ARM soportadas, como se muestra en la Tabla 5.

| Tipo de ABI | Set de instrucciones soportadas | Notas. |
|---|---|---|
| armeabi-v7a | armeabi<br>Thumb-2<br>VFPv3-D16 | Incompatible with ARMv5/v6 devices. |
| arm64-v8a | AArch64 | |
| x86 | x86 (IA-32)<br>MMX<br>SSE/2/3<br>SSSE3 | No support for MOVBE or SSE4. |
| x86_64 | x86-64<br>MMX<br>SSE/2/3<br>SSSE3<br>SSE4.1, 4.2<br>POPCNT | |

*Tabla 5.Parametros ABI soportados para algunas gamas de producto.*

#### 7.3.2.1.2.1 Configuración del DUT de prueba.

- Los dispositivos que no poseen una interfaz gráfica deben estar conectados a una interfaz gráfica.
- Al igual que el dispositivo "TI Beagle board", si el DUT posee ranuras para tarjetas SD, estas deben contar con tarjetas SD clase 10. Aquí, sólo cabe aclarar que durante la ejecución de la prueba, el contenido de estas pueda ser borrado.
- De manera similar a las ranuras de las tarjetas SD, si las tarjetas de desarrollo o sistemas como tabletas o celulares poseen ranuras para tarjetas SIM, estas deben estar montadas.

Al conectar el dispositivo a la PC, este desplegará una notificación, se precederá a instalar ciertas aplicaciones de soporte a la ejecución de casos de prueba.

```
$adb shell getprop ro.build.version.release
```

El “adb” sirve para confirmar el dato de la versión y de la revisión con los comandos API del SDK de la con los comandos de terminal siguientes en la Imagen 67.

```
$adb shell getprop ro.build.version.sdk
```


*Imagen 67. Respuestas de versión de reléase y SDK.*

Esto confirma la versión de Android y nos da la versión del API 24, estas pueden ser usadas para descargar los archivos correspondientes.

#### 7.3.2.1.2.2 Configuración de DUT (Beagle board-x15)

De manera similar a la ejecutada con el proyecto de ejemplo, se seguirá el mismo procedimiento para el Beagle board-x15 como se muestra en la Imagen 68.

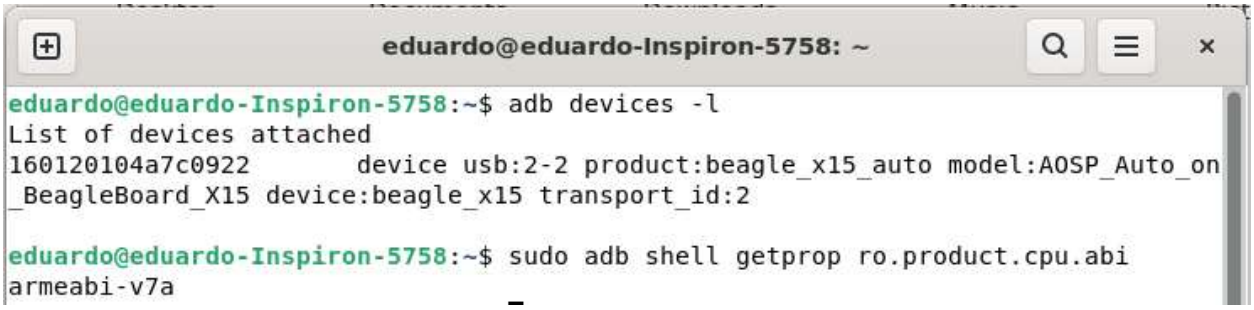



*Imagen 68. Reconocimiento de dispositivo conectado y lectura del abi.*

En dado caso que no se conociera la versión del código en la cual se basó el AOSP, el “adb” serviría para confirmar la versión y de la revisión usando los comandos API del SDK, mostrados en la Imagen 69.

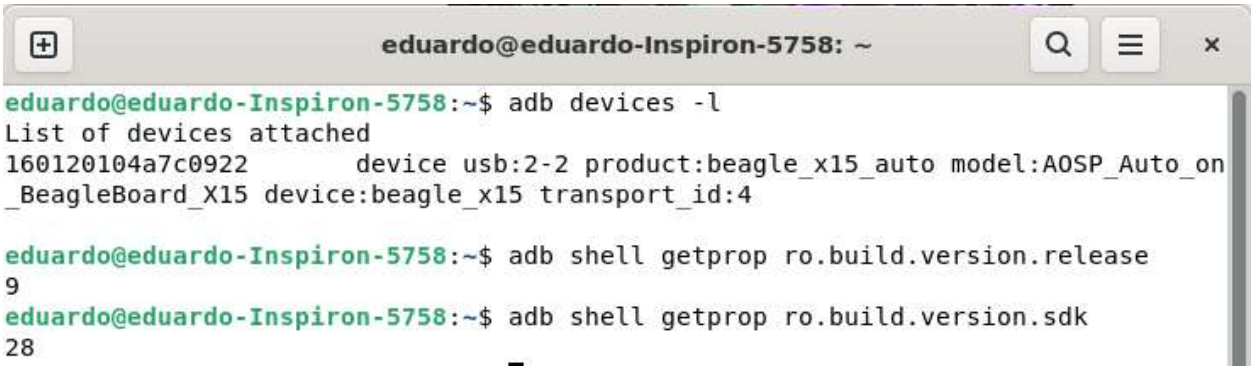



*Imagen 69. Confirmación de la versión de Android del Beagle board.*

Por ende, el tipo y versión de archivos que se debieran descargar del repositorio del CTS llamado "Compatibility Test Suite downloads" [29] son los que consideran para la opción de Android 9 como se muestra en la Imagen 70.

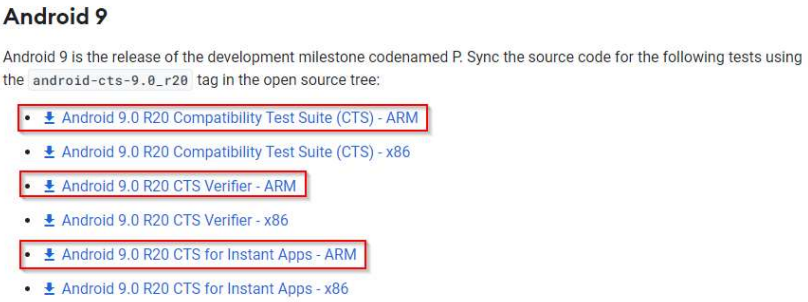



*Imagen 70. Conjunto de archivos seleccionados para chips basados en ARM.*

## 7.4. Ejecución de pruebas y configuración del secuenciador de pruebas CTS o "Compatibility Test Suite".

Antes de empezar se requieren ciertos requisitos para poder ejecutar el secuenciador CTS. Los requisitos son los siguientes:

- Una computadora ejecutando un ambiente de Linux.
- ABD funcional
- Android SDK.
- Java SDK 6 o superior.
- Android CTS.
- Android CTS Media.

#### 7.4.1.1 Pasos de la configuración de “Compatibility Test Suite”.

Podemos tener en cuenta los siguientes 3 principales pasos:

1. Selección y descarga de los archivos necesarios para instalar el CTS.
2. Configuración de la estación de pruebas.
3. Configuración de los dispositivos tipo Android.

##### 7.4.1.1.1 Selección y descarga de los archivos necesarios para instalar el CTS.

1. Selección de los archivos y paquetes necesarios. En este caso se selecciona la instalación del Android SDK con los paquetes necesarios. Siendo el emulador y las descargas de la versión de CTS necesarias para esta aplicación. En este caso, se tendrá la versión Android 9.0 API Level 28. Sin embargo, a diferencia del emulador, esta sólo se podrá correr en la tarjeta de desarrollo usando la imagen del sistema e interfaces de la aplicación para procesadores ARM como se muestra en Imagen 71. Estos archivos van en directorio de trabajo.

2. Después, se descargan lo archivos de CTS para pruebas de “media”.

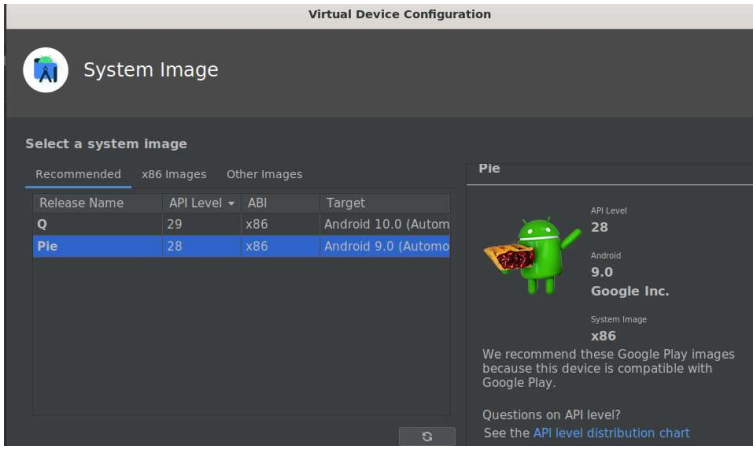



*Imagen 71. Datos para poder ser usados al momento de seleccionar las plantillas de prueba.*

#### 7.4.1.1.2 Configuración del dispositivo bajo prueba.

En este trabajo utilizamos la plataforma Beagle de TI, la cual es de HW abierto, cuyo procesador es basado en la arquitectura ARM (armeabi-v7a), ejecuta Android Automotive. En este caso se tiene un prototipo y por lo tanto no se tiene un dispositivo de producción disponible en el mercado.

Ahora, el banco de prueba tendrá el siguiente concepto inicial basado en el sub-proceso de equipamiento *(*ver la Imagen 55*)*. La unidad de equipo mínima para pruebas de sistemas es la mostrada y sugerida por el fabricante de la plataforma de prototipo en el "1.1. Getting Started Guide [30]". La Imagen 72 muestra cada una de sus partes para posteriormente definir cada uno de los elementos.

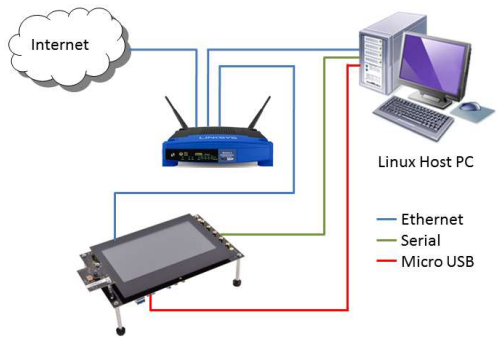



*Imagen 72. Banco de prueba mínimo para pruebas de sistema de la beagle board.*

1) **Linux Host:** El host puede ser cualquier PC capaz de tener una terminal para ejecutar los comandos en Linux. Dando pie a un punto importante, que esta PC puede ser un industrial o una máquina virtual, esto es lo más común en el ambiente automotriz. Las interfaces son Wi-Fi, micro USB a la tarjeta prototipo; y si se necesitara hacer cierto tipo de depuración se usuaria el serial basado en UART.
2) **Beagle board:** El DUT de este trabajo se puede consultar en la sección "4.1.2".

#### 7.4.1.1.3 Puesta a punto del sistema Beagle y definición de estados de la aplicación del sistema.

Ahora una vez formateada la unidad, se procede a conocer los estados del sistema a través de pruebas exploratorias, esas que se ejecutan antes de pruebas más completas. Y de igual manera se denominan los diferentes estados y sub-estados del sistema para cada opción, si se presentará un “bug” o error ya sea funcional o de calidad de SW.

A continuación, se lista cada uno de los estados.

- **Estado 0 (cero):** Es el estado más básico del sistema previo en la inicialización, es meramente el sistema sin suministro eléctrico en el dispositivo como se muestra en la Imagen 73. Se notan el “1) Linux Host” y el “2) Beagle board” de acuerdo con la sección “7.4.1.3 Configuración del dispositivo bajo prueba.”

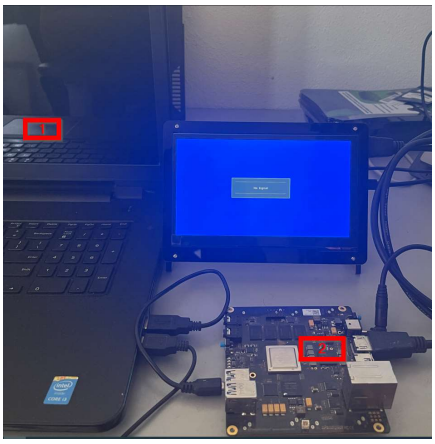


*Imagen 73. Estado cero (0) antes de inicializar el sistema.*

- **Estado 1:** La inicialización del sistema es la carga de 3 sub-estados. Consisten en la carga visual de 3 animaciones. Sin embargo, cada “animación” involucra un sub-proceso diferente. Por lo tanto, hay 3 sub-estados mostrados en la Imagen 74, Imagen 75 y la Imagen 76.

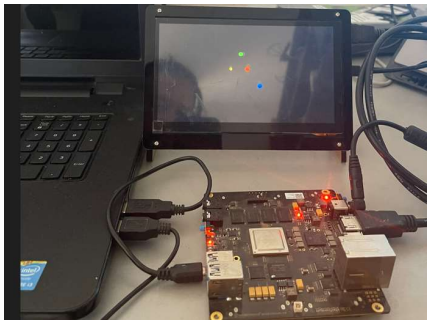


*Imagen 74. Primer sub-estado 1 con la animación previa al despliegue de la marca del proveedor. Primera carga después del arranque.*

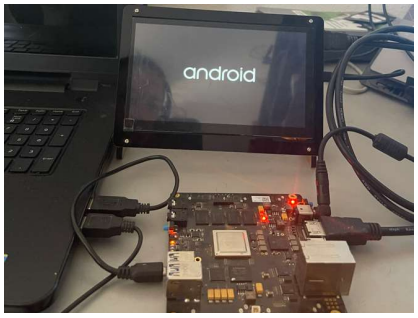



*Imagen 75. Sub-estado 2 con la animación de la marca del proveedor.*

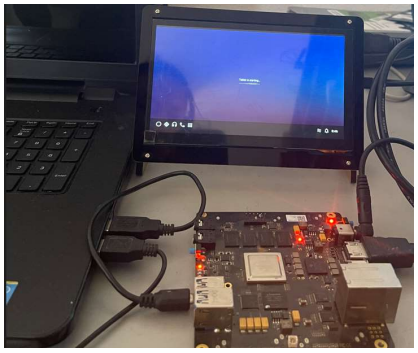


*Imagen 76. Sub-estado 3 con la animación de la carga de la aplicación. Nótese la leyenda de "Tablet is starting".*

- **Estado 2**: El estado que se denomina principal, ya que no se despliega información de algún sub-menu o posterior sub-estado. Este es el programa predeterminado inmediato a la inicialización de la aplicación del sistema. Se muestra en la Imagen 77.

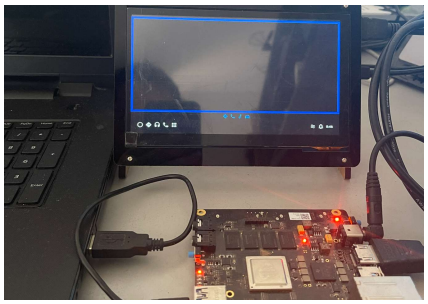


*Imagen 77. Estado principal del sistema posterior a su inicialización.*

Ahora, nótese la presencia del apuntador nativo en la aplicación principal, esta puede apreciarse después que se conecta al sistema un dispositivo adicional como se muestra en la Imagen 72. Puede ser reemplazado por un ratón convencional, pero también puede usarse otro dispositivo como un recurso tipo "touch-screen".

En la Imagen 78 se muestra el puntero del periférico numerado como opción 1 y se dio de alta de forma "plug-and-play" mostrándolo en la pantalla.

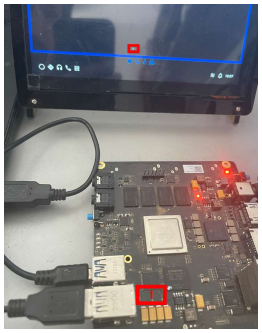


*Imagen 78. Conexión de un recurso para señalar y seleccionar opciones dentro de la aplicación marcada como "**1**".*

- **Estado 3**: Es el que lleva del estado del principal hacia aquel que muestra aplicaciones para el usuario final por parte del desarrollador. Hay 3 fases reconocibles o *sub-estados* en las imágenes y son: *Sub-estado 3a* o selección del menú de la Imagen 79, *sub-estado 3b* o cambio del menú de la Imagen 80, *sub-estado 3c* o regreso hacia el menú principal o salida del menú de la Imagen 81.

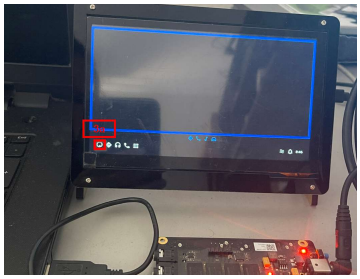


*Imagen 79. El sub-estado 3a es la selección del menú.*

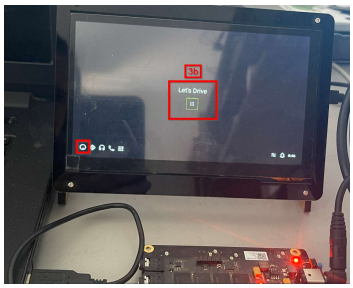



*Imagen 80. El sub-estado 3b es la transición efectiva en el estado donde se muestran aplicaciones desarrolladas.*

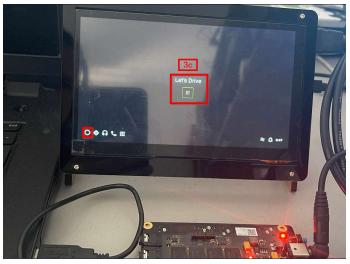



*Imagen 81. El sub-estado 3c es la transición efectiva en el estado donde se muestran aplicaciones desarrolladas hacia el menu principal. Notese el detalle que no regresa al estado principla efectivamente dando lugar a una anomalía.*

- **Estado 4**: Es el que lleva del estado del principal hacia aquel que muestra aplicaciones para el usuario final por parte del desarrollador. Aquí hay un punto diferente a la aplicación principal, ya que en el *estado 3c* regresa hacia la aplicación principal del *sub-estado 3a*. Hay 3 fases reconocibles o *sub-estados* para esta región de aplicaciones llamado “mapa” que se muestran en la Imagen 82, Imagen 83 e Imagen 84.

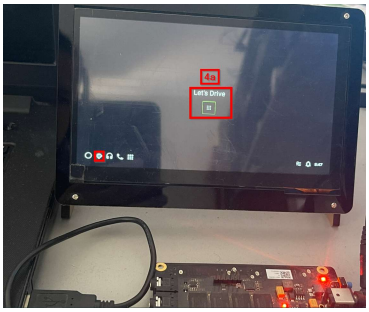



*Imagen 82. El sub-estado 4a es la selección del menú llamado “mapa”.*

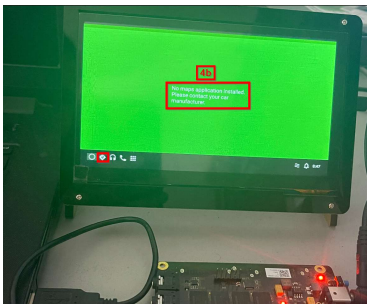



*Imagen 83. El sub-estado 4b es la transición efectiva hacia el menú llamado "mapa". Nótese la ausencia de una aplicación integrada a esta aplicación bajo estudio.*

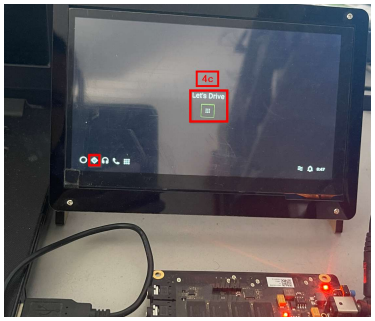



*Imagen 84. El sub-estado 4c es la transición efectiva en el estado donde se muestran los "mapas" hacia el menú principal.*

- **Estado 5**: Lleva del estado principal al que muestra aplicaciones por parte del desarrollador. Hay 3 fases reconocibles o sub-estados para esta región de aplicaciones principales llamado "audio" que van como: Sub-estado *5a* o selección del menú de la Imagen 85, sub-estado 5b o cambio del menú de la Imagen 86, sub-estado *5c* o regreso hacia el menú principal o salida del menú de la Imagen 87.

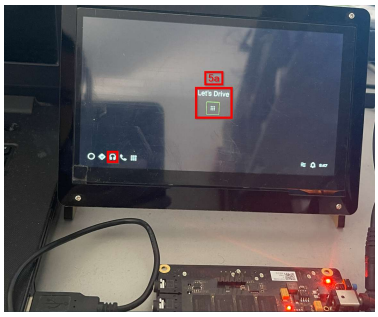



*Imagen 85.El sub-estado 5a es la selección del menú llamado "Audio".*

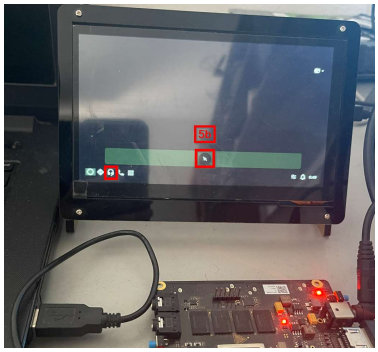



*Imagen 86. El sub-estado 5b es la transición efectiva hacia el menú llamado "Audio". Nótese la ausencia de una aplicación integrada a esta aplicación bajo estudio.*

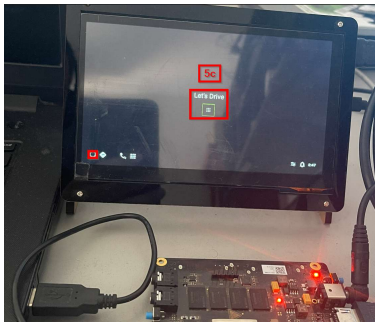



*Imagen 87. El sub-estado 5c es la transición efectiva en el estado donde se muestran los "Audio" hacia el menú principal.*

En general, la aplicación da muchas oportunidades para analizar varias transiciones y estados. Eso, en un contexto de ingeniería de requisitos da pie a usar herramientas, estrategias y un producto final en una especificación de requisitos. Una vez hecho el paréntesis de los requisitos, la mayoría de la ejecución de los casos de prueba se haría en alguno de los estados mencionados como 3, 4 y 5 aunque pudiera ser en más escenarios.

## 7.5. Reporte de Pruebas de Android CTS.

### 7.5.1 Descripción y uso de los “test cases” predefinidos de CTS en el DUT.

Una vez configurados el DUT y el dispositivo que ejecutará las pruebas, se puede usar el “tradefed” o “cts-tradefed” para ejecutar algunos casos de prueba. Se conecta en el DUT como se muestra en la Imagen 88. Usando esta transacción el DUT se identifica como conectado y se prepara para ejecutar un conjunto de pruebas.

```
$./cts-tradefed
```

Al ejecutar el comando, se reconoce la conexión con el dispositivo conectado como se muestra en la salida de la terminal de acuerdo con la Imagen 88.

```
(base) eduardo@eduardo-Inspiron-5758:~/Android/Sdk/cts/android-cts-7.0_r33-linux_x86-arm/android-cts/tools$ ./cts-tradefed
Android Compatibility Test Suite 7.0_r33 (5932566)
11-28 16:51:02 I/DeviceManager: Detected new device J3GR2101001915
```

*Imagen 88. Conexión del DUT.*

En el ejemplo, para ejecutar el script de las pruebas, se tiene que dirigir a la ubicación donde se descargaron los archivos de Android y los resultados se muestran en la Imagen 89.

```
cts-tf > list results
Session  Pass  Fail  Modules Complete  Result Directory     Test Plan  Device serial(s)  Build ID         Product
0        2443  30    116 of 466        2018.11.19_15.03.09  cts        742ce050          OPM1.171019.011  msm8937_64
1        3638  37    132 of 466        2018.11.19_18.13.43  cts-retry  742ce050          OPM1.171019.011  msm8937_64
2        3638  37    132 of 466        2018.11.20_11.15.36  cts        742ce050          OPM1.171019.011  msm8937_64
cts-tf > run cts --retry 2
```

*Imagen 89. Resultado en la terminal.*

Una vez ejecutado y reconocido el DUT, se podrán ejecutar las pruebas de una suite. La consola de pruebas CTS provee algunos comandos útiles.

Por otra parte, se muestra el uso de los módulos o especificaciones de los casos de prueba para la versión 9 en la Imagen 70. Para dicha versión se tienen alrededor de 322 especificaciones de prueba. Para correr todas se ejecuta el comando en la terminal de acuerdo con como se muestra la Imagen 90. Hay que tener en cuenta que tampoco todas las especificaciones de prueba pudieran aplicar a este sistema ya que no cuenta con periféricos como Bluetooth, cámaras o varios tipos de sensores.

```
cts-tf > run cts
```


*Imagen 90. Comando para correr todas las especificaciones de prueba.*

Sin embargo, pareciera que hubo casos de prueba que durante su ejecución provocaron un reset en el DUT. Al tener esto, es necesario reiniciar la sesión como se muestra en la Imagen 88. Después, habría que indagar que ocurrió en las sesiones de prueba anteriores y aprender a leer los "test logs". En los ciclos de pruebas esto es algo recurrente. Por ejemplo, una vez terminada la ejecución se puede revisar la última sesión de pruebas con el comando siguiente y como se muestra en la ejecución de la Imagen 91.

```
cts-tf > list results
```


*Imagen 91. Informacion general de las ejecuciones de prueba de los cuales se han almacenado test logs.*

Para lo anterior en el que hubo una ejecución de prueba incompleta, que es algo muy frecuente al momento de depurar pruebas, el secuenciador de pruebas CTS no tiene una manera directa de excluir puntualmente por comando en su propia consola. Pero, hay ciertas alternativas como:

1. Generar un archivo tipo XML llamado "cts-exclude.xml", en el cual pueden listar los módulos que no se desease considerar,
2. Usar la opción de "retry" del mismo secuenciador CTS.
3. Por último, se puede hacer un script para discriminar por medio de una lista las especificaciones de prueba o test modules no serán considerados.

Los modos de ejecución anterior serían útiles para hacer pruebas de regresión excluyendo aquellas especificaciones de prueba fuera del alcance.

Otras opciones útiles para el secuenciador de pruebas, que se pueden usar en posteriores análisis para ejecutar diferentes tipos de pruebas, son:

- Ejecutar un módulo de un tipo de pruebas especifico:

```
<plan> --module/-m <module> .
```

- Ejecutar un caso de prueba de un tipo de pruebas especifico:

```
<plan> --module/-m <module> --test/-t <test_name>    Donde el argumento del
commando "test name" puede ser <package>.<class>, <package>.<class>#<method>
o tambien <native_binary_name>
```

### 7.5.2 Análisis de la ejecución de pruebas para el Beagle board-x15

Antes de ejecutar un nuevo plan de pruebas con los casos de uso para automotive, se describirá como es que está constituida la unidad mínima de pruebas, un caso de prueba y un "test plan".

### 7.5.3 Análisis del resultado de casos de prueba.

Si bien, un caso de prueba es un conjunto o una colección mucho más grande de pasos imperativos que estimulan un sistema, esto no tendría significado hasta que éste se delimita a probar una sección de un contexto del sistema. Por ende, la colección de eventos secuenciales es el "test case" o caso de prueba.

Para el caso de las pruebas de CTS, los “test steps” o pasos de prueba están basados en el lenguaje JUnit y de las API de prueba de Android. Al ser Java un lenguaje Orientado a objetos, este tiene una clase principal y todos los test cases serán abstracciones sucesivas de una clase padre llamada “CTS”, estos son colecciones de pasos de prueba que pueden emular incluso ciertas condiciones reales implementando soluciones o medidas para depurar o analizar eventos, a esto se le conoce como instrumentación, esta instrumentación se apoya de las API ya previamente desarrolladas.

Ahora bien, a un conjunto de “test cases” el sistema de prueba de Android le conoce como “Test plan”, satisfacen una estrategia de prueba. Sin embargo, para efecto de nomenclatura interna se le conoce como “Test specification” o especificación de pruebas, que se invoca en el secuenciador de prueba como “test module”. A continuación, se define cuáles son las etapas en las cuales bajo la estrategia del plan de pruebas se operará para ejecutar las especificaciones, esto consta de las siguientes 4 etapas:

1. Determinación de las secciones a probar.
2. Alcance de la cobertura del plan de prueba.
3. Selección de los casos de prueba.
4. Análisis y reporte.

Con las etapas generales anteriormente mostradas, de manera concreta de las 4 etapas se deducen los siguientes sub-ciclos de pruebas:

- **Planeación de las pruebas o “Test planning”:** Creación de la estrategia de prueba.
    - *Entregable*: CtsCarTestCases.
- **Diseño de las pruebas o “Test design”:** Se le conocen como la creación de los “test cases”.
    - *Entregable*: “Test Cases Document”.
- **Ejecución de los casos de prueba o “Test execution”:** Ejecución de los casos de prueba.
    - *Entregable*: Ejecutable de prueba o secuenciador de pruebas.

- **Reporte de casos de prueba o "Test report":** Reportes de prueba al finalizar el ciclo de prueba.
    - *Entregable*: "Test report".

Hay un plan de pruebas predeterminados para Android Automotive. El plan determinado para Android Automotive se llama "CtsCarTestCases", esta especificación de pruebas contiene 18 "test cases" ya predefinidos, evaluados y lo cuales ya han sido ejecutados previamente para la tarjeta prototipo, en dado caso que haya un cambio en el hardware o configuración, se ajustarán los pasos de pruebas.

Por otra parte, los casos de prueba estarán documentados en la base de datos de las especificaciones de prueba. Dentro de la compañía Tier 1, es conocido con éxito probado, a la hora de tener una trazabilidad robusta el empleo de herramientas como IBM DOORS que así lo permiten. A continuación, los casos de prueba se describen brevemente:

1) **Permission Tests:** En este caso de prueba se ejercitan los permisos de aplicaciones. Especifica un permiso de sistema que el usuario debe otorgar para que la aplicación funcione correctamente. El usuario otorga permisos cuando se instala la aplicación.
2) **Exceptions Test:** En este caso de prueba se ejercitan las excepciones cuando el hardware no se conecta en alguna de las interfaces internas o externa (por ejemplo, el adb).
3) **Car Test:** En este caso de prueba se ejercitan las conexiones a servidores externos.
4) **Car Sensor Manager Test Must Support Night Sensor:** En este caso de prueba se ejercitan la interacción de algún tipo de sensor y los permisos para leer sus valores, además de excepciones y permisos con su aplicación. El grupo de sensores es el de lectura de intensidad luminosa del día. Esto va conectado a un gestor llamado "Car Sensor Manager".
5) **Car Sensor Manager Test Must Support Parking Brake:** En este caso de prueba se ejercitan la interacción de algún tipo de sensor y los permisos para leer sus valores, además de excepciones y permisos con su aplicación. El grupo de sensores es el de

lectura de sensores de estado del frenado. Esto va conectado a un gestor llamado "*Car Sensor Manager*".

6) **Car Sensor Manager Test Required Sensors For Driving State:** En este caso de prueba se ejercitan la interacción de algún tipo de sensor y los permisos para leer sus valores, además de excepciones y permisos con su aplicación. El grupo de sensores es el de lectura de sensores de la condición del manejo. Esto va conectado a un gestor llamado "Car Sensor Manager".
7) **Car Package Manager Test Activity Distraction Optimized:** En este caso de prueba se ejercitan la interacción de algún tipo de sensor y los permisos para leer sus valores, además de excepciones y permisos con su aplicación. El grupo de sensores es el de lectura de sensores de la condición del conductor durante el manejo. Esto va conectado a un gestor llamado "Car Sensor Manager".
8) **Car Package Manager Test Distraction Optimized Activity Is Allowed:** En este caso de prueba se ejercitan la interacción de algún tipo de sensor y los permisos para leer sus valores, además de excepciones y permisos con su aplicación. El grupo de sensores es el de lectura de sensores de la condición del conductor durante el manejo. Esto va conectado a un gestor llamado "*Driver Distraction Guidelines*".
9) **Car Package Manager Test Non-Distraction Optimized Activity Not Allowed:** En este caso de prueba se ejercitan la interacción de algún tipo de sensor y los permisos para leer sus valores, además de excepciones y permisos con su aplicación. El grupo de sensores es el de lectura de sensores de la condición del conductor durante el manejo. Esto va conectado a un gestor llamado "*Driver Distraction Guidelines*".
10) **Car Info Manager Test Nullables:** En este caso de prueba se ejercitan la interacción de la librería que maneja la información compartida sobre accesorio del vehículo. Esto va conectado a una API gestora llamada "Car Info Manager".
11) **Car Info Manager Test Vehicle Id:** En este caso de prueba se ejercitan la interacción de la librería que maneja la información compartida sobre accesorio de identificación del vehículo. Esto va conectado a una API gestora llamada "Car Info Manager".
12) **Car Bluetooth Test:** En este caso de prueba se ejercitan la interacción de la librería que maneja la información compartida sobre bluetooth. Esto va conectado a una API

gestora llamada "*Car Bluetooth*". Actualmente la tarjeta "Beagle board" no tiene soporte de HW para usar estos recursos por SW. La prueba relacionada debería fallar.

Ahora se tocará el tema del "CarAppFocusManager", es una clase que permite que las aplicaciones que usen o llamen este recurso configuren y se relacionen en cuanto a las APIs adecuadamente para el intercambio de datos como la navegación activa o el comando de voz. Por lo general, sólo una instancia de dicha aplicación debe ejecutarse en el sistema, y otra aplicación que quisiera establecer comunicación, el indicador para la aplicación tenga prioridad y debe hacer que otra aplicación se detenga. Los casos de prueba *13 a 18* prueban los siguientes métodos de dicha clase Test Filter, Focus Change, Listeners Per Manager, Register Null, Register Unregister, Set Active Null Listener.

Se hace un paréntesis en Apéndices de la sección "*IMPORTACION DE TEST CASES A BASE DE DATOS DE REQUISITOS*", donde se da un poco de contexto sobre cómo estarían administradas las pruebas dentro la herramienta IBM DOORS para gestión de requisitos y pruebas.

#### *7.5.3.1.1 Ejecución del "test plan" de Automotive.*

Una vez se ha descargado los archivos adecuados, se vuelve a cargar los casos de prueba del CTS, se vuelve a ejecutar el comando. Si el DUT se ha detectado, entonces, se hacen llamar los "test cases" correspondientes automotrices como se muestra en la Imagen 92:

```
$sudo ./cts-tradefed
$cts-tf > run cts --module CtsCarTestCases
```


*Imagen 92. Llamada a las pruebas de automotive.*

Una vez que se finaliza la ejecución del "test plan", el "test script" creará una serie de documentos útiles (test logs) para trazar resultados pasados, fallidos y no ejecutados. Estos se almacenarán en la siguiente ubicación:

```
$ cd android-cts/results/<time_stamp>
```

La consola de la terminal arrojará el siguiente resumen de los resultados Imagen 93:

```
10-31 22:46:54 I/SuiteResultReporter:
============================================
================ Results ===================
=============== Consumed Time ==============
    armeabi-v7a CtsCarTestCases: 10s
Total aggregated tests run time: 10s
============== TOP 1 Slow Modules ==============
    armeabi-v7a CtsCarTestCases: 1.79 tests/sec [18 tests / 10033 msec]
============== Modules Preparation Times ==============
    armeabi-v7a CtsCarTestCases => prep = 2981 ms || clean = 333 ms
Total preparation time: 2s  ||  Total tear down time: 333 ms
============================================================
============== Summary ===============
Total Run time: 1m 58s
1/1 modules completed
Total Tests       : 18
PASSED            : 16
FAILED            : 1
ASSUMPTION_FAILURE: 1
============== End of Results ==============
============================================
```

*Imagen 93. Resultados de los test cases de automotive.*

### *7.5.3.1.2 Interpretación de los resultados de CTS de Automotive.*

Como se ha comentado previamente, dentro de las carpetas creadas por los scripts se arrojan los resultados siguientes de acuerdo con la Imagen 94:

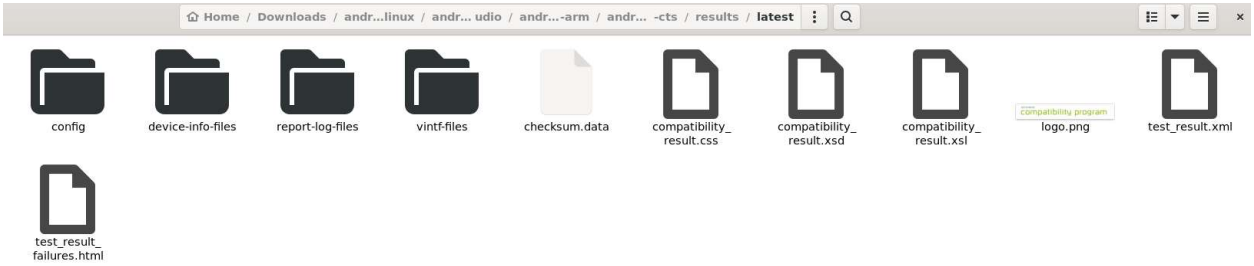



*Imagen 94. Localización de los resultados de los test cases de automotive.*

La sección "información del dispositivo" proporciona detalles sobre el dispositivo y el firmware (marca, modelo, compilación del firmware, plataforma) además de las versiones del hardware del dispositivo. Los detalles del plan de prueba estarán en la sección de “resumen de prueba” donde aparece el nombre del “test plan en CTS”, las horas de inicio y finalización de las pruebas. De igual manera, se presentará un resumen (o “test summary report”) acerca de la cantidad de “test cases” o pruebas que pasaron, fallaron, expiraron o no pudieron ser ejecutadas. Dichas pruebas se corrieron bajo los estados 1, 2 y 3 de la sección “7.4.1.1.3”.

En caso de que falle una prueba, los detalles para depurar los errores se presentarán en forma de reporte. Además, el seguimiento de la falla estará disponible en un archivo en formato XML, dicho archivo XML proporciona detalles de la prueba fallida (ubicando la etiqueta <Test> correspondiente a las pruebas fallidos y seguidos de la etiqueta <StackTrace>) como se muestra en la Imagen 95.


android compatibility program

| Summary | |
|---|---|
| Suite / Plan | CTS / cts |
| Suite / Build | 9.0_r20 / 8503603 |
| Host Info | Result/@start eduardo-Inspiron-5758 (Linux - 5.15.0-52-generic) |
| Start time / End Time | Sat Nov 12 18:58:07 CST 2022 / Sat Nov 12 19:01:32 CST 2022 |
| Tests Passed | 17 |
| Tests Failed | 1 |
| Modules Done | 1 |
| Modules Total | 1 |
| Fingerprint | Android/beagle_x15_auto/beagle_x15:9/PPR1.181005.003/alejandro11080831:userdebug/test-keys |
| Security Patch | 2018-10-05 |
| Release (SDK) | 9 (28) |
| ABIs | armeabi-v7a,armeabi |

| Module | Passed | Failed | Total Tests | Done |
|---|---|---|---|---|
| armeabi-v7a CtsCarTestCases | 17 | 1 | 18 | true |

armeabi-v7a CtsCarTestCases

| Test | Result | Details |
|---|---|---|
| android.car.cts.CarBluetoothTest#testRequiredBluetoothProfilesExist | fail | java.lang.NullPointerException: Attempt to invoke virtual method 'boolean android.bluetooth.BluetoothAdapter.disable()' on a null object reference |

*Imagen 95. Test report. Nota: Hay una excepción en el ciclo de pruebas ejecutadas con la tarjeta Beagle board X15.*

El reporte de pruebas arroja resultados mostrados como “test_result_failures.html returned one failed test *android.car.cts.CarBluetoothTest#testRequiredBluetoothProfilesExist*”. La causa raíz de la falla muestra “java.lang.NullPointerException: Attempt to invoke virtual method ‘boolean android.bluetooth.BluetoothAdapter.disable()’ on a null object reference”. Esto es un resultado esperado para el AM57xx BeagleBoard-X15 and AM65x EVM ya que no tiene soporte para Bluetooth actualmente, si hubiera requisitos de alguna aplicación que lo necesite, entonces la contramedida es adaptar un transmisor-receptor de Bluetooth al prototipo X15. Por lo tanto, este resultado es consistente o esperado. La idea de mantener esta prueba, que de antemano se sabe que no estaba soportada por la plataforma BeagleBoard-X15, es para mostrar que el secuenciador tiene capacidad para reconocer fallos en el dispositivo bajo prueba.

#### *7.5.3.1.3 Resto de casos de prueba de CTS.*

Si bien hay sólo 18 de CtsCarTestCases, hay cerca de 322 especificaciones de prueba que pudieran ejecutarse para tener un análisis completo del prototipo en esta fase. Por ahora el enfoque es sólo los datos relevantes para automotriz, pero los datos de las demás pruebas se van a mostrar en Apéndices (véase LISTA COMPLETA DE ESPECIFICACIÓN DE PRUEBAS DEL CTS V.9.0)

# Conclusiones

Conforme a los objetivos propuestos al inicio del proyecto, se buscó el uso de hardware y software de terceros ya disponible con AAOSP para soportar actividades relacionadas al desarrollo y configuración inicial. De igual manera, se analizó la facilidad de ajuste de tiempos de arranque del sistema, desarrollo de una aplicación, y pruebas para validación de software. La intención era evaluar si el uso de estas plataformas puede reducir el tiempo de desarrollo significativamente, y a la vez mantener la calidad que la industria automotriz requiere.

Podemos tomar como referencia la Tabla 1, la cual lista cinco Epics usados sólo para la etapa de bring-up en un entorno similar al usado en este proyecto:

- HAL Infrastructure Standalone bringup
- HAL application Framework Standalone
- HAL Security, Media, Location
- HAL Integration
- HAL systems

Los Epics anteriores tomaron un tiempo mayor a siete meses para ser completados, solamente con el objetivo de manejar el hardware y cargar el software básico, como drivers y el sistema operativo. Este proyecto de trabajo recepcional se ha desarrollado en un tiempo efectivo final de cuatro meses (ver el Apéndice "Diagrama Gantt con la planeación definida"), sólo contemplando el desarrollo. En total, tomando en cuenta el diseño, planeación, y el documento final, fueron siete meses. Esto representa una reducción de tiempo valiosa comparado con los siete meses que tomó desarrollar el "bring-up" en el proyecto de referencia.

Con respecto de la preparación del ambiente de desarrollo. Fue extremadamente evidente el beneficio obtenido al cambiar el proyecto en un entorno y plataforma orientadas al desarrollo "Open Source".

La elección inicial de hardware y software, aunque basados en AAOSP, pero parte de un "release" propietario, demostró casi inmediatamente desventajas para el desarrollo de este

proyecto. Lejos de agilizar el inicio de actividades al tener propietarios que pudieran responder, representó una ralentización completamente innecesaria de actividades críticas para establecer la plataforma sobre la cual se trabajaría. Las limitaciones en información disponible de esquemáticos de la tarjeta de desarrollo, la falta de visibilidad completa y acceso a los cambios aplicados sobre AAOSP, el uso obligatorio de herramientas específicas del fabricante del "chipset", la incertidumbre a la versatilidad a largo plazo de la tarjeta de desarrollo debido al proceso de adquisición de la compañía fabricante, y las condicionantes al soporte cuando fue requerido; fueron factores determinantes en la pronta elección de una segunda alternativa. En esta primera instancia se perdieron alrededor de dos meses, tras los cuales fue necesario reiniciar el proyecto.

Una vez establecido un entorno de Hardware y Software orientado a "Open Source", la preparación de la plataforma sobre la cual se iniciarían actividades de desarrollo fue casi trivial, tomando meramente un par de semanas. La información de los esquemáticos de la tarjeta de desarrollo permitió rápidamente comprender errores de configuración inicial, la visibilidad de la totalidad de los cambios en código permitió entender de manera sencilla el reléase de software que se estaba usando, y la enormidad de los foros de discusión hicieron del soporte una simple búsqueda en todos los casos.

Se puede concluir, al menos para el desarrollo de este proyecto, que el uso de una plataforma realmente "Open Source", ya sea de hardware o de software, acelera el inicio de actividades de desarrollo al poner al alcance del equipo toda la información disponible sin otra limitación que el alcance mismo de la comunidad que lo soporta.

En cuanto a la estrategia de arranque del sistema, al usar la mayoría de las prácticas de los sistemas Linux y mejorarlas con el uso de convenciones específicas en Android, también se encontró una adecuación ágil de los tiempos de arranque, a pesar de sus limitaciones en flexibilidad debido a la naturaleza misma del sistema. El tiempo obtenido no fue el reflejando en la documentación con su "benchmark" en "Android Marshmellow" y "Jacinto board", pero el tiempo invertido en documentarse adecuadamente y aplicar el cambio fue mínimo, un par de días en total.

De los resultados obtenidos con respecto al uso de la RAM en la aplicación de multimedia, se puede concluir que hay cierto riesgo de presentar "leaks" de memoria en la clase "SampleQueue" puesto que el espacio reservado por dicha clase puede aumentar incluso el doble

tomando como referencia el caso de uso en la sección “6.4.1.3”. Si bien, no se obtuvieron mediciones cuando la aplicación está en condiciones de estrés, se puede asumir que la clase podría incrementar mucho más el espacio reservado al observar su implementación en código, es por ello, que en este tipo de casos es importante obtener más datos para saber de manera más certera donde aplicar mejoras. Con estos resultados se considera como una posible mejora, pero no se garantiza que optimizando el manejo de la clase SampleQueue podría reducir el uso de memoria sin afectar el desempeño de la aplicación.

También se puede concluir que el uso de herramientas como Mat Tool para analizar como una aplicación asigna memoria es una forma interesante de aprender como Android maneja las asignaciones de memoria y encontrar formas de optimizar como una aplicación maneja la memoria. Muchas veces cuando se tiene prisa de lanzar un producto este tipo de análisis no son prioridad por el tiempo que requieren, pero el uso de este tipo de herramientas simplifica en gran parte el proceso.

Las pruebas hechas para obtener los resultados de la memoria fueron sólo para analizar el uso de la RAM y sus posibles optimizaciones, lamentablemente los resultados de consumo de RAM en el proyecto desarrollado en la compañía Tier 1 no se pueden mostrar para hacer comparativas, pero son similares.

Como cualquier proyecto automotriz, se realizaron pruebas para validar y verificar, pero no de forma exhaustiva porque es sólo un prototipo en esta fase, sin embargo, el proyecto parece ser capaz de ser incorporado en un vehículo con la adecuada adaptación a requisitos de industrialización.

Por otra parte, es claro que este documento no representa todo lo que involucra desarrollar un sistema automotriz al de manera integral, pues hay diferentes factores financieros como la inversión en investigación y desarrollo, administrativos por la condición de mercado en cada país o región, y hasta humanos por aquello de las capacidades técnicas, pero con base a lo anterior, podemos concluir que el proyecto desarrollado podría ser usado en un sistema automotriz, lo cual

demuestra que usar software de terceros, en este caso de Android con su aplicación primaria y el verificador CTS con su infraestructura de pruebas, lo cual optimiza el tiempo de desarrollo de un proyecto automotriz de Info-entretenimiento, pero con el tiempo adecuado en investigación y desarrollo tomar conceptos clave como el caso del "kernel" del SO.

Con respecto al capítulo 7, hacer pruebas dentro de un ambiente automotriz es una cuestión retadora en cuanto a tiempo de desarrollar toda una infraestructura de pruebas completa, lo cual va integrando un ambiente ya probado como el caso de Android que ya ha estado en el mercado por más de 10 años, eso dentro del entorno automotriz agrega confiabilidad y robustes con mucho menores tiempos de desarrollo de aplicaciones basadas en el que hace pocos años sólo estaba dirigido a móviles. Un área de oportunidad detectada durante la integración de los "test harnesses" como los que tiene Android desarrollados para automotriz y dirigidos para sistemas embebidos móviles basados en Android, es la creación de un símil de equipos de pruebas independientes enfocados en esta área en etapa cuasi de prototipo por parte de proveedores disruptivos como Android que ganen experiencia en este sistema que va despegando y entrenar recursos tanto humanos como materiales aptos para adaptar todos los cambios de la manera más ágil. Usualmente lleva 2 años desarrollar una plataforma de HW, SW y sus respectivas actividades de pruebas, respetando el modelo imperante en V a pesar de que la gestión se está llevando a cabo en herramientas con miras a hacerlo en ágil. Por otra parte, el tiempo en configurar el ambiente de pruebas y correr las primeras pruebas de estabilización antes de la llegada de la muestra final de producción tome cerca de 2 meses entre el periodo de aprendizaje, adaptación de la infraestructura como el CTS, integración de las herramientas de Android con secuenciadores nativos de las áreas de negocio y la ejecución para depuración de los primeros. Además, una vez que el proyecto este fuera o este por salir al mercado en la etapa de producción masiva, sería más sencillo que los que llevan a cabo cualquier tipo de pruebas lo hagan en un tiempo considerablemente menor debido a la adaptación previa. En cuestiones de implementación, estas pruebas serían las primeras a un sistema basados en Android Automotriz en la región para la unidad de negocios lo que también abona a cuestiones de innovación en el proceso de prueba además de abrir la puerta para profundizar en el conocimiento de nuevas herramientas, procesos y productos por lo que al implantar esas mejoras en la metodología, técnicas y herramientas aumentaría la productividad del equipo de pruebas. Los resultados obtenidos por ellos se verían en la optimización o reducción de

tiempo y costos ya que la expectativa produciría menos ciclos de pruebas, como ejemplo se tiene que el VTS no tuvo que ejecutarse que el fabricante lo hizo previamente para poder ofertar la plataforma, algún cambio en la HAL sólo requeriría una regresión. Aquí un criterio cuantificable es la ejecución y adaptación de la infraestructura de pruebas. Esta por lo menos toma la mitad del tiempo que lleva hacer desde la nada en un proyecto no Android. Sin embargo, dicha ventaja no se tiene en la parte de desarrollo de requisitos y su arquitectura. Sólo mantiene una ventaja muy ligera con respecto al reúso.

En la parte de integración con el vehículo, es decir si cumple con los estándares mínimos de integración no tuvieron conflicto con los casos de prueba relacionados al estado de ignición, al controlador inicial de información ni los sensores relevantes por consecuencia las pruebas de sistema arrojar que es candidato para integrar el ECU o DUT en un automóvil.

Otro aspecto que interesa es unir el uso de ambientes de prueba enfocados a un producto que es candidato a estado del arte son su refinamiento y adaptación a requisitos para su comercialización en serie, conclusión de sus mejores prácticas y seguimiento de estas. Para lo anterior se debe escoger métricas específicas, cualitativas y cuantificables, de esa forma se puede llevar un registro para agregar a los nuevos reportes.

# Glosario

| Palabra | Definición |
|---|---|
| AAOS | Android Automotive Operating Systems |
| Aapt | Android Asset Packaging Tool |
| ABI(s) | Application Binary Interface |
| Activity | El cliente principal dentro de la aplicación que incluye a los módulos de software Media Browser, Media Controller y la interfaz de usuario (UI). |
| Adb | Android Debug Bridge |
| AM | Amplitud modulada |
| Android KitKat | Undécima versión del sistema operativo Android |
| Android Lollipop | Quinta versión del sistema operativo Android |
| Android Studio Profiler | Herramientas de Android que proporciona datos en tiempo real que te ayudan a comprender la forma en que tu app utiliza los recursos de la CPU, la memoria, la red y la batería. |
| API(s) | Application Programming Interfaces |
| App | Application |
| ARM | Advanced RISC Machine |
| ART | Android Run time |
| Assets | Conjunto de archivos arbitrarios como texto, tipo XML, HTML, fuentes, música y video en una aplicación. |
| Backlog | Es pila de espera para las próximas actividades o tareas. |
| Bluetooth | Es un protocolo de comunicaciones que sirve para la transmisión inalámbrica de datos (fotos, música, contactos, etc.) y voz entre diferentes dispositivos. |
| Bootloader | Es el área de código encargada de la gestión de inicio en un sistema basado en un procesador. |
| BringUp | Conjunto de tareas de desarrollo del tipo Epic ejecutadas para levantar. |
| S | Es código ejecutable que se pasa como argumento a otro código, que se espera que devuelva la llamada, es decir, este debe ejecutar el argumento. |
| Canary | Uno de los canales o versiones de Android Studio. |
| CDD | Compatibility Definition Document. |
| CMP | |
| Code | Memoria de código que usa el procesador para que se realice sus funciones. |
| CPU | Central Processing Unit. |
| CTS | Compatibility Test Suite es un conjunto de funciones para gestión de pruebas nativo en el ambiente Android. |
| Daemon | Es un servicio ejecutado en segundo plano. |
| Dalvik | El manejador principal de la memoria y pronto |

| | |
|---|---|
| DDMS | Dalvik Debug Monitor Server. |
| Delay loops | Ciclos de retardo |
| DOORS | Dynamic Object-Oriented Requirements System de IBM para gestión de requisitos bajo un paradigma de objetos. |
| Dump (Archivos) | Son aquellos y obtenidos y exportados desde Android Studio, se convirtieron a un formato estándar que Mat Tool pudiera entender pues Android tiene un formato especifico. |
| DUT | Device Under Test es un artefacto de pruebas. |
| Eclipse | Editor de Código |
| ExoPlayer | Es un proyecto de código abierto que no forma parte del marco de trabajo de Android y se distribuye por separado desde el SDK de Android. |
| Features | Aspecto de una aplicación que cubre las necesidades de un usuario. |
| FM | Frecuencia Modulada de ondas de radio. |
| Garbage Collection | Mecanismo de gestión automático de memoria |
| GHz | Giga Hertz |
| GPS | Global Positioning System. |
| HAL | Hardware Abstraction Layer. |
| Hardware | Elementos que componen un sistema embebido. |
| Heap | Memoria dinámica. |
| HMI | Human Machine Interface. |
| IDLE | Es un estado de inactividad. |
| IEC 15504 | Norma para desarrollo de Software. |
| Info-entretenimiento | Estrategias de interacción visual y auditiva para información de estados del vehículo. |
| Java | Plataforma de programación orientada a objetos. |
| JUnit | Una infraestructura de prueba para unidades de código basadas en Java. |
| Kernel | Referencia al programa del núcleo del sistema operativo. |
| Leak | Fuga de memoria debido a problemas de memoria en la aplicación. |
| LMK | Low Memory Killer mecanismo cuando no puede liberar suficiente memoria para el sistema. |
| LTS | Long Term Support es una versión de alguna distribución de Linux confiables |
| MAT Tool | Eclipse Memory Analyzer es un analizador para el heap en Java |
| MB | Megabytes |

| | |
|---|---|
| Memory Profiler | Herramienta del IDE de Android en el cual se podrá observar el uso de la memoria RAM de los procesos seleccionados o del sistema completo. |
| Multitasking | Enfoque de un sistema operativo que divide el tiempo de procesador de una manera programática |
| NAD | Network Access Device. |
| Network | Conexión a red. |
| OEM(s) | Original Equipment Manufacturer. |
| OHA | Open Handset Alliance. |
| OS | Operating System |
| PC | Personal Computer |
| RAM | Random Access Memory |
| ROM | Read-Only memory |
| Root file system | Es el directorio a alto nivel de un sistema de archivos |
| SAFe | Scaled agile framework es un conjunto de patrones de organización y flujo de trabajo. |
| Scheduler | Es un componente funcional que planifica la ejecución de tareas programáticas |
| SD | Secure Digital es una tarjeta de memoria tipo flash |
| SDK | Software Development Kit |
| SIM | Subscriber Identity Module es un tipo de chip para transmisión de datos |
| Software | Rutinas programáticas para realización de tareas. |
| SPICE | Automotive Software Process Improvement Capability Determination conjunto de prácticas para evaluación de un ciclo de desarrollo de software. |
| Sprint(s) | Periodo determinado de tiempo en el cual una o varias tareas deben ser completadas |
| Stack | Tipo de memoria para llamadas de funcionales |
| Tester | Elemento que gestiona las pruebas |
| Timers | Elementos temporizadores |
| Tradefed | Trade Federation. |
| UI | Interfaz de usuario. |
| UML | Unified Modeling Language es un conjunto de reglas de modelado de sistemas. |
| USB | Universal Serial Bus es un protocolo de transmisión de datos. |
| V&V | Verificación y validación. |
| WearOS | Versión de Android para relojes y otros dispositivos usados como accesorios de vestimenta. |
| Wi-Fi | Protocolo de conexión inalámbrica. |
| 3G | Tercera generación de telefonía inalámbrica. |
| 4G | Cuarta generación de telefonía inalámbrica. |
| 5G | Quinta generación de telefonía inalámbrica. |

# Apéndices

## DIAGRAMA GANTT CON LA PLANEACIÓN DEFINIDA

El siguiente archivo contiene un archivo con el diagrama generado a partir de código escrito en Plant UML, la cual es una herramienta para crear diagramas de UML. El Anexo B es el código usado para generar el siguiente diagrama (también disponible en ITESO_MDE_Project [31]):

AAOSP_Planning.svg

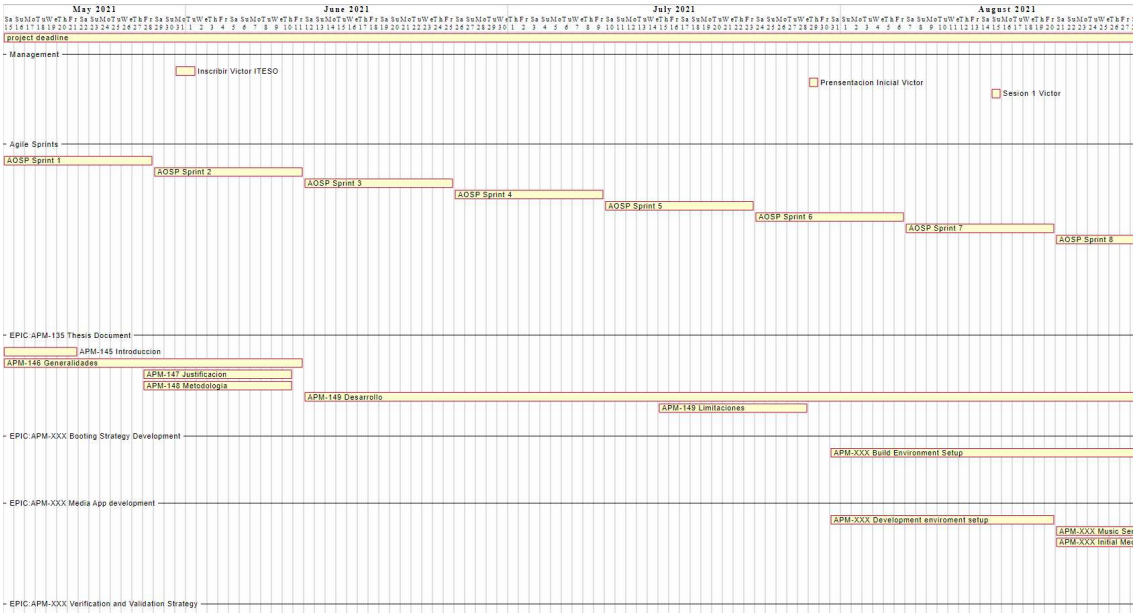



*Imagen 96. línea de tiempo de desarrollo de proyecto.*

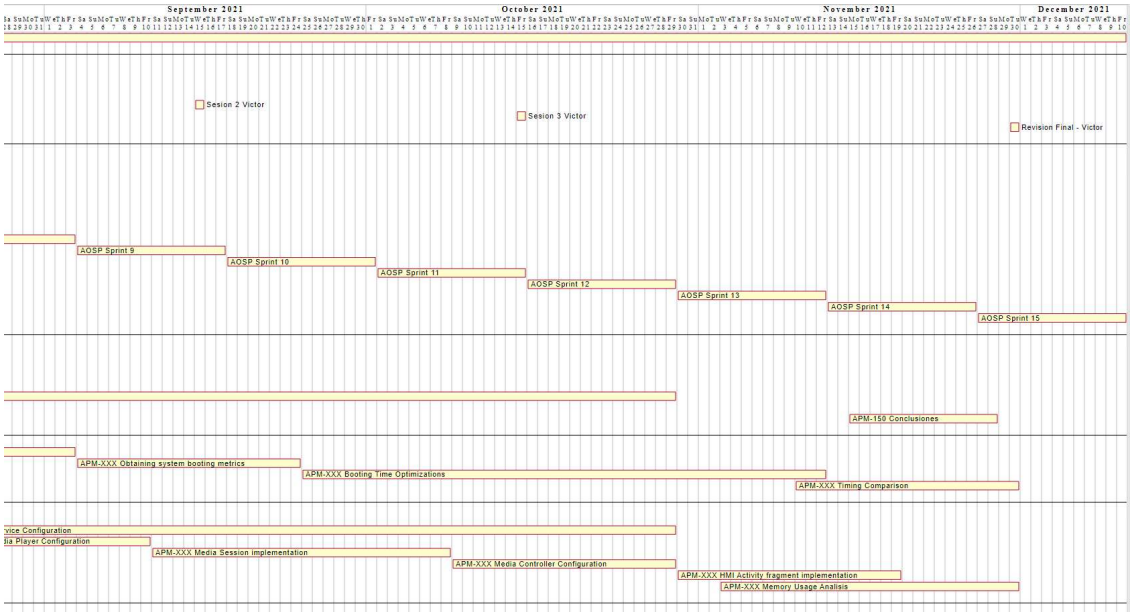



*Imagen 97. Linea de tiempo de desarrollo 2*

## CODIGO DE PLANTUML PARA GENERACION DE GANTT

El código de PLANTUML para generar el Gantt está también disponible en : https://gitlab.com/iteso_mde_project

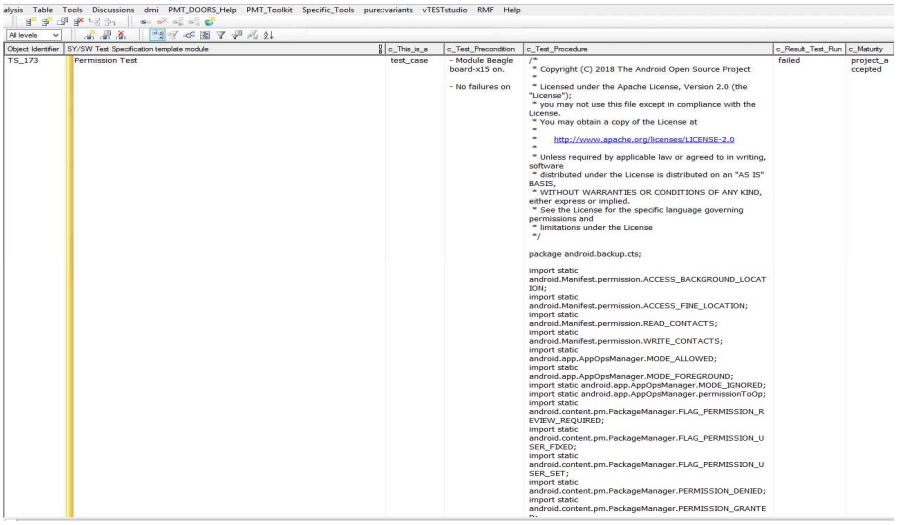
```
@startgantt
Project starts 2021-05-15
[project deadline] lasts 210 days

-- Management --
[Inscribir Victor ITESO] starts 2021-05-31
[Inscribir Victor ITESO] lasts 2 day

[Prensentacion Inicial Victor] starts 2021-07-29
[Prensentacion Inicial Victor] lasts 1 day

[Sesion 1 Victor] starts 2021-08-15
[Sesion 1 Victor] lasts 1 day

[Sesion 2 Victor] starts 2021-09-15
[Sesion 2 Victor] lasts 1 day

[Sesion 3 Victor] starts 2021-10-15
[Sesion 3 Victor] lasts 1 day

[Revision Final - Victor] starts 2021-11-30
[Revision Final - Victor] lasts 1 day

-- Agile Sprints --
[AOSP Sprint 1] starts 2021-05-15
[AOSP Sprint 1] lasts 2 week

[AOSP Sprint 2] starts 2021-05-29
[AOSP Sprint 2] lasts 2 week

[AOSP Sprint 3] starts 2021-06-12
[AOSP Sprint 3] lasts 2 week

[AOSP Sprint 4] starts 2021-06-26
[AOSP Sprint 4] lasts 2 week

[AOSP Sprint 5] starts 2021-07-10
[AOSP Sprint 5] lasts 2 week

[AOSP Sprint 6] starts 2021-07-24
[AOSP Sprint 6] lasts 2 week

[AOSP Sprint 7] starts 2021-08-07
[AOSP Sprint 7] lasts 2 week

[AOSP Sprint 8] starts 2021-08-21
```

```
[AOSP Sprint 8] lasts 2 week

[AOSP Sprint 9] starts 2021-09-04
[AOSP Sprint 9] lasts 2 week

[AOSP Sprint 10] starts 2021-09-18
[AOSP Sprint 10] lasts 2 week

[AOSP Sprint 11] starts 2021-10-02
[AOSP Sprint 11] lasts 2 week

[AOSP Sprint 12] starts 2021-10-16
[AOSP Sprint 12] lasts 2 week

[AOSP Sprint 13] starts 2021-10-30
[AOSP Sprint 13] lasts 2 week

[AOSP Sprint 14] starts 2021-11-13
[AOSP Sprint 14] lasts 2 week

[AOSP Sprint 15] starts 2021-11-27
[AOSP Sprint 15] lasts 2 week

-- Epic:APM-135 Thesis Document --
[APM-145 Introduccion] starts 2021-05-15
[APM-145 Introduccion] lasts 7 day

[APM-146 Generalidades] starts 2021-05-15
[APM-146 Generalidades] lasts 4 week

[APM-147 Justificacion] starts 2021-05-28
[APM-147 Justificacion] lasts 2 week

[APM-148 Metodologia] starts 2021-05-28
[APM-148 Metodologia] lasts 2 week

[APM-149 Desarrollo] starts 2021-06-12
[APM-149 Desarrollo] lasts 20 week

[APM-149 Limitaciones] starts 2021-07-15
[APM-149 Limitaciones] lasts 2 week

[APM-150 Conclusiones] starts 2021-11-15
[APM-150 Conclusiones] lasts 2 weeks

-- Epic:APM-XXX Booting Strategy Development --
[APM-XXX Build Environment Setup] starts 2021-07-31
[APM-XXX Build Environment Setup] lasts 5 week

[APM-XXX Obtaining system booting metrics] starts 2021-09-04
[APM-XXX Obtaining system booting metrics] lasts 3 week

[APM-XXX Booting Time Optimizations] starts 2021-09-25
```

```
[APM-XXX Booting Time Optimizations] lasts 7 week

[APM-XXX Timing Comparison] starts 2021-11-10
[APM-XXX Timing Comparison] lasts 3 week

-- Epic:APM-XXX Media App development --
[APM-XXX Development enviroment setup] starts 2021-07-31
[APM-XXX Development enviroment setup] lasts 3 week

[APM-XXX Music Service Configuration] starts 2021-08-21
[APM-XXX Music Service Configuration] lasts 10 week

[APM-XXX Initial Media Player Configuration] starts 2021-08-21
[APM-XXX Initial Media Player Configuration] lasts 3 week

[APM-XXX Media Session implementation] starts 2021-09-11
[APM-XXX Media Session implementation] lasts 4 week

[APM-XXX Media Controller Configuration] starts 2021-10-09
[APM-XXX Media Controller Configuration] lasts 3 week

[APM-XXX HMI Activity fragment implementation] starts 2021-10-30
[APM-XXX HMI Activity fragment implementation] lasts 3 week

[APM-XXX Memory Usage Analisis] starts 2021-11-03
[APM-XXX Memory Usage Analisis] lasts 4 week

-- Epic:APM-XXX Verification and Validation Strategy --

@endgantt
```

# IMPORTACION DE TEST CASES A BASE DE DATOS DE REQUISITOS

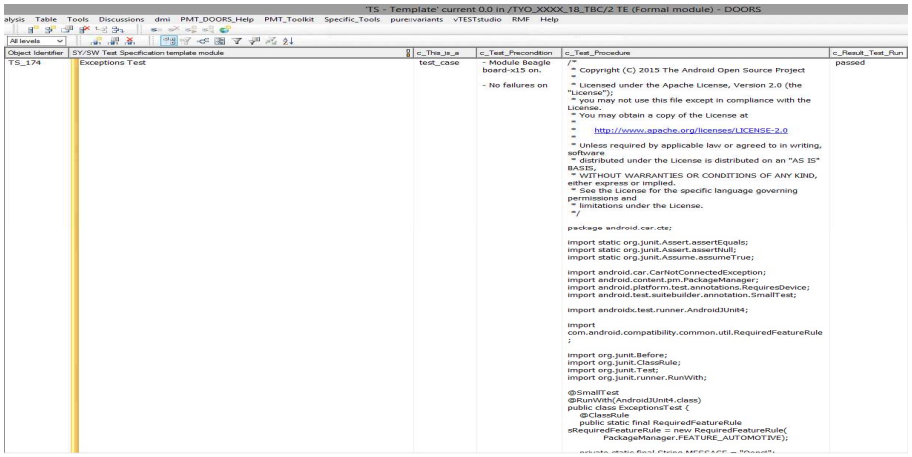
1. Permission Tests



| Object Identifier | SY/SW Test Specification template module | c_This_is_a | c_Test_Precondition | c_Test_Procedure | c_Result_Test_Run | c_Maturity |
|---|---|---|---|---|---|---|
| TS_173 | Permission Test | test_case | - Module Beagle board-x15 on.<br><br>- No failures on | /*<br>* Copyright (C) 2018 The Android Open Source Project<br>*<br>* Licensed under the Apache License, Version 2.0 (the "License");<br>* you may not use this file except in compliance with the License.<br>* You may obtain a copy of the License at<br>*<br>* http://www.apache.org/licenses/LICENSE-2.0<br>*<br>* Unless required by applicable law or agreed to in writing, software<br>* distributed under the License is distributed on an "AS IS" BASIS,<br>* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.<br>* See the License for the specific language governing permissions and<br>* limitations under the License<br>*/<br><br>package android.backup.cts;<br><br>import static android.Manifest.permission.ACCESS_BACKGROUND_LOCATION;<br>import static android.Manifest.permission.ACCESS_FINE_LOCATION;<br>import static android.Manifest.permission.READ_CONTACTS;<br>import static android.Manifest.permission.WRITE_CONTACTS;<br>import static android.app.AppOpsManager.MODE_ALLOWED;<br>import static android.app.AppOpsManager.MODE_FOREGROUND;<br>import static android.app.AppOpsManager.MODE_IGNORED;<br>import static android.app.AppOpsManager.permissionToOp;<br>import static android.content.pm.PackageManager.FLAG_PERMISSION_REVIEW_REQUIRED;<br>import static android.content.pm.PackageManager.FLAG_PERMISSION_USER_FIXED;<br>import static android.content.pm.PackageManager.FLAG_PERMISSION_USER_SET;<br>import static android.content.pm.PackageManager.PERMISSION_DENIED;<br>import static android.content.pm.PackageManager.PERMISSION_GRANTE | failed | project_accepted |

*Imagen 98. Permission Tests en DOORS*

2. Exceptions Test



| Object Identifier | SY/SW Test Specification template module | c_This_is_a | c_Test_Precondition | c_Test_Procedure | c_Result_Test_Run |
|---|---|---|---|---|---|
| TS_174 | Exceptions Test | test_case | - Module Beagle board-x15 on.<br><br>- No failures on | /*<br>* Copyright (C) 2015 The Android Open Source Project<br>*<br>* Licensed under the Apache License, Version 2.0 (the "License");<br>* you may not use this file except in compliance with the License.<br>* You may obtain a copy of the License at<br>*<br>* http://www.apache.org/licenses/LICENSE-2.0<br>*<br>* Unless required by applicable law or agreed to in writing, software<br>* distributed under the License is distributed on an "AS IS" BASIS,<br>* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.<br>* See the License for the specific language governing permissions and<br>* limitations under the License.<br>*/<br><br>package android.car.cts;<br><br>import static org.junit.Assert.assertEquals;<br>import static org.junit.Assert.assertNull;<br>import static org.junit.Assume.assumeTrue;<br><br>import android.car.CarNotConnectedException;<br>import android.content.pm.PackageManager;<br>import android.platform.test.annotations.RequiresDevice;<br>import android.test.suitebuilder.annotation.SmallTest;<br><br>import androidx.test.runner.AndroidJUnit4;<br><br>import com.android.compatibility.common.util.RequiredFeatureRule;<br><br>import org.junit.Before;<br>import org.junit.ClassRule;<br>import org.junit.Test;<br>import org.junit.runner.RunWith;<br><br>@SmallTest<br>@RunWith(AndroidJUnit4.class)<br>public class ExceptionsTest {<br>@ClassRule<br>public static final RequiredFeatureRule sRequiredFeatureRule = new RequiredFeatureRule(<br>PackageManager.FEATURE_AUTOMOTIVE); | passed |

*Imagen 99. Exceptions Test en DOORS.*

3. Car Test

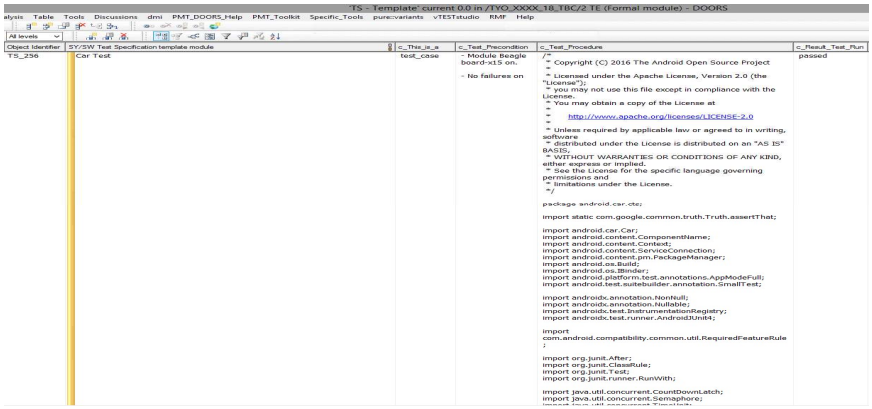



*Imagen 100. Car Test en DOORS.*

4. Car Sensor Manager Test Must Support Night Sensor

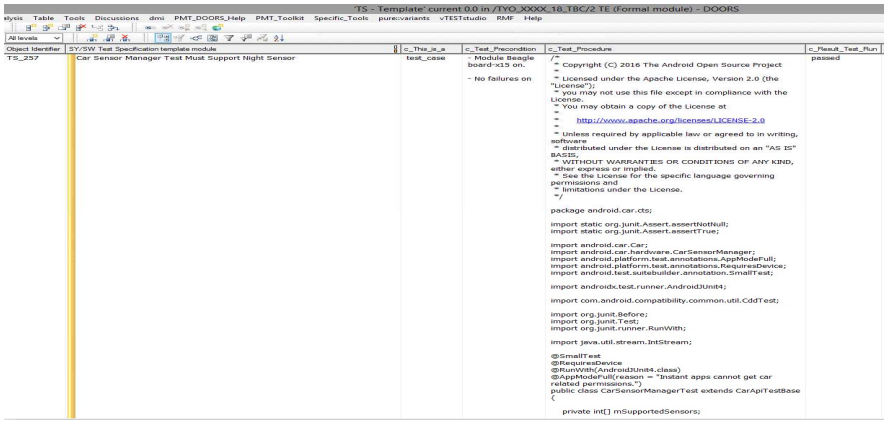



*Imagen 101. Car Sensor Manager Test Must Support Night Sensor en DOORS.*

5. Car Sensor Manager Test Must Support Parking Brake

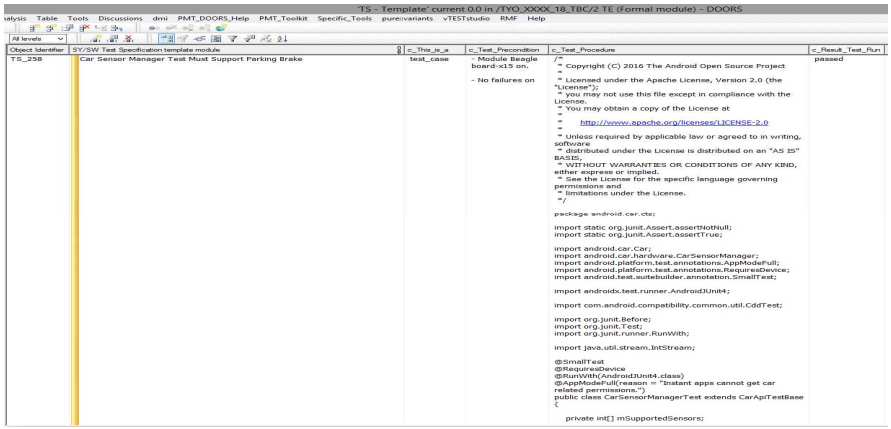



*Imagen 102. Car Sensor Manager Test Must Support Parking Brake en DOORS.*

6. Car Sensor Manager Test Required Sensors For Driving State

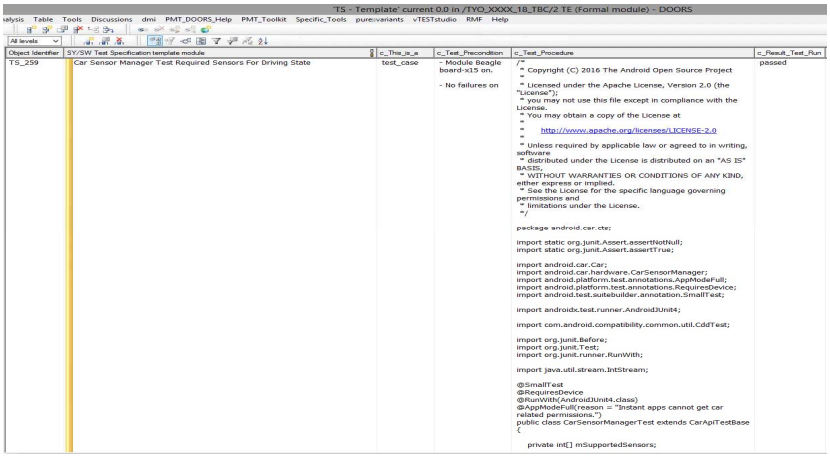



*Imagen 103. Car Sensor Manager Test Required Sensors For Driving State en DOORS.*

7. Car Package Manager Test Activity Distraction Optimized

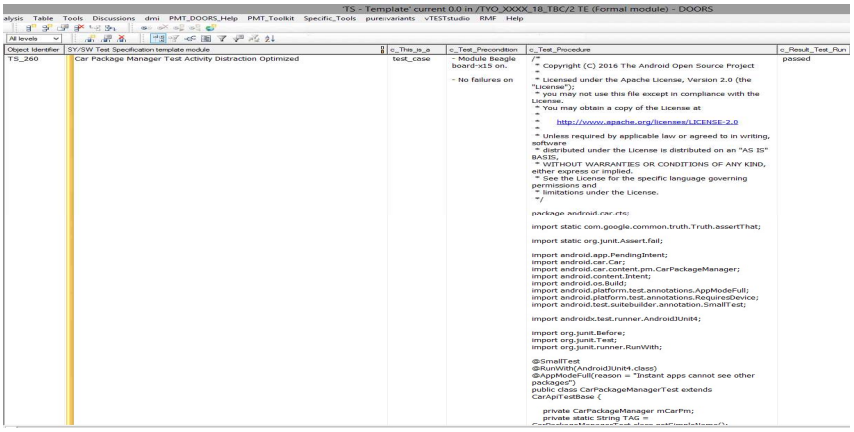



*Imagen 104. Car Package Manager Test Activity Distraction Optimized en DOORS.*

8. Car Package Manager Test Distraction Optimized Activity Is Allowed

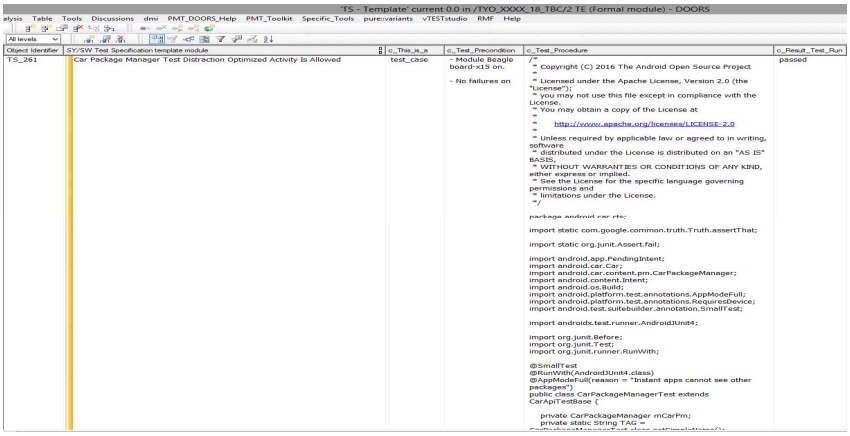



*Imagen 105. Car Package Manager Test Distraction Optimized Activity Is Allowed en DOORS.*

9. Car Package Manager Test Non-Distraction Optimized Activity Not Allowed

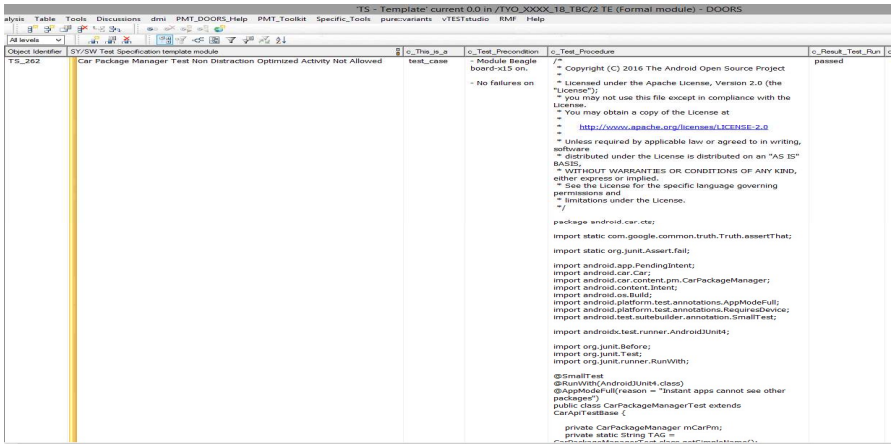



*Imagen 106. Car Package Manager Test Non-Distraction Optimized Activity Not Allowed en DOORS.*

10. Car Info Manager Test Nullables

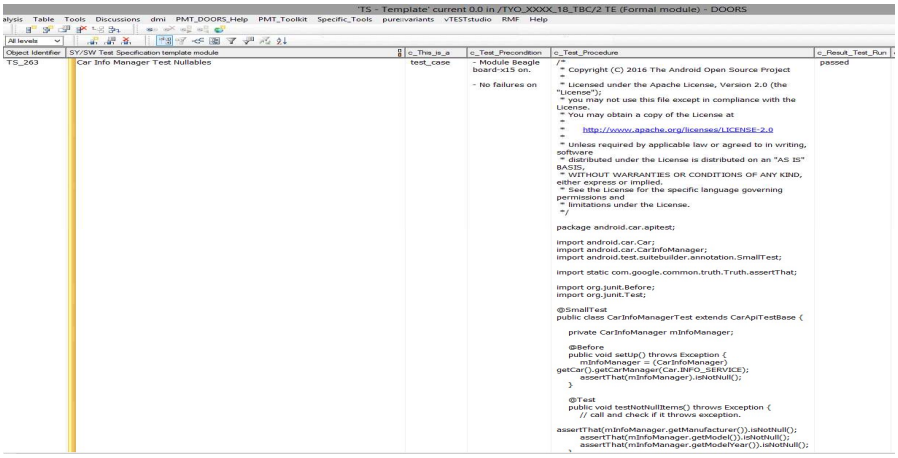



*Imagen 107. 10. Car Info Manager Test Nullables en DOORS.*

11. Car Info Manager Test Vehicle ID

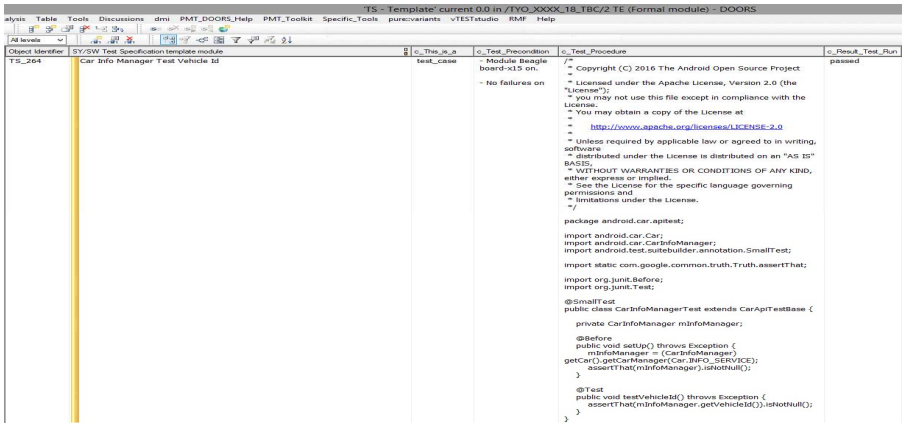



*Imagen 108. Car Info Manager Test Vehicle ID en DOORS.*

12. Car Bluetooth Test

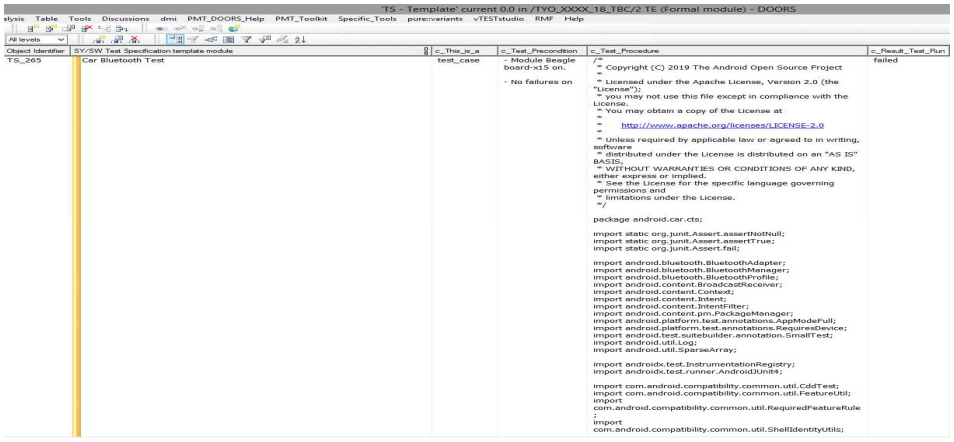



*Imagen 109. Car Bluetooth Test en DOORS.*

13. Car App Focus Manager Test Filter

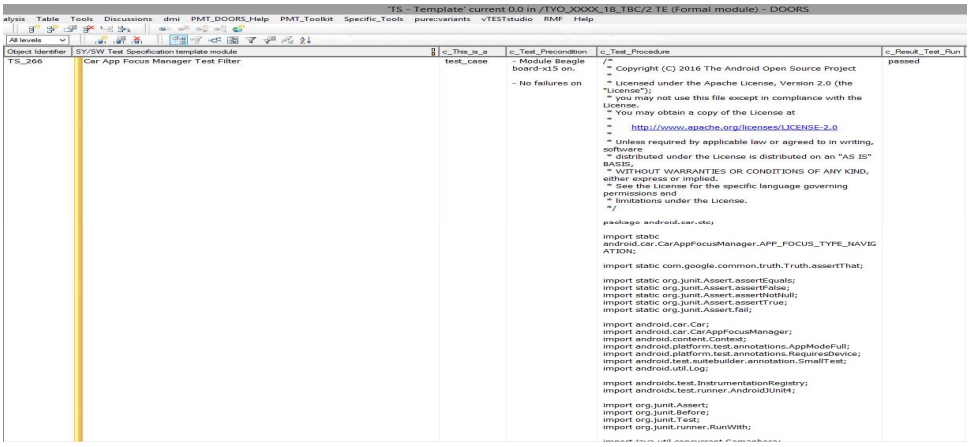



*Imagen 110. Car App Focus Manager Test Filter en DOORS.*

## 14. Car App Focus Manager Test Focus Change

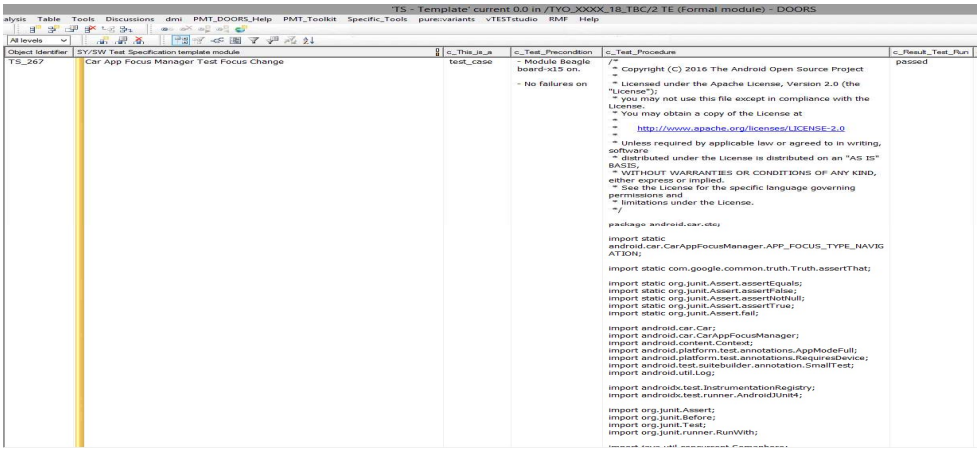



*Imagen 111. Car App Focus Manager Test Focus Change en DOORS.*

## 15. Car App Focus Manager Multiple Change Listeners Per Manager

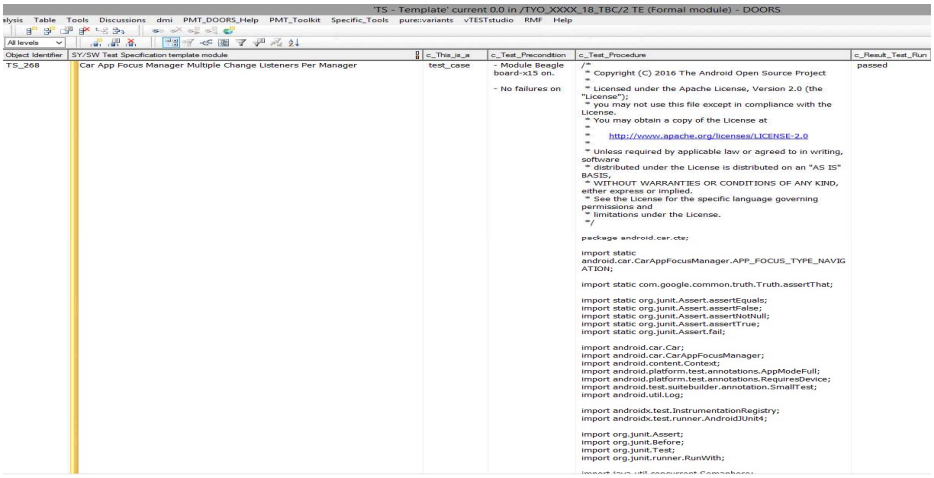



*Imagen 112. Car App Focus Manager Multiple Change Listeners Per Manager en DOORS.*

16. Car App Focus Manager Register Null

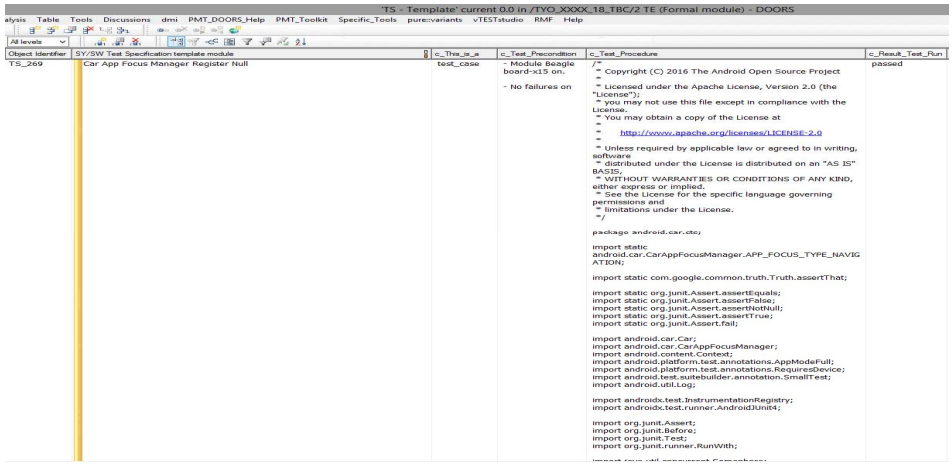



*Imagen 113. Car App Focus Manager Register Null en DOORS.*

17. Car App Focus Manager Register Unregister

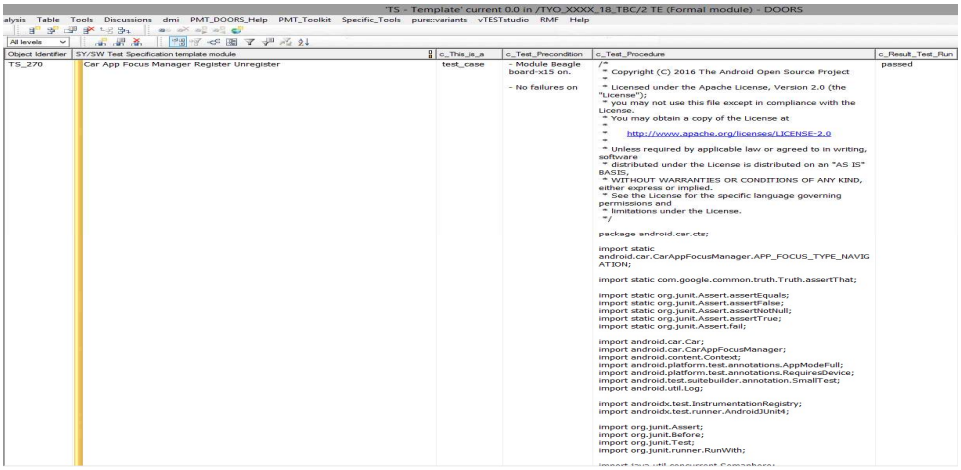



*Imagen 114. Car App Focus Manager Register Unregister en DOORS.*

18. Car App Focus Manager Set Active Null Listener

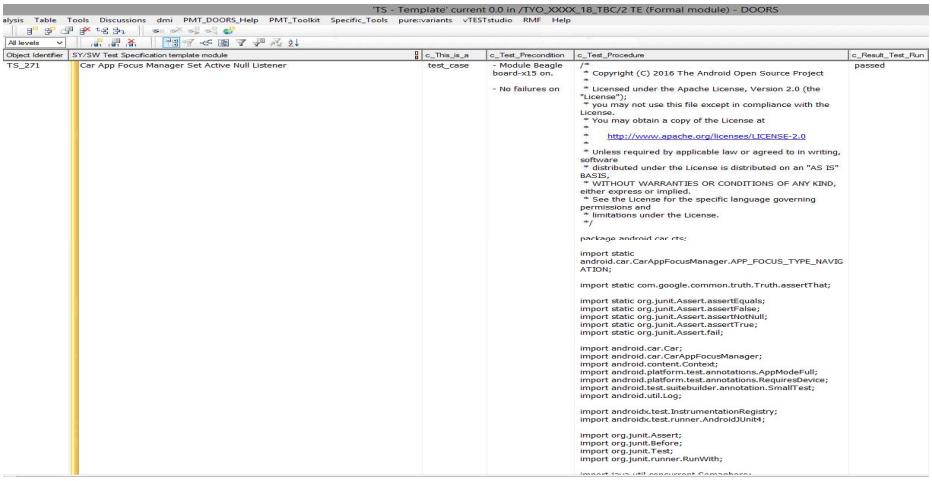



*Imagen 115. Car App Focus Manager Set Active Null Listener en DOORS.*

Manejo de casos de prueba en la base de datos. Estos están desplegados en dicha herramienta bajo un esquema preestablecido en para alguna unidad de negocio. Ahora, las pruebas en DOORS tienen diferentes "columnas" percibidas en las pruebas. Cada renglón tiene una parte de esas columnas o atributos.

Atributos de DOORS o columnas:

- Los atributos son características de un objeto.
- Se utilizan para contener información esencial de forma estructurada.

Evitando crear reglas no probadas se reúsan algunas ya preestablecidas para este ejercicio.

- This is a (c_This_is_a). Este atributo especifica el tipo de objeto: requisito_funcional, requisito_no_funcional, encabezado, comentario, recomendación, referencia, caso_de_prueba, gen_caso_de_prueba, conjunto_de_parámetros, resultado. Para este ejercicio es "test_case" o "caso_de_prueba".

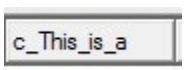



*Imagen 116. Vista de el atributo para clasificar que es un caso de prueba.*

- Test case precondition (c_Test_Precondition). Estado inicial del DUT.

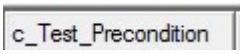



*Imagen 117. Vista de el atributo para clasificar las precondiciones de un caso de prueba.*

- Test case procedure (c_Test_Procedure). En definición de los eventos secuenciales que desencadenan alguna acción especificada. Están definidos los pasos de la prueba del caso específico.


*Imagen 118.Vista de el atributo para clasificar que es procedimiento de un caso de prueba.*

- Test case result test run (c_Result_Test_Run). Este atributo indica si la ejecución más reciente del caso de prueba fue exitosa (aprobada) o no (fallida). Si el caso de prueba no se ejecutó, el valor del atributo es not_done,

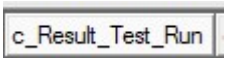



*Imagen 119.Vista de el atributo para clasificar los resultados de un caso de prueba.*

## D. LISTA COMPLETA DE ESPECIFICACIÓN DE PRUEBAS DEL CTS V.9.0

Nótese en la especificación 39 la CtsCarTest, pero hay un universo un tanto más grande hasta 322 especificaciones. No todas aplican a un sistema Automotriz, como el usado con esta aplicación.

1.-CtsAbiOverrideHostTestCases
2.-CtsAccelerationTestCases
3.-CtsAccessibilityServiceTestCases
4.-CtsAccessibilityTestCases
5.-CtsAccountManagerTestCases
6.-CtsActivityManagerDeviceSdk25TestCases
7.-CtsActivityManagerDeviceTestCases
8.-CtsAdminPackageInstallerTestCases
9.-CtsAdminTestCases
10.-CtsAlarmClockTestCases
11.-CtsAlarmManagerTestCases
12.-CtsAndroidAppTestCases
13.-CtsAndroidTestBase27ApiSignatureTestCases
14.-CtsAndroidTestMockCurrentApiSignatureTestCases
15.-CtsAndroidTestRunnerCurrentApiSignatureTestCases
16.-CtsAnimationTestCases
17.-CtsApacheHttpLegacy27ApiSignatureTestCases
18.-CtsApacheHttpLegacyCurrentApiSignatureTestCases
19.-CtsApacheHttpLegacyUsesLibraryApiSignatureTestCases
20.-CtsAppComponentFactoryTestCases
21.-CtsAppSecurityHostTestCases
22.-CtsAppTestCases
23.-CtsAppUsageHostTestCases
24.-CtsAppWidgetTestCases
25.-CtsAslrMallocTestCases
26.-CtsAssistTestCases
27.-CtsAtraceHostTestCases
28.-CtsAutoFillServiceTestCases
29.-CtsBackgroundRestrictionsTestCases
30.-CtsBackupHostTestCases
31.-CtsBackupTestCases
32.-CtsBatterySavingTestCases
33.-CtsBionicTestCases
34.-CtsBluetoothTestCases
35.-CtsBootStatsTestCases
36.-CtsCalendarcommon2TestCases
37.-CtsCameraApi25TestCases
38.-CtsCameraTestCases
39.-**CtsCarTestCases**
40.-CtsCarrierApiTestCases
41.-CtsColorModeTestCases
42.-CtsCompilationTestCases
43.-CtsContactsProviderWipe
44.-CtsContentTestCases
45.-CtsCppToolsTestCases
46.-CtsCurrentApiSignatureTestCases
47.-CtsDatabaseTestCases
48.-CtsDebugTestCases
49.-CtsDeqpTestCases
50.-CtsDeviceIdleHostTestCases
51.-CtsDevicePolicyManagerTestCases
52.-CtsDexMetadataHostTestCases
53.-CtsDisplayTestCases
54.-CtsDpiTestCases
55.-CtsDpiTestCases2
56.-CtsDreamsTestCases
57.-CtsDrmTestCases
58.-CtsDumpsysHostTestCases
59.-CtsDynamicLinkerTestCases
60.-CtsEdiHostTestCases
61.-CtsEffectTestCases
62.-CtsExternalServiceTestCases
63.-CtsExternalSourcesTestCases
64.-CtsFileSystemTestCases
65.-CtsFragmentTestCases
66.-CtsFragmentTestCasesSdk26
67.-CtsGestureTestCases
68.-CtsGpuToolsHostTestCases
69.-CtsGraphicsTestCases
70.-CtsHardwareTestCases
71.-CtsHarmfulAppWarningHostTestCases
72.-CtsHiddenApiBlacklistApi27TestCases
73.-CtsHiddenApiBlacklistCurrentApiTestCases
74.-CtsHiddenApiBlacklistDebugClassTestCases
75.-CtsHiddenApiKillswitchDebugClassTestCases
76.-CtsHiddenApiKillswitchWhitelistTestCases
77.-CtsHiddenApiKillswitchWildcardTestCases
78.-CtsHostTzDataTests
79.-CtsHostsideNetworkTests
80.-CtsHostsideNumberBlockingTestCases
81.-CtsHostsideTvTests
82.-CtsHostsideWebViewTests
83.-CtsIcuTestCases
84.-CtsIncidentHostTestCases
85.-CtsInlineMockingTestCases
86.-CtsInputMethodServiceHostTestCases
87.-CtsInputMethodTestCases
88.-CtsIntentSignatureTestCases

89.-CtsJankDeviceTestCases
90.-CtsJdwpSecurityHostTestCases
91.-CtsJdwpTestCases
92.-CtsJniTestCases
93.-CtsJobSchedulerSharedUidTestCases
94.-CtsJobSchedulerTestCases
95.-CtsJvmtiAttachingHostTestCases
96.-CtsJvmtiAttachingTestCases
97.-CtsJvmtiRedefineClassesHostTestCases
98.-CtsJvmtiRunTest1900HostTestCases
99.-CtsJvmtiRunTest1901HostTestCases
100.-CtsJvmtiRunTest1902HostTestCases
101.-CtsJvmtiRunTest1903HostTestCases
102.-CtsJvmtiRunTest1904HostTestCases
103.-CtsJvmtiRunTest1906HostTestCases
104.-CtsJvmtiRunTest1907HostTestCases
105.-CtsJvmtiRunTest1908HostTestCases
106.-CtsJvmtiRunTest1909HostTestCases
107.-CtsJvmtiRunTest1910HostTestCases
108.-CtsJvmtiRunTest1911HostTestCases
109.-CtsJvmtiRunTest1912HostTestCases
110.-CtsJvmtiRunTest1913HostTestCases
111.-CtsJvmtiRunTest1914HostTestCases
112.-CtsJvmtiRunTest1915HostTestCases
113.-CtsJvmtiRunTest1916HostTestCases
114.-CtsJvmtiRunTest1917HostTestCases
115.-CtsJvmtiRunTest1920HostTestCases
116.-CtsJvmtiRunTest1921HostTestCases
117.-CtsJvmtiRunTest1922HostTestCases
118.-CtsJvmtiRunTest1923HostTestCases
119.-CtsJvmtiRunTest1924HostTestCases
120.-CtsJvmtiRunTest1925HostTestCases
121.-CtsJvmtiRunTest1926HostTestCases
122.-CtsJvmtiRunTest1927HostTestCases
123.-CtsJvmtiRunTest1928HostTestCases
124.-CtsJvmtiRunTest1930HostTestCases
125.-CtsJvmtiRunTest1931HostTestCases
126.-CtsJvmtiRunTest1932HostTestCases
127.-CtsJvmtiRunTest1933HostTestCases
128.-CtsJvmtiRunTest1934HostTestCases
129.-CtsJvmtiRunTest1936HostTestCases
130.-CtsJvmtiRunTest1937HostTestCases
131.-CtsJvmtiRunTest1939HostTestCases
132.-CtsJvmtiRunTest1941HostTestCases
133.-CtsJvmtiRunTest1942HostTestCases
134.-CtsJvmtiRunTest1943HostTestCases
135.-CtsJvmtiRunTest902HostTestCases
136.-CtsJvmtiRunTest903HostTestCases
137.-CtsJvmtiRunTest904HostTestCases
138.-CtsJvmtiRunTest905HostTestCases
139.-CtsJvmtiRunTest906HostTestCases
140.-CtsJvmtiRunTest907HostTestCases
141.-CtsJvmtiRunTest908HostTestCases
142.-CtsJvmtiRunTest910HostTestCases
143.-CtsJvmtiRunTest911HostTestCases
144.-CtsJvmtiRunTest912HostTestCases
145.-CtsJvmtiRunTest913HostTestCases
146.-CtsJvmtiRunTest914HostTestCases
147.-CtsJvmtiRunTest915HostTestCases
148.-CtsJvmtiRunTest917HostTestCases
149.-CtsJvmtiRunTest918HostTestCases
150.-CtsJvmtiRunTest919HostTestCases
151.-CtsJvmtiRunTest920HostTestCases
152.-CtsJvmtiRunTest922HostTestCases
153.-CtsJvmtiRunTest923HostTestCases
154.-CtsJvmtiRunTest924HostTestCases
155.-CtsJvmtiRunTest926HostTestCases
156.-CtsJvmtiRunTest927HostTestCases
157.-CtsJvmtiRunTest928HostTestCases
158.-CtsJvmtiRunTest930HostTestCases
159.-CtsJvmtiRunTest931HostTestCases
160.-CtsJvmtiRunTest932HostTestCases
161.-CtsJvmtiRunTest940HostTestCases
162.-CtsJvmtiRunTest942HostTestCases
163.-CtsJvmtiRunTest944HostTestCases
164.-CtsJvmtiRunTest945HostTestCases
165.-CtsJvmtiRunTest947HostTestCases
166.-CtsJvmtiRunTest951HostTestCases
167.-CtsJvmtiRunTest982HostTestCases
168.-CtsJvmtiRunTest983HostTestCases
169.-CtsJvmtiRunTest984HostTestCases
170.-CtsJvmtiRunTest985HostTestCases
171.-CtsJvmtiRunTest986HostTestCases
172.-CtsJvmtiRunTest988HostTestCases
173.-CtsJvmtiRunTest989HostTestCases
174.-CtsJvmtiRunTest990HostTestCases
175.-CtsJvmtiRunTest991HostTestCases
176.-CtsJvmtiRunTest992HostTestCases
177.-CtsJvmtiRunTest993HostTestCases
178.-CtsJvmtiRunTest994HostTestCases
179.-CtsJvmtiRunTest995HostTestCases
180.-CtsJvmtiRunTest996HostTestCases
181.-CtsJvmtiRunTest997HostTestCases
182.-CtsJvmtiTaggingHostTestCases
183.-CtsJvmtiTrackingHostTestCases
184.-CtsKernelConfigTestCases
185.-CtsKeystoreTestCases
186.-CtsLeanbackJankTestCases
187.-CtsLegacyNotificationTestCases
188.-CtsLibcoreFileIOTestCases
189.-CtsLibcoreJsr166TestCases
190.-CtsLibcoreLegacy22TestCases
191.-CtsLibcoreOjTestCases
192.-CtsLibcoreOkHttpTestCases
193.-CtsLibcoreTestCases
194.-CtsLibcoreWycheproofBCTestCases
195.-CtsLibcoreWycheproofConscryptTestCases
196.-CtsLiblogTestCases
197.-CtsLocation2TestCases
198.-CtsLocationTestCases
199.-CtsLogdTestCases
200.-CtsMediaBitstreamsTestCases
201.-CtsMediaHostTestCases
202.-CtsMediaStressTestCases
203.-CtsMediaTestCases
204.-CtsMidiTestCases
205.-CtsMockingDebuggableTestCases
206.-CtsMockingTestCases
207.-CtsMonkeyTestCases
208.-CtsMultiUserHostTestCases
209.-CtsMultiUserTestCases
210.-CtsNNAPITestCases
211.-CtsNativeHardwareTestCases
212.-CtsNativeMediaAAudioTestCases
213.-CtsNativeMediaSlTestCases
214.-CtsNativeMediaXaTestCases
215.-CtsNativeNetTestCases
216.-CtsNdefTestCases
217.-CtsNetSecConfigAttributeTestCases
218.-CtsNetSecConfigBasicDebugDisabledTestCases
219.-CtsNetSecConfigBasicDebugEnabledTestCases
220.-CtsNetSecConfigBasicDomainConfigTestCases
221.-CtsNetSecConfigCleartextTrafficTestCases
222.-CtsNetSecConfigDownloadManagerTestCases
223.-CtsNetSecConfigInvalidPinTestCases
224.-CtsNetSecConfigNestedDomainConfigTestCases
225.-CtsNetSecConfigPrePCleartextTrafficTestCases
226.-CtsNetSecConfigResourcesSrcTestCases
227.-CtsNetSecPolicyUsesCleartextTrafficFalseTestCases
228.-CtsNetSecPolicyUsesCleartextTrafficTrueTestCases
229.-CtsNetSecPolicyUsesCleartextTrafficUnspecifiedTestCases
230.-CtsNetTestCases
231.-CtsNetTestCasesLegacyApi22
232.-CtsNetTestCasesLegacyPermission22
233.-CtsOmapiTestCases
234.-CtsOpenGLTestCases
235.-CtsOpenGlPerf2TestCases
236.-CtsOpenGlPerfTestCases
237.-CtsOsHostTestCases
238.-CtsOsTestCases
239.-CtsPdfTestCases
240.-CtsPerfettoTestCases
241.-CtsPermission2TestCases
242.-CtsPermissionTestCases
243.-CtsPreference2TestCases
244.-CtsPreferenceTestCases
245.-CtsPrintTestCases
246.-CtsProtoTestCases
247.-CtsProviderTestCases
248.-CtsRenderscriptLegacyTestCases
249.-CtsRenderscriptTestCases
250.-CtsRsBlasTestCases
251.-CtsRsCppTestCases
252.-CtsSampleDeviceTestCases
253.-CtsSampleHostTestCases
254.-CtsSaxTestCases
255.-CtsSeccompHostTestCases
256.-CtsSecureElementAccessControlTestCases1
257.-CtsSecureElementAccessControlTestCases2
258.-CtsSecureElementAccessControlTestCases3
259.-CtsSecurityBulletinHostTestCases
260.-CtsSecurityHostTestCases
261.-CtsSecurityTestCases
262.-CtsSelinuxTargetSdk25TestCases
263.-CtsSelinuxTargetSdk27TestCases
264.-CtsSelinuxTargetSdkCurrentTestCases
265.-CtsSensorTestCases
266.-CtsShortcutHostTestCases
267.-CtsShortcutManagerTestCases
268.-CtsSimRestrictedApisTestCases
269.-CtsSimpleCpuTestCases
270.-CtsSimpleperfTestCases
271.-CtsSkQPTestCases
272.-CtsSliceTestCases
273.-CtsSpeechTestCases
274.-CtsStatsdHostTestCases
275.-CtsSustainedPerformanceHostTestCases
276.-CtsSyncAccountAccessOtherCertTestCases
277.-CtsSyncContentHostTestCases
278.-CtsSyncManagerTestsCases
279.-CtsSystemApiAnnotationTestCases
280.-CtsSystemApiSignatureTestCases
281.-CtsSystemIntentTestCases
282.-CtsSystemUiHostTestCases
283.-CtsSystemUiTestCases
284.-CtsTelecomTestCases
285.-CtsTelecomTestCases2
286.-CtsTelecomTestCases3
287.-CtsTelephony2TestCases
288.-CtsTelephonyTestCases
289.-CtsTextTestCases
290.-CtsThemeDeviceTestCases
291.-CtsThemeHostTestCases
292.-CtsToastLegacyTestCases
293.-CtsToastTestCases
294.-CtsTransitionTestCases
295.-CtsTrustedVoiceHostTestCases
296.-CtsTvProviderTestCases
297.-CtsTvTestCases
298.-CtsUiAutomationTestCases
299.-CtsUiDeviceTestCases
300.-CtsUiRenderingTestCases
301.-CtsUidIsolationTestCases
302.-CtsUsageStatsTestCases
303.-CtsUsbTests
304.-CtsUtilTestCases
305.-CtsVideoTestCases
306.-CtsViewTestCases
307.-CtsVmTestCases
308.-CtsVoiceInteractionTestCases
309.-CtsVoiceSettingsTestCases
310.-CtsVrTestCases
311.-CtsWebkitTestCases
312.-CtsWidgetTestCases
313.-CtsWindowManagerDeviceTestCases
314.-CtsWrapNoWrapTestCases
315.-CtsWrapWrapDebugMallocDebugTestCases
316.-CtsWrapWrapDebugTestCases
317.-CtsWrapWrapNoDebugTestCases
318.-cts-system-all.api
319.-signed-CtsSecureElementAccessControlTestCases1
320.-signed-CtsSecureElementAccessControlTestCases2
321.-signed-CtsSecureElementAccessControlTestCases3
322.-vm-tests-tf

# Bibliografía


[1] B. Berk, «The Unending Struggle to Make Your Car Feel Like Your Phone,» 13 05 2017. [En línea]. Available: https://www.wired.com/2017/05/unending-struggle-make-car-feel-like-phone/. [Último acceso: 09 06 2023].

[2] «Global Market Insights,» [En línea]. Available: https://www.gminsights.com/industry-analysis/software-testing-market. [Último acceso: 12 06 2023].

[3] T. Hoff, «latency-everywhere-and-it-costs-you-sales-how-crush-it,» 25 July 2009. [En línea]. Available: http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it. [Último acceso: 09 06 2023].

[4] D. Sillars, High Performance Android Apps, Sebastopol: O'Reilly, 2015.

[5] J. P. I. Q. Study, 2020. [En línea]. Available: https://www.jdpower.com/business/press-releases/2020-initial-quality-study-iqs. [Último acceso: 09 06 2023].

[6] DIGIBYTE, «eCall in all new cars from April 2018,» 12 12 2017. [En línea]. Available: https://digital-strategy.ec.europa.eu/en/news/ecall-all-new-cars-april-2018. [Último acceso: 09 06 2023].

[7] Lantronix, Inc, «www.lantronix.com,» [En línea]. Available: https://www.lantronix.com/. [Último acceso: 09 06 2023].

[8] Qualcomm Technologies, Inc, «Qualcomm Technologies, Inc,» [En línea]. Available: https://www.qualcomm.com/home. [Último acceso: 09 06 2023].

[9] Lantronix, Inc, «s820am-v2-automotive-development-platform,» [En línea]. Available: https://www.lantronix.com/products/s820am-v2-automotive-development-platform/. [Último acceso: 09 06 2023].

[10] Lantronix, Inc, «lantronix completes acquisition intrinsyc expanding iot value proposition,» 16 January 2020. [En línea]. Available: https://www.lantronix.com/newsroom/press-releases/lantronix-completes-acquisition-intrinsyc-expanding-iot-value-proposition/. [Último acceso: 09 06 2023].

[11] Texas Instruments Incorporated, «Texas Instruments Incorporated,» [En línea]. Available: https://www.ti.com/. [Último acceso: 09 06 2023].

[12] beagleboard.org, «beagleboard.org,» [En línea]. Available: https://beagleboard.org/. [Último acceso: 09 06 2023].

[13] Instruments, Texas, «Enabling Android Automotive on Your TI Development,» August 2019. [En línea]. Available:

https://www.ti.com/lit/an/spraco0/spraco0.pdf?ts=1628139615436&ref_url=https%253A%252F%252Fwww.google.com%252F. [Último acceso: 09 06 2023].

[14] google LLC, «Requirements,» 2020. [En línea]. Available: https://source.android.com/setup/build/requirements. [Último acceso: 09 06 2023].

[15] Texas Instruments, «PROCESSOR-SDK-ANDROID-AM57X 06_03_00_106,» 20 Abril 2020. [En línea]. Available: http://software-dl.ti.com/processor-sdk-android/esd/AM57X/latest/index_FDS.html. [Último acceso: 09 06 2023].

[16] BeagleBoard.org, «BeagleBoard-X15_Quick-Start-Guide,» July 2017. [En línea]. Available: https://github.com/beagleboard/beagleboard-x15/blob/master/BeagleBoard-X15_Quick-Start-Guide.pdf. [Último acceso: 09 06 2023].

[17] google, LLC, «Cómo probar apps de Android para vehículos,» [En línea]. Available: https://developer.android.com/training/cars/testing?hl=es-419#system-images. [Último acceso: 09 06 2023].

[18] A. L. S. N. P. Raghavan, Embedded Linux System Design and Development, Boca Raton, FL: Auerbach Publications, 2006, p. 41.

[19] C. Simmonds, «The embedded Linux Quick Start Guide, Kernel and user space,» de *Embedded Linux Conference Europe 2010*, 2010.

[20] S. Solis, «The Android Booting Process,» 10 10 2014. [En línea]. Available: https://community.nxp.com/t5/i-MX-Processors-Knowledge-Base/The-Android-Booting-process/ta-p/1129182.

[21] elinux.org, «elinux.org,» 24 Julio 2015. [En línea]. Available: https://elinux.org/Android_Kernel_Features. [Último acceso: 09 06 2023].

[22] T. Bird, «A_note_about_overhead,» 22 06 2010. [En línea]. Available: https://elinux.org/Using_Bootchart_on_Android#A_note_about_overhead. [Último acceso: 09 06 2023].

[23] google, LLC, «Optimización de los tiempos de arranque,» [En línea]. Available: https://source.android.com/docs/core/perf/boot-times?hl=es-419. [Último acceso: 09 06 2023].

[24] Texas Instruments Incorporated, «Android Boot Optimization on DRA7xx Devices - Application Report,» Febrero 2018. [En línea]. Available: https://www.ti.com/lit/an/sprac30a/sprac30a.pdf?ts=1685291687517. [Último acceso: 09 06 2023].

[25] Texas instruments Incorporated, «Linux Boot Time Optimizations on DRA7xx Devices,» Marzo 2017. [En línea]. Available: https://www.ti.com/lit/an/sprac82/sprac82.pdf?ts=1610981409976&. [Último acceso: 06 09 2023].

[26] Google LLC, «MediaBrowserService,» 2020. [En línea]. Available: https://developer.android.com/reference/android/service/media/MediaBrowserService. [Último acceso: 09 06 2023].

[27] exoplayer.dev, «MediaSessionConnector,» 2021. [En línea]. Available: https://exoplayer.dev/doc/reference/com/google/android/exoplayer2/ext/mediasession/MediaSessionConnector.html. [Último acceso: 09 06 2023].

[28] techpad, [En línea]. Available: https://techpad.mx/wp-content/uploads/2019/09/0202Ficha-Tecnica-3GR-18.pdf. [Último acceso: 12 06 2023].

[29] «Compatibility Test Suite downloads,» [En línea]. Available: https://source.android.com/docs/compatibility/cts/downloads. [Último acceso: 09 06 2023].

[30] [En línea]. Available: https://software-dl.ti.com/processor-sdk-android/esd/docs/latest/android/Overview_Getting_Started_Guide.html. [Último acceso: 12 06 2023].

[31] «iteso_mde_project,» [En línea]. Available: https://gitlab.com/iteso_mde_project/iteso_mde_project/-/blob/master/AAOSP_Planning.svg. [Último acceso: 12 06 2023].

[32] «iteso_mde_project,» [En línea]. Available: https://gitlab.com/iteso_mde_project/iteso_mde_project/-/blob/master/PlantUml/Gantt.txt.

[33] VDA QMC Working Group 13 / Automotive SIG, «AutomotiveSPICE_PAM_31,» 01 11 2017. [En línea]. Available: https://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf. [Último acceso: 09 06 2023].

[34] «Agile611.com,» Agile 611, [En línea]. Available: https://www.agile611.com/como-podemos-hacer-buenas-user-stories-para-ayudar-al-equipo-de-desarrollo-y-al-product-owner/?v=0b98720dcb2c#:~:text=Una%20user%20story%20es%20una,requisitos%20del%20sistema%20de%20software. [Último acceso: 14 11 2023].

[35] C. AG, «Continental AG,» Continental AG, [En línea]. Available: https://www.continental.com/en/products-and-innovation/innovation/connectivity/high-performance-computer/. [Último acceso: 12 06 2023].

[36] elinux.org, «Android_Binder,» 04 Octubre 2021. [En línea]. Available: https://elinux.org/Android_Binder. [Último acceso: 09 06 2023].