Universidad de las Ciencias Informáticas

Facultad 3

Aplicación para dispositivos móviles del módulo de Planificación en el SIPAC

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Adrian Pang Rojas

Tutores:

Ing. Claudia Bravo Batista

Dr.C Marieta Peña Abreu

La Habana, junio de 2018

"Año 60 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____.

_____

Adrian Pang Rojas

_____ _____

Ing. Claudia Bravo Batista Dr.C Marieta Peña Abreu

AGRADECIMIENTOS

Aquí en la UCI pase unos de los mejores años de mi vida y es porque aquí he aprendido casi todo lo que se y no solo de informática, aquí además hice cosas por impulso, abracé para proteger, me decepcioné con algunas personas, y también yo decepcioné a alguien, me enamoré por una sonrisa, amé y fui amado, pero también rechacé y fui rechazado, fui amado y no supe amar, hice amigos eternos, grité y salté de felicidad, me caí y me levanté una y otra vez, ¡pero sobreviví!, ¡y todavía sigo vivo! , porque los problemas no son eternos, siempre tienen solución, lo único que no se resuelve es la muerte, y hay que ser fuertes y levantarse de los tropiezos que nos pone la vida, para avisarnos que después de un túnel oscuro y lleno de soledad, vienen cosas muy buenas

Por eso aprendí a ir a la lucha con determinación abrazar la vida y vivirla con pasión perder con clase y vencer con osadía porque el mundo pertenece a quien se atreve y la vida es mucho más para ser insignificante.

Y quiero agradecerles a todas esas personas que formaron parte de mi vida estos cinco años y estuvieron a mi lado en un momento u otro siempre para estrecharme una mano y ayudarme en lo que pudieron, pero en especial agradecer:

A *mis padres por todo su esfuerzo y entrega, que hicieron posible terminar mis estudios.*

A *Ariadna por toda su preocupación, ayuda y disposición incondicional.*

A *Cari por todo su ilimitado cariño y bondad.*

A *mi tutora por su preocupación, por sus palabras siempre de aliento y su ayuda, sin ella no hubiese podido terminar.*

A *el tribunal por todas sus recomendaciones y señalamientos.*

A *mi gente de la escalera y la pala, el lobby del edificio va a cerrar así que no se pierdan, los quiero.*

A *Luis y el Bati que fueron piezas fundamentales para mi vida en esta escuela gracias mis hermanos por todo los quiero y les debo el mundo y más no dejen de contar conmigo para lo que sea.*

DEDICATORIA

A *mis padres por ser para mí, el mayor ejemplo de constancia, dedicación y amor, por ser mi fortaleza e inspiración diaria, ha sido y es, un privilegio ser su hijo, gracias por darme siempre lo mejor que pudieron, gracias mamá, gracias papá, ahora me toca a mí.*

A *mi pequeña hermanita que es mi nuevo tesoro.*

RESUMEN

El sistema de Planificación de Actividades (SIPAC) forma parte del paquete de soluciones integrales de gestión CEDRUX para las entidades presupuestadas y empresariales, el cual está basado en los principios de independencia tecnológica y con funcionalidades generales de los procesos y las particularidades de la economía cubana. Sus características funcionales fueron diseñadas de acuerdo con las políticas emitidas por los organismos rectores del Estado Cubano. Incluye el módulo de Planificación que permite ejecutar la planificación de actividades basadas en reglas de la compartimentación de la información, permitiendo que la información planificada sea accedida por la persona autorizada, en el momento indicado. La presente investigación tiene como objetivo el desarrollo de una Aplicación para dispositivos móviles que permita brindar y consumir los servicios REST (por sus siglas en inglés de Representational State Transfer) del módulo de Planificación de SIPAC, para mejorar la portabilidad del sistema.

Para guiar el proceso de desarrollo de la aplicación se utilizó como metodología AUP-UCI, Kotlin y XML como lenguajes de programación y Android Studio 3.0.1 como Entorno de Desarrollo Integrado (IDE en sus siglas en inglés). Se definieron los requisitos de la aplicación, se realizó el análisis y diseño garantizando la correcta implementación. Se ejecutaron las pruebas unitarias utilizando la herramienta JUnit. Se validó la investigación utilizando la norma ISO/IEC 25023 que define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.

Palabras Claves: dispositivos móviles**,** planificación, portabilidad, servicios REST, SIPAC.

ABSTRACT

The Activity Planning System (SIPAC) is part of the CEDRUX comprehensive management solutions package for budgeted and corporate entities, which is based on the principles of technological independence and with general functionalities of the processes and the characteristics of the Cuban economy. ts functional characteristics were designed in accordance with the policies issued by the governing bodies of the Cuban State. It includes the Planning module that allows executing the planning of activities based on rules for the compartmentalization of information, allowing the planned information to be accessed by the authorized person, at the indicated time. The objective of this research is the development of an Application for mobile devices that allows to provide and consume the REST services (for its acronym in English of Representational State Transfer) of the Planning module of SIPAC, to improve the portability of the system.

To guide the development process, the application was used as methodology AUP-UCI, Kotlin and XML as programming languages and Android Studio 3.0.1 as Integrated Development Environment (IDE in its acronym in English). The requirements of the application were defined, the analysis and design was carried out guaranteeing the correct implementation. The unit tests were executed using the JUnit tool. The research was validated using the ISO / IEC 25023 standard that specifically defines the metrics for measuring the quality of software products and systems.

Keywords: mobile devices, planning, portability, REST services, Android, SIPAC.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La Instrucción no.1 del Presidente de los Consejos de Estado y de Ministros para la Planificación de los objetivos y actividades en los órganos, Organismos de la Administración Central de Estado, entidades nacionales y Administraciones locales del Poder Popular, constituye el documento rector de la planificación en Cuba. La misma, tiene como objetivo establecer el procedimiento para llevar a cabo el proceso de planificación del Gobierno, que permita dar cumplimiento a los acuerdos y resoluciones aprobadas en el VI Congreso del Partico Comunista de Cuba, las decisiones de la Asamblea Nacional del Poder Popular, el consejo de Ministros y la actualización de los planes de la economía.

El Sistema de Planificación de Actividades (SIPAC) se basa en la Instrucción no.1 del Presidente de los Consejos de Estado y de Ministros. Forma parte del paquete de soluciones integrales de gestión Xedro para las entidades presupuestadas y empresariales, el cual está basado en los principios de independencia tecnológica y con funcionalidades generales de los procesos y las particularidades de la economía cubana. Sus características funcionales fueron diseñadas de acuerdo con las políticas emitidas por los organismos rectores del Estado Cubano. El sistema pone al servicio de su entidad las potencialidades de la tecnología informática y provee facilidades para la integración de las diferentes áreas productivas y departamentos administrativos, permite interrelacionar objetivos de trabajo y actividades en tiempo real; garantizando el seguimiento del desarrollo y cumplimiento de los objetivos y tareas principales en las organizaciones.

Incluye los módulos encargados de generar las configuraciones necesarias que otorgan al sistema y al cliente una simulación de los procesos de organización del personal, así como los niveles de subordinación necesarios e indispensables. Para efectuar una planificación de actividades basadas en reglas de la compartimentación de la información, permitiendo que la información planificada sea accedida por la persona autorizada, en el momento indicado. El módulo Planificación contiene las operaciones a realizar con los documentos de la planificación, entiéndase: Planes, Objetivos, Actividades, Áreas de Resultados Clave (ARC), Factores que Intervienen en la Planificación (FIP).

Los desarrolladores de software coinciden en que la portabilidad es un atributo deseable para las aplicaciones informáticas que se construyen. Rick Kazman, Paul Clements y Mark Klein definen la portabilidad como “la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos” (Clements, y otros, 2001). Actualmente con la tendencia hacia el desarrollo de tecnologías móviles, permite que las aplicaciones se puedan ejecutar en

diferentes plataformas y entornos, creando diversos ambientes de interacción por parte de los usuarios.

Un dispositivo móvil se puede definir como un aparato de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, que ha sido diseñado específicamente para una función, pero que puede llevar a cabo otras funciones más generales. De acuerdo con esta definición existen multitud de dispositivos móviles, desde los reproductores de audio portátiles hasta los navegadores GPS, pasando por los teléfonos móviles, los PDAs o los Tablet PCs (Alonso, y otros, 2016).

Actualmente las operaciones que se realizan sobre el SIPAC con los elementos de planificación, impiden que el sistema pueda ser ejecutado en diferentes ambientes computacionales como los dispositivos móviles. Los múltiples usuarios que hacen uso del SIPAC deben tener disponible una computadora para realizar las diferentes actividades de planificación, dígase visualizar y administrar la información, dificultando que la misma se encuentre disponible en diferentes espacios y lugares.

El SIPAC tiene la capacidad para ser instalado en sistemas operativos como Windows 7 o superior y Linux en versión 16.04 o superior. Además, puede ser adaptado a diversos entornos de hardware y software, como una PC con características de Procesador Intel(R) Core(TM) i3-2120, 4GB de RAM y sistema operativo de 64 bits. Sin embargo, no puede ser instalado en otros ambientes computacionales como los dispositivos móviles, debido a sus características tecnológicas.

Actualmente, para realizar las operaciones sobre el SIPAC con los elementos de planificación, dígase visualizar y administrar la información desde un dispositivo móvil. Se deben realizar actividades como: utilizar un método de conexión, por ejemplo, Wi-Fi; acceder desde el navegador web a la ruta donde se encuentra disponible la aplicación en el servidor web. Lo que implica, que la conectividad sea estable y el SIPAC se encuentre disponible en el espacio y lugar de acceso. Sin embargo, el sistema no se encuentra diseñado, para gestionar la pérdida de la conexión. Además, que se puede acceder solamente en el espacio y lugar, que esté disponible.

A partir de la situación antes descrita se ha identificado el siguiente **problema a resolver:** ¿Cómo mejorar la portabilidad de los elementos de planificación del Sistema de Planificación de Actividades?

El **objeto estudio**, Aplicaciones para dispositivos móviles, estableciendo como **campo de acción**, Aplicaciones para dispositivos móviles, que realicen las operaciones sobre los elementos de la planificación.

Para darle solución al problema planteado se define como **objetivo general** desarrollar una aplicación para dispositivos móviles que permita brindar y consumir los servicios REST del módulo de Planificación, para mejorar la portabilidad de los elementos de planificación.

Para guiar la investigación se definen los siguientes **objetivos específicos:**

- Construir el marco teórico conceptual de la investigación sobre las aplicaciones para dispositivos móviles que realicen las operaciones sobre los elementos de la planificación.
- Análisis y diseño de una aplicación para dispositivos móviles que permita brindar y consumir los servicios REST del módulo de Planificación.
- Implementación de una aplicación para dispositivos móviles que permita brindar y consumir los servicios REST del módulo de Planificación.
- Validar la solución aplicación para dispositivos móviles que permita brindar y consumir los servicios REST del módulo de Planificación.

Se plantea como **idea a defender:** si se desarrolla una aplicación para dispositivos móviles que permite brindar y consumir servicios REST del módulo de Planificación, entonces se mejorará la portabilidad de los elementos de la planificación de SIPAC.
Los resultados de la investigación están centrados en la documentación técnica y la aplicación para dispositivos móviles que permita brindar y consumir los servicios REST del módulo de Planificación.
Con la misión de obtener conocimientos necesarios que hagan posible la materialización del objetivo general, se han utilizado diferentes tipos de métodos de investigación teóricos y empíricos, los cuales se describen a continuación:

Métodos teóricos

Histórico-lógico: este método ha sido aplicado en la búsqueda realizada de la documentación sobre las aplicaciones móviles que permiten realizar configuraciones en los elementos de la planificación. Además, de los diferentes servicios REST que se implementan.

Analítico-sintético: este método ha sido aplicado en el análisis de la documentación identificada, para extraer los elementos que propicien la solución a la problemática planteada, así como la síntesis de los elementos necesarios para la selección de las tecnologías y metodologías adecuadas para el desarrollo de la aplicación Android propuesta.

Métodos empíricos

Observación: La observación científica como método consiste en la percepción directa del objeto de investigación. Este método ha sido aplicado en la observación de la apariencia de las páginas web al tamaño de la pantalla de los diferentes dispositivos. Es necesario lograr un diseño que la vista de la página sea visualizada correctamente con independencia del

tamaño de pantalla. De esta forma, al diseñar una única vista que se adapte dinámicamente, permite reducir tiempo de producción.

Entrevista no estructurada o abierta: Obteniendo información para la construcción simultánea de la aplicación a partir de las respuestas del entrevistado, necesitando el entrevistador una gran cantidad de documentación y preparación.

El presente trabajo de diploma está compuesto por 3 capítulos, estructurados de la siguiente manera:

Capítulo 1. Fundamentación Teórica de la Aplicación

En este capítulo se realiza la elaboración del marco teórico donde se exponen los conceptos asociados a la solución de la problemática y las diferentes soluciones existentes a nivel mundial. También se describen y caracterizan la metodología, herramientas y tecnologías a utilizar para el desarrollo de la aplicación Android.

Capítulo 2. Descripción de la aplicación

Se especifica el Modelo Conceptual para identificar los principales conceptos del negocio. Se elabora los artefactos generados por la metodología seleccionada. Se modelan y detallan los diagramas que representan las funcionalidades del sistema, a partir de los patrones de diseño identificados.

Capítulo 3. Implementación, prueba y validación de la aplicación

Se muestra la distribución física de los distintos componentes lógicos desarrollados, a través del modelo de despliegue y la organización del sistema mediante el modelo de componentes. Se explican los estilos de programación y estándares de codificación empleados y por último se valida el sistema desarrollado aplicándole las pruebas necesarias para demostrar que la solución es correcta.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA DE LA APLICACIÓN

Introducción

En el presente capítulo son abordados los principales conceptos utilizados en la investigación. Se describen las principales tecnologías, lenguajes de programación y herramientas definidas para el desarrollo de la solución, así como la metodología de desarrollo, los patrones a emplear y las métricas para la validación del diseño.

1.1 Principales conceptos

La planificación constituye una herramienta que facilita la organización del trabajo para el funcionamiento y labor de los vicepresidentes del Consejo de Ministros y de los organismos de la Administración del Estado, de los consejos de la Administración Provincial y de las entidades del país, y exige la participación de todos para armonizar y lograr cumplir lo planteado por el Partido, el Estado y el Gobierno a las diferentes esferas.

1.1.1 Plan de Actividades Principales del Gobierno con sus anexos

El Plan de Actividades Principales del Gobierno con sus anexos, se elabora sobre la base de las principales actividades del PCC, las propuestas de los vicepresidentes del Consejo de Ministros, los órganos y organismos que son atendidos directamente, así como las entidades del Estado y del Gobierno, en el formato establecido en el Modelo No. 1. Las actividades del resto de los miembros del Consejo de Ministros a incluir en el referido plan y en sus anexos, serán tramitadas mediante los vicepresidentes que los atienden.

Como anexo al Plan de Actividades Principales del Gobierno se elabora:

Plan de Temas (Modelo No. 2), para organizar las reuniones del Consejo de Ministros y los Consejos de Dirección de cada nivel, sobre la base de las políticas trazadas por el Partido, el Estado, el Gobierno y de sus resoluciones y acuerdos, así como de otros temas que surjan durante el desarrollo del trabajo.

Para conformar el referido plan, los vicepresidentes y los ministros que son atendidos directamente enviarán al Secretario del Consejo de Ministros sus propuestas, las cuales serán analizadas y una vez coordinadas con los proponentes se incluirán en el anexo del plan de trabajo anual.

Los temas comprendidos en el referido plan anual deben ser elaborados con tiempo de antelación, que permita que todos los participantes en el Consejo de Ministros posean el documento correspondiente con no menos de 8 días de antelación a la reunión.

Los temas que se presentan por un organismo de la Administración Central del Estado en que esté implicado otro, deberá ser conciliado oportunamente con este, antes de su presentación.

Principales actos, eventos y otros (Modelo No. 3), deberá contener los actos conmemorativos y eventos recogidos en el anexo del Plan de trabajo del Comité Central del PCC en que participa el Gobierno. Para ello, se seguirán los mismos procedimientos que con el Plan de Actividades Principales del Gobierno.

Las salidas al exterior aprobadas de los miembros del Consejo de Ministros y de otro personal que se designe, serán recogidas en el Modelo No. 4.

Las visitas que se realizarán en el año (Modelo No. 5) serán presentadas al órgano de planificación del CECM con el objetivo de conformar el anexo de las visitas del Plan de Actividades Principales del Gobierno.

Los cursos (Modelo No. 6) que se impartirán cada año por el Partido, el Estado y la Defensa, que tienen incidencia en el Consejo de Ministros, serán presentadas al órgano de planificación del CECM por las instancias correspondientes.

Los principales documentos legislativos (Modelo No. 7) que serán analizados en el año por la Asamblea Nacional del Poder Popular y los consejos de Estado y de Ministros.

El Plan de Actividades Principales del Gobierno con sus anexos conciliado, se somete a la aprobación previa del Presidente de los Consejo de Estado y de Ministros y se lleva a la aprobación del Consejo de Ministros.

Plan de Trabajo Anual

En el Plan de Trabajo Anual se instrumentan las tareas y aseguramientos para dar cumplimiento a las actividades y tareas previstas en el Plan de Actividades Principales del Gobierno con sus anexos.

Para conformar este plan, se seguirán los mismos procedimientos que para el Plan de Actividades Principales del Gobierno con sus anexos, con la diferencia de que cada nivel de dirección, deberá determinar los factores a los cuales se les consulta y circula este plan.

El formato para su elaboración, será igual que el del Plan de Actividades Principales del Gobierno, (Modelo No. 1).

Como parte del proceso de elaboración del plan de trabajo anual, se confecciona el Gráfico de planificación, conciliación y coordinación de las actividades (Modelo No. 8); este documento recogerá las actividades que se desarrollarán y los aseguramientos correspondientes, ayudará a la racional planificación y distribución de las tareas y será una herramienta útil en la conciliación y coordinación de las actividades, así como en sus aseguramientos y se empleará durante todo el año que se planifica.

Plan de Trabajo Mensual

En el Plan de Trabajo Mensual (Modelo No. 9) se reflejan todas las actividades previstas en el plan de trabajo anual para ese mes, en correspondencia con la puntualización trimestral y mensual que se reciba del nivel superior. Cada nivel determinará la inclusión en su plan de

las nuevas tareas que surjan como parte del proceso de dirección que no fueron previstas o que surja la necesidad de hacerlas.

Se realizará una evaluación cualitativa de los principales resultados obtenidos por cada nivel de dirección. El Plan de trabajo mensual no se aprueba y se archiva por un mes.

Plan de Trabajo Individual

El plan de trabajo individual (Modelo No. 10) se elabora por cada cuadro, funcionario y especialista, para cada mes; en él se recogen las tareas y aseguramientos que cada cual debe realizar, que posibilite dar cumplimiento al plan mensual del nivel que planifica. Este lo presenta a la aprobación con un análisis cuantitativo de las principales tareas cumplidas (Presidente de los Consejos de Estado y de Ministros 2009).

1.1.2 Portabilidad

La portabilidad es la capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro. Esta característica se subdivide a su vez en las siguientes subcaracterísticas:

Adaptabilidad: Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.

Capacidad para ser instalado: Facilidad con la que el producto se puede instalar o desinstalar de forma exitosa en un determinado entorno.

Capacidad para ser reemplazado: Capacidad del producto para ser utilizado en lugar de otro producto de software determinado con el mismo propósito y el mismo entorno (Norma ISO 25000, 2004).

1.1.3 Servicios REST

REST (Representational State Transfer) es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. REST permite crear servicios compatibles con cualquier dispositivo o cliente HTTP. Ninguna información de contexto del cliente se almacena en el servidor, cada petición del cliente contiene toda la información necesaria para que el servidor sea capaz de preparar la respuesta. El cliente puede querer cachear las respuestas, por lo que estas deben ser cacheables. Es decir, debe incluirse en los headers de las respuestas la cabecera Last-Modified y soportar la recepción de la cabecera If-Unmodified-Since en las peticiones GET del cliente.

En el extremo del servidor, el estado y la funcionalidad de la aplicación se dividen en recursos. Un recurso es un elemento de interés, una identidad conceptual que se expone a los clientes. Cada recurso es de acceso único a través de una URI (Universal Resources Identifier - identificador de recurso universal). Todos los recursos comparten una interfaz uniforme para la transferencia de estados entre el cliente y servidor. Se usan métodos estándar HTTP como GET, PUT, POST y DELETE.

REST contiene una interfaz uniforme de identificación de recursos, manipulación de recursos mediante su representación, mensajes auto descriptivos, hipermedia como motor de estado. El servidor debe seguir una arquitectura de capas, en la que cada capa conoce únicamente la capa inmediatamente inferior. De esta forma, la capa superior (interfaz de interacción) abstrae el comportamiento interno. El cliente no debe conocer nada sobre la implementación del servidor. Se provee al cliente con código capaz de manejar la respuesta del servidor. El mejor ejemplo de código en demanda son los scripts de JavaScript que contienen las páginas web.

Diseñar un servicio Web basado en REST

1. Identificar todas las entidades conceptuales que se desean exponer como servicio.
2. Crear una URL para cada recurso. Los recursos deberían ser nombres no verbos(acciones).
3. Categorizar los recursos si los clientes pueden obtener una representación del recurso o si pueden modificarlo. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, se debe hacer los recursos accesibles mediante HTTP POST, PUT y/o DELETE.
4. Todos los recursos accesibles mediante GET no deben tener efectos secundarios. Es decir, los recursos deben devolver la representación del recurso. Por tanto, invocar al recurso no debe ser resultado de modificarlo.
5. Ninguna representación debe estar aislada. Es decir, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información.
6. Especificar el formato de los datos de respuesta mediante un esquema (DTD, W3C Schema). Para los servicios que requieren un POST o un PUT se recomienda proporcionar un esquema para especificar el formato de la respuesta.
7. Describir como el servicio va a ser invocado, mediante un documento WSDL/WADL o simplemente HTML (Claudia Bravo Batista, Nemury Silega Martínez y Janiel Treto Portal 2017).

1.2 Aplicaciones de planificación existentes

En el mundo existen aplicaciones móviles para la gestión e actividades con el objetivo de organizar mejor el trabajo, mejorar la productividad y optimizar el tiempo. A continuación, se exponen algunas de las aplicaciones que se analizan para obtener elementos que ayuden a guiar el diseño de la interfaz visual, así como la distribución de la información y las funcionalidades principales de una aplicación de este tipo.

1.2.1 Evernote

Es una aplicación móvil para la gestión de una agenda de trabajo. Se presenta como un espacio para "capturar todas las experiencias" y acceder a ellas desde cualquier lugar, ya que todas las anotaciones se actualizarán al instante en todos los dispositivos en los que tengas instalados la aplicación. Se puede anotar todas las ideas y tareas pendientes a través de texto, imágenes, audios, capturas de pantalla, etc. Lo más interesante como herramienta es:

- Está disponible para dispositivos iOS y Android.
- Puede combinar varios formatos de contenidos (texto, foto y audio)
- Puede sincronizar todo con varios dispositivos (con los que se puede trabajar offline)
- Tiene una interfaz muy agradable y fácil de usar.
- Dispone de aplicación de escritorio, por lo que permite vincular todos tus dispositivos.
- Es una herramienta muy versátil.

1.2.2 Todoist: lista de tareas

Esta agenda online, permite anotar cualquier tarea o idea, accediendo a las mismas estando offline. Todolist tiene varias características, como son:

- Puede sincronizarce con todos los dispositivos en los que tengas instalada la aplicación.
- Guarda lista de tareas en la nube y puede gestionar las tareas aun estando offline.
- Puede organizar usando fechas de plazo, plazos recurrentes, subtareas, prioridades, subproyectos y codificación por colores.
- Permite compartir proyectos, asignar tareas y colaborar en línea.
- Es muy buena para planificar los días con un cronograma visual.
- Está integrada con Dropbox y G-Drive para añadir documentos a las tareas.
- Agrega tareas desde cualquier aplicación gracias a una extensión de Todoist.

1.2.3 Toodledo

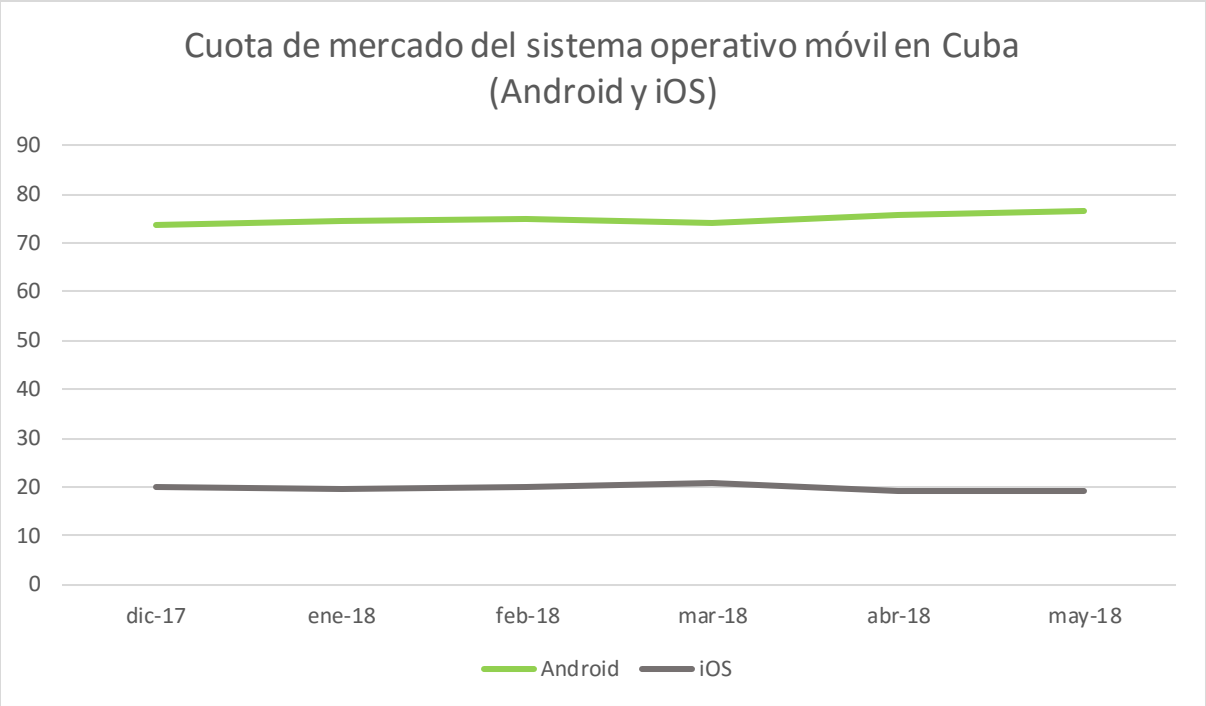
Esta aplicación es para gestionar una agenda electrónica de trabajo**,** las "cosas por hacer", notas, listas, calendarios, eventos, etc. Lo más interesante de la ella es:

- Es gratuita.
- Disponible para iOS y Android.
- Facilita la fijación de metas y objetivos, que pueden verse fácilmente en el calendario anual.
- Es muy flexible y multifuncional, puedes adaptarla a tus necesidades.
- Es muy buena para recordar eventos y otras citas.

1.3 Sistemas operativos móviles

La evolución de los dispositivos móviles ha sido universal y veloz, debido al rápido avance de las tecnologías. Estos dispositivos se incorporan a la vida de las personas como una

herramienta indispensable en la vida cotidiana. Los dispositivos de este tipo han tomado ese auge debido a la portabilidad de los mismos, que te permiten utilizar sus funcionalidades en diferentes espacios. Los dispositivos móviles cuentan con un Sistema Operativo (SO) y existen varios, dentro los que más se usan, Android y iOS. A continuación, se muestra en la gráfica siguiente un estudio de la cuota de mercado del SO móvil en Cuba consultado en http://gs.statcounter.com/os-market-share/mobile/cuba.



Se selecciona Android como sistema operativo para el desarrollo de la solución propuesta considerando las características de ser el más utilizado, según el estudio realizado de la cuota de mercado del SO móvil en Cuba. Además, es una plataforma abierta, porque no está atada a ningún fabricante de hardware; es de código abierto y las herramientas y tecnologías que se utilizan para el desarrollo son libres.

1.4 Tecnologías y herramientas

1.4.1 Sistema Operativo Android

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

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA DE LA APLICACIÓN

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

Arquitectura de Android

Android presenta una arquitectura basada en 4 niveles como se muestra en la Figura 1 Arquitectura de Android:

- Un **Kernel Linux** versión 2.6 que sirve como base de la pila de software y se encarga de las funciones más básicas del sistema: gestión de drivers, seguridad, comunicaciones, etc.
- Una **capa de bibliotecas de bajo nivel en C y C++,** como SQLite para persistencia de datos; OpenGL ES para gestión de gráficos 3D, con aceleración 3D opcional y Webkit como navegador web embebido y motor de renderizado HTML.
- Un **framework para el desarrollo de aplicaciones**, dividido en subsistemas para gestión del sistema como el “Administrador de paquetes”, el “Administrador de telefonía” (para la gestión del hardware del teléfono anfitrión) o el acceso a APIs sofisticadas de geolocalización o mensajería XMPP. Los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar el recurso de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Éste mecanismo permite que los componentes sean reemplazados por el usuario. También incluye un sistema de vistas para manejar la interfaz de usuario de las aplicaciones, que incluyen la posibilidad de visualización de mapas o renderizado HTML directamente en la interfaz gráfica de la aplicación.
- **Aplicaciones:** Las aplicaciones base incluyen un teléfono, cliente de email, programa de envió SMS, calendario, mapas, navegador, contactos, que pueden a su vez ser usados por otras aplicaciones.(Paco Blanco et al. 2013)

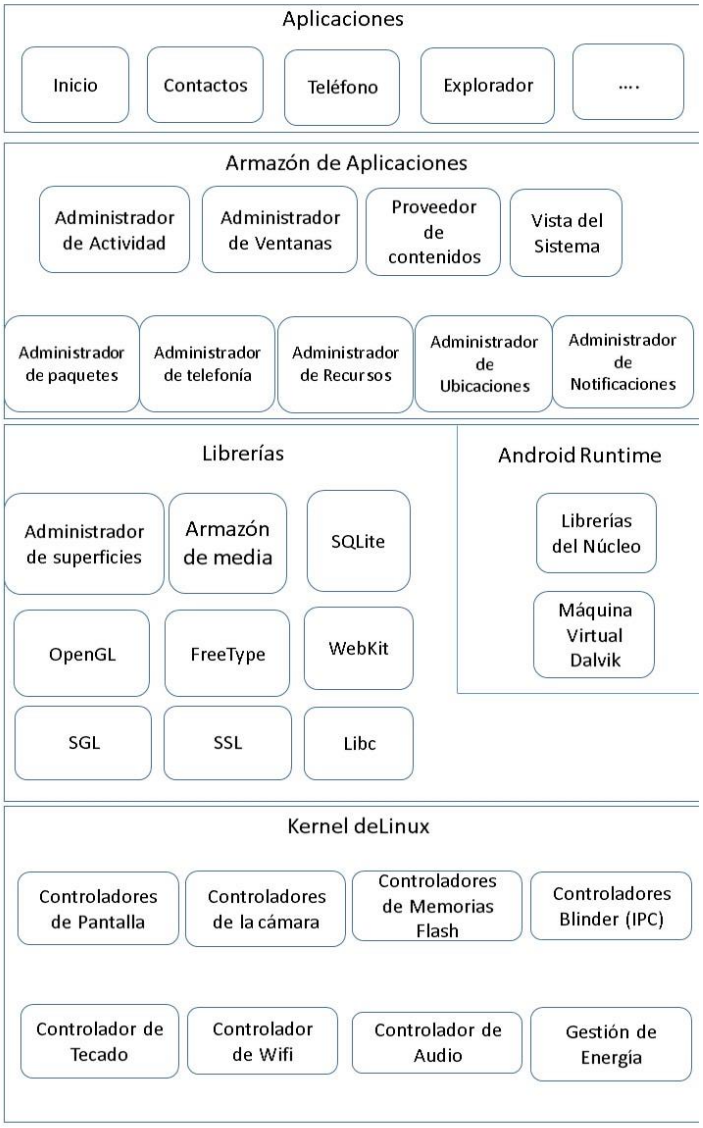


Figura 1. Arquitectura de Android

1.4.2 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android y se basa en IntelliJ IDEA. Además, el potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan tu productividad durante la compilación de apps para Android como:

- Un sistema de compilación basado en Gradle flexible.
- Un emulador rápido con varias funciones.
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android.
- Instant Run para aplicar cambios mientras tu app se ejecuta sin la necesidad de compilar un nuevo APK.

- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las apps e importar ejemplos de código.
- Gran cantidad de herramientas y frameworks de prueba (Android Developers, 2018).

1.4.3 Kotlin

Kotlin es un lenguaje basado en JVM (Java Virtual Machine o Máquina Virtual de Java). Esta nueva tecnología cuenta con, la capacidad de LLVM-backed para compilar nativos ejecutables. Seguridad de referencias a nulos, clases de datos y la facilidad de crear DSLs son algunos de los beneficios que se han aprovechado junto con la librería Anko para el desarrollo de Android. A pesar de las desventajas de una compilación inicial lenta y la dependencia en IntelliJ para soporte de IDE de primera clase (Kotlin.es, 2018) (Kotlinlang, 2018).

1.4.4 Gradle

Gradle es una herramienta de automatización de compilación de código abierto centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben usando Groovy o Kotlin DSL. Lea sobre las características de Gradle para saber qué es posible con Gradle.

- **Altamente personalizable:** Gradle se modela de una manera que se puede personalizar y ampliar de las formas más fundamentales.
- **Rápido:** Completa las tareas rápidamente reutilizando las salidas de las ejecuciones anteriores, procesando solo las entradas que cambiaron y ejecutando las tareas en paralelo.
- **Potente:** Es la herramienta de compilación oficial para Android y viene con soporte para muchos lenguajes y tecnologías populares (Gradle Inc., 2018).

1.4.5 XML

XML (Extensible Markup Language o Lenguaje de Marcado Extensible), se utiliza para representar información estructurada en la web, de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa, por muy diversos tipos de aplicaciones y dispositivos. Se clasifica como un lenguaje extensible porque permite a sus usuarios definir sus propios elementos. Su objetivo principal es ayudar a los sistemas de información a compartir datos estructurados, particularmente a través de Internet, y se usa tanto para codificar documentos como para serializar datos (XML | Object Management Group., 2018).

1.4.6 SQLite

SQLite es una biblioteca en proceso que implementa un motor de base de datos SQL transaccional independiente, sin servidor y de configuración cero. El código para SQLite es de dominio público y, por lo tanto, es gratuito para cualquier uso, comercial o privado. SQLite es la base de datos más implementada del mundo con más aplicaciones de las que podemos contar, incluidos varios proyectos de alto perfil. SQLite es un motor de base de datos SQL incorporado. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor por separado. SQLite lee y escribe directamente en archivos de disco ordinarios. Una base de datos SQL completa con múltiples tablas, índices, disparadores y vistas, está contenida en un solo archivo de disco. El formato de archivo de la base de datos es multiplataforma: puede copiar libremente una base de datos entre sistemas de 32 bits y de 64 bits o entre arquitecturas big-endian y little-endian. Estas características hacen que SQLite sea una opción popular como formato de archivo de aplicación (About SQLite, 2018).

1.4.7 Herramienta CASE Visual Paradigm for UML

Visual Paradigm es una herramienta multiplataforma, característica que la favorece y que viene muy acorde a la migración al software libre que lleva a cabo Cuba. Es una herramienta UML profesional que soporta el ciclo de vida del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Se integra con varios IDEs (Entorno de Desarrollo Integrado) y soporta múltiples usuarios trabajando sobre un mismo proyecto. Ofrece interoperabilidad entre diagramas, ya que permite a partir de un diagrama obtener otro que guarde relación con el mismo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (Visual Paradigm , 2018).Para el desarrollo de la solución se utilizó Visual Paradigm for UML 8.0.

1.5 Metodología de desarrollo

Las metodologías ágiles poseen ciertas propiedades que las hacen totalmente aplicables al dominio de software móviles. Las metodologías que se pueden destacar son Adaptive Software Development, la familia de metodologías Crystal, el Método de Desarrollo de Sistemas Dinámicos (Dynamic System Development Methodology, DSDM), eXtreme Programming(XP), Feature-Driven Development (FDD), Lean Sofware Development, Scrum, Agile Modeling o Pragmatic Programming, Agile Model-Driven Development, Agile Unified Process, Rational Unified Process (Camarero, y otros, 2003).

A continuación, se refieren algunas de las propiedades que presentan estas metodologías que son aplicables al desarrollo móvil.

Tabla 1.Comparativa entre las características básicas o bases (home ground) ágiles y los rasgos

Características ágiles	**Motivación lógica**	**Aplicable a tecnologías móviles**
Alta volatilidad del entorno	Debido a la frecuencia en el cambio que sufren los requisitos, se tienen menos necesidad de diseño y planificación inicial y mayor necesidad de desarrollos incrementables e iterativos.	Alta incertidumbre, entornos dinámicos, cientos de nuevos terminales cada año.
Equipos de desarrollos pequeños.	Capacidad de reacción más rápida, trabajo basado en la compartición de la información menos documentos.	La mayor parte de los proyectos de desarrollo móvil son de equipos pequeños.
Cliente identificable	Conocimiento de los intereses de los clientes	Potencialmente, hay un número ilimitado de usuarios finales, pero los clientes son fáciles de identificar.
Entornos de desarrollo orientados a objetos	Mayoría de las herramientas de desarrollo ágil existen bajo plataformas orientadas a objetos.	Ejemplo, Java y C++ se usan, en algunos problemas en herramientas como refactorizaciones o primeros test.
Sistemas pequeños	Menos necesidad de diseño inicial.	Las aplicaciones, aunque variables en tamaño, no suelen superar las 10.000 líneas de código.
Ciclos de desarrollo cortos	Propósito de realimentación rápida.	Periodos de desarrollo de 1 a 6 meses.

Se ha seleccionado utilizar metodologías ágiles en el desarrollo de la aplicación propuesta para realizar las operaciones sobre los elementos de la planificación, por las características que presentan este tipo de desarrollo de software antes expuestas. Se presentan tres propuestas en el desarrollo de la tesis, analizando los elementos como:

1. Ciclo de desarrollo
2. Fases e iteraciones.

3. Productos de trabajo generados por cada una de las fases.
4. Cantidad de iteraciones de cada una de las fases.

Se ha seleccionado Variación AUP para la UCI, para guiar el proceso de desarrollo de software de la propuesta solución, por la solicitud realizada por el Departamento de Desarrollo de Componentes, en correspondencia con los lineamientos de desarrollo de software que posee la universidad.

1.5.1 Modelo híbrido para el desarrollo ágil

La metodología propuesta por Rahinmian, V., Ramsin, R., en el documento Designing and agile methodology for mobile software development: a hybrid method engineering approach, se apoya en una combinación del desarrollo adaptativo de software (Adaptive Software Development, ASD) y el diseño de nuevos productos (New Product Development, NPD). Esto supone una decisión crítica para decantarse más del lado del desarrollo de productos que del lado de la gestión de proyectos, lo cual quiere decir que una de las características más sensibles, desde el punto de vista metodológico, para la consolidación de una metodología propia de un entorno móvil, es la presión de los plazos para llegar al mercado, un mercado volátil y altamente dinámico.

En la primera iteración como se muestra en la Figura 2, se divide la fase de análisis con la intención de mitigar riesgos de desarrollo, de la misma forma, el diseño también se segmenta para introducir algo de diseño basado en arquitectura. La implementación y las pruebas sin embargo fusionan introduciendo conceptos de desarrollo orientado a pruebas (Test-Driven Development, TDD). Aparece además una fase de comercialización, incidiendo en el sesgo hacia el desarrollo de producto que se imponen en el escenario del desarrollo de aplicaciones para plataformas móviles. Desde el punto de vista metodológico, los autores afirman haberse apoyado en metamodelos como SPEN (Software Processes Engineering Metamodel, soportado por el entorno de desarrollo de Eclipse, por ejemplo) y OPF, (Open Processes Framework), así como conceptos genéricos de ciclos de vida orientados a objetos como OPSP (Object Oriented Software Processes)

Figura 2. Primera iteración en el diseño híbrido: instanciación de análisis y diseño

La segunda iteración como se muestra en la Figura 3, realiza una integración de ciertas partes de los modelos NPD, añadiendo la generación de ideas en el inicio del ciclo y una prueba de mercado antes de lanzar la fase de comercialización.

Figura 3. Segunda iteración en el diseño híbrido: integración hacia el desarrollo de producto

La tercera iteración como se muestra en la Figura 4, integra directamente el "motor de desarrollo" de los métodos de desarrollo adaptativo (ASD) muy orientado al aseguramiento de la calidad de los procesos de desarrollo.

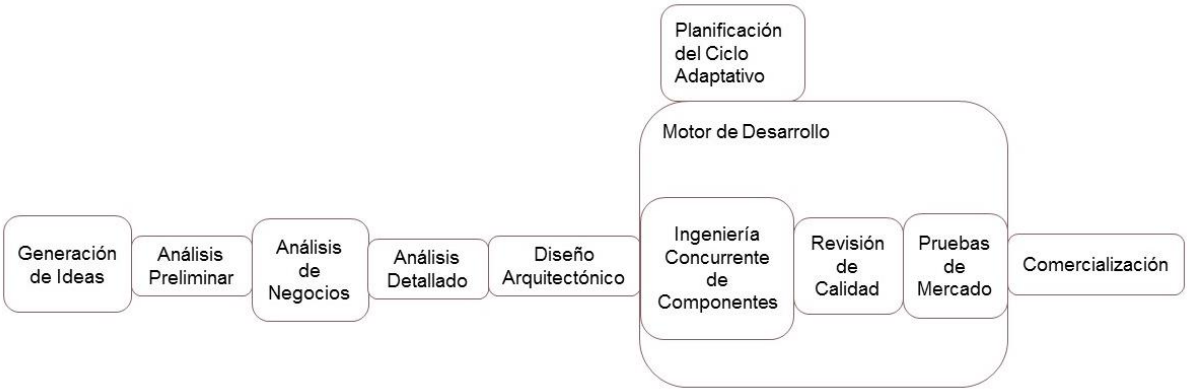


Figura 4.Tercera iteración en el diseño híbrido: motor de desarrollo

Pensando en aquella propiedad "ideal" de disponer de la arquitectura física en una fase temprana del proceso, en la cuarta iteración como se muestra en la Figura 5 se añaden elementos de prototipado; se refina, además, la fase de iniciación del proyecto, sobre la base del mismo elemento de los procesos adaptativos (Camarero, y otros, 2003).

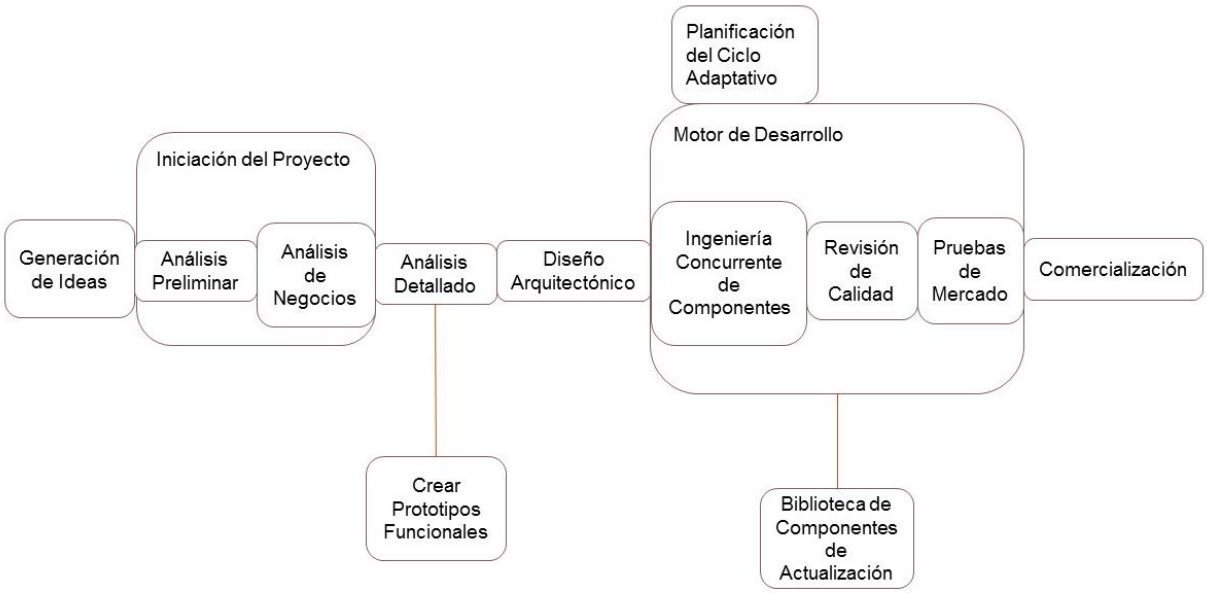


Figura 5.Cuarta iteración en el diseño híbrido: prototipado

1.5.2 Mobile –D

Es un modelo propuesto por Pekka Abrahamsson y su equipo del VTT (Valtion Teknillinen Tutkimuskeskus), en inglés Technical Research Centre of Finland) en Filandia. El ciclo del proyecto se divide en cinco fases: exploración, inicialización, producción, estabilización y prueba del sistema, como se muestra en la Figura 6. En general, todas las fases con la excepción de la primera fase exploratoria contienen tres días de desarrollo distintos: planificación, trabajo y liberación.

Figura 6. Ciclo de Desarrollo Mobile-D

La fase de exploración, siendo ligeramente diferente del resto del proceso de producción, se dedica al establecimiento de un plan de proyecto y los conceptos básicos.

Durante la fase de inicialización, los desarrolladores preparan e identifican todos los recursos necesarios. Se preparan los planes para las siguientes fases y se establece el entorno técnico (incluyendo el entrenamiento al desarrollo ágil se centra fundamentalmente en esta fase, en la investigación de la línea arquitectónica). Esta acción se lleva a cabo durante el día de planificación. Los desarrolladores analizan el conocimiento y los patrones arquitectónicos utilizados en la empresa y los relacionan con el proyecto actual. Se agregan observaciones, se identifican similitudes y se extraen soluciones viables para su aplicación en el proyecto.

En la fase de producción se repite la programación de tres días (planificación-trabajo-liberación) se repite iterativamente hasta implementar todas las funcionalidades. Primero se planifica la iteración de trabajo en términos de requisitos y tareas a realizar. Se preparan las pruebas de iteración. Las tareas se llevan a cabo durante el día de trabajo, desarrollando e integrando el código con los repositorios existentes. Durante el último día se realiza la integración del sistema.

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

La última fase (prueba y reparación del sistema) tiene como objetivo la disponibilidad de una versión estable y plenamente funcional del sistema. El producto terminado e integrado se prueba con los requisitos de cliente y se eliminan todos los defectos encontrados (Camarero, y otros, 2003).

1.5.3 Variación de AUP para la UCI

La metodología de desarrollo a emplearse en los proyectos productivos de la Universidad de las Ciencias Informáticas (UCI) es Variación AUP-UCI. Dicha definición se basa en una variación de la metodología “Proceso Unificado Ágil (AUP por sus siglas en inglés) en unión con el modelo CMMI-DEV v1.3 que significa Capacibility Maturity Model Integration”

La metodología Variación AUP-UCI consta de tres fases: Inicio donde se realizan las actividades relacionadas con la planeación del proyecto, permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto, la fase de Ejecución donde se ejecutan las

actividades para desarrollar el software, se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y libera el producto. Por último, la fase de Cierre donde se analizan los resultados del proyecto y se realizan las actividades formales de cierre del proyecto.

Además, el ciclo de vida de los proyectos consta de siete disciplinas, Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas, Pruebas de liberación y Pruebas de aceptación. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2, serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto) (Sánchez, 2015).

1.5.4 Escenario para la disciplina de requisito

Luego de definir como metodología de desarrollo la Variación AUP-UCI. A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (CUN, DPN o MC) y existen tres formas de encapsular los requisitos (CUS, HU, DRP), surgen cuatro escenarios para modelar el sistema en los proyectos. Se propone para la solución propuesta utilizar el escenario número tres aplicados a los proyectos que modelan el negocio con los productos de trabajo: Descripción de procesos del negocio, Modelo conceptual y Descripción de requisitos por proceso (Sánchez, 2015).

1.6 Conclusiones parciales

En este capítulo se exponen los principales conceptos de la investigación, y tipos de plan que permitió un mejor entendimiento del negocio. Se abordó sobre, los servicios REST esclareciendo como se debe diseñar un servicio de este tipo, las tecnologías, lenguajes de programación y herramientas. Además de la metodología de desarrollo, elementos que se definieron para la solución de la investigación. Por lo que de esta forma se logró construir el marco teórico conceptual de la investigación, sobre el Desarrollo de Aplicaciones en sistemas operativos para dispositivos móviles.

CAPÍTULO 2. DESCRIPCIÓN DE LA APLICACIÓN

En el presente capítulo se realiza una descripción de la solución propuesta, a partir de las disciplinas requisitos, análisis y diseño e implementación definidas por la metodología seleccionada.

El modelado de negocio es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Para modelar el negocio de la solución se propone la variante de Modelo conceptual.

2.1 Modelo Conceptual

Un modelo conceptual es una representación de conceptos en un dominio del problema. En UML se ilustra como un grupo de diagramas de estructura estática donde no se define ninguna operación. Ofrece como ventaja subrayar fuertemente una concentración en los conceptos del dominio, no las entidades del software (Craig Larman 1999).

El modelo conceptual de la solución propuesta está representado por el concepto de Plan, los planes tienen asociadas un conjunto de Actividades, Objetivos, Indicadores, Áreas de Resultados Claves, Observaciones y Anexos.

El concepto de **plan** está representado por un modelo sistemático que se elabora antes de realizar una acción, con el propósito de dirigirla y encausarla.

El concepto de **objetivo** está representado por los resultados, situaciones o estados que una entidad pretende alcanzar o a los que pretende llegar, en un período de tiempo y a través del uso de recursos con los que dispone o planea disponer.

El concepto de **observación** está representado para denominar una nota en un escrito para aclarar o precisar un punto dudoso. Puede ser también un comentario o una indicación.

El concepto de **actividad** está representado por un conjunto de operaciones o tareas, propias de una persona o entidad, destinadas para cumplir determinado objetivo.

El concepto de **anexo** está representado por un fichero que se anexa a un elemento de la planificación, el mismo puede contener cualquier objeto digitalizado de los siguientes documentos (PDF, EXCEL, DOC) o imágenes.

El concepto de **Área de Resultados Clave (ARC)** está representado para definir un resultado cuyo logro implica la participación de dos o más elementos (departamento, dirección, grupo de trabajo, etc.) de la estructura funcional de una entidad.

La figura 7 muestra los principales conceptos del dominio y las relaciones entre ellos.

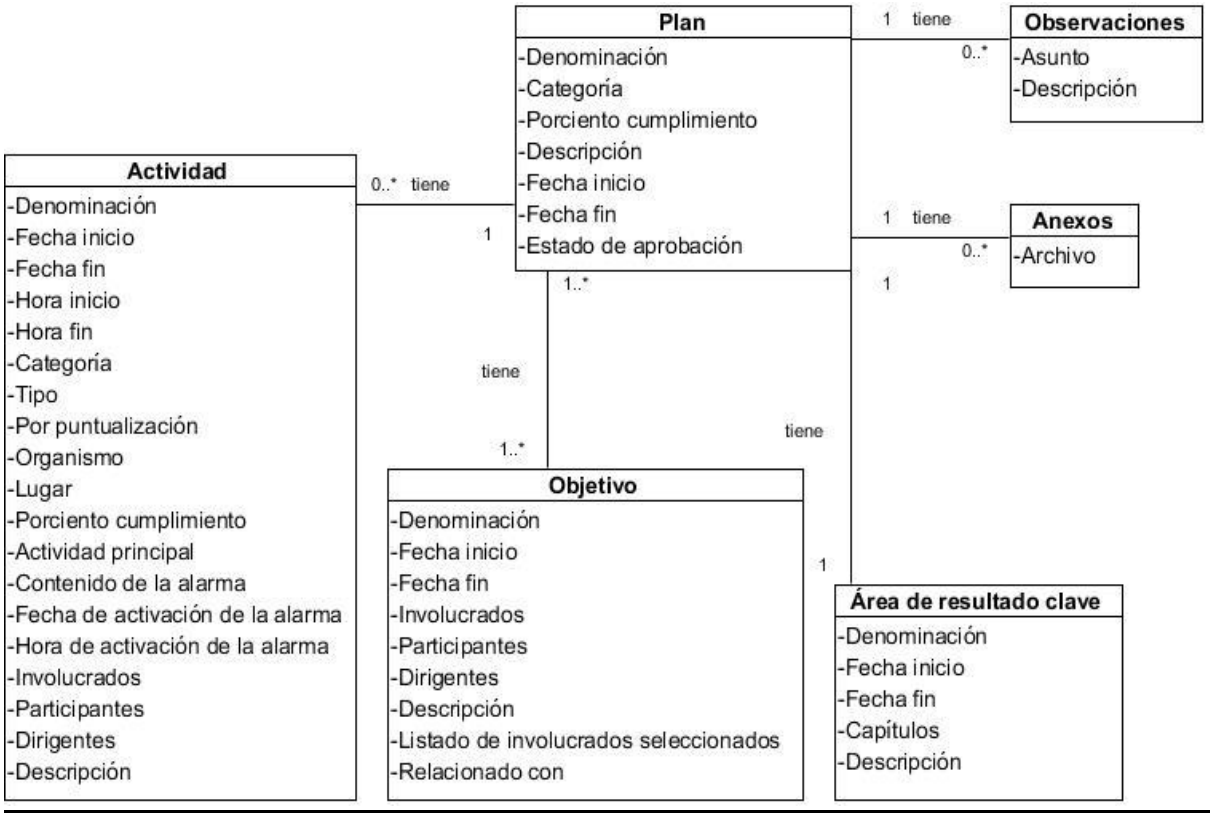


Figura 7. Modelo Conceptual de la aplicación propuesta

2.2 Disciplina requisitos

El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir y comprende la administración de los requisitos funcionales y no funcionales del producto (Sánchez, 2015).

2.2.1 Técnicas para la identificación de requisitos

Para definir los requisitos que forman parte de la solución propuesta se utilizó la siguiente técnica:

- Entrevista: Requiere de una buena selección a los entrevistados para obtener información de calidad en el menor tiempo posible. Esta técnica fue aplicada a los desarrolladores del centro CEIGE a cargo del desarrollo y mantenimiento de SIPAC.
- Análisis de la documentación: Requiere de varios tipos de documentación, como manuales y reportes, pueden proporcionar al analista información valiosa con respecto al SIPAC y a sus operaciones. Se analizaron las descripciones de requisitos por proceso, documentadas para la versión 3.0 de la aplicación web del SIPAC.

Como resultado de aplicar las técnicas de capturas de requisitos, se obtuvo una lista de los requisitos funcionales (RF) y los requisitos no funcionales (RNF) que se listan en el siguiente subepígrafe.

2.2.2 Especificación de requisitos

Los requisitos constituyen un punto clave en el desarrollo de aplicaciones informáticas. Un gran número de proyectos de software fracasan debido a una mala definición, especificación o administración de estos. Los requisitos se enfocan en las especificaciones de lo que se desea desarrollar y tienen dos clasificaciones: requisitos funcionales y no funcionales. Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particularidades y de cómo se debe comportar en distintas situaciones y los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema (Sommerville, 2005). A continuación, se presentan los requisitos definidos para el desarrollo de la aplicación propuesta.

Requisitos funcionales

Tabla 2. Requisitos funcionales del sistema

No	Nombre	Descripción	Prioridad	Complejidad
RF1	Adicionar Plan	Adicionar un nuevo plan con la información requerida.	Alta	Alta
RF2	Modificar Plan	Modifica un plan especificado con la información necesaria.	Alta	Alta
RF3	Eliminar Plan	Elimina un plan especificado.	Alta	Baja
RF4	Listar Plan	Muestra todos los planes existentes.	Alta	Alta
RF6	Buscar Plan	Muestra los planes que concuerden con el criterio de búsqueda.	Media	Baja
RF7	Adicionar observación del plan	Adicionar una nueva observación con la información requerida.	Media	Alta
RF8	Modificar observación del plan	Modifica una observación especificada con la información necesaria.	Media	Alta
RF9	Eliminar observación del plan	Elimina una observación especificada.	Media	Baja

RF10	Listar observación del plan	Muestra todas las observaciones existentes.	Media	Media
RF11	Adicionar anexos	Adicionar un nuevo anexo con la información requerida.	Media	Alta
RF12	Modificar anexos	Modifica un anexo especificado con la información necesaria.	Media	Alta
RF13	Eliminar anexos	Elimina un anexo especificado.	Media	Baja
RF14	Listar anexos	Muestra los anexos que concuerden con el criterio de búsqueda.	Media	Media

2.2.3 Descripción de requisitos por proceso

Tabla 3 Descripción textual del requisito Adicionar Plan

Precondiciones	El usuario inicia la aplicación, se autentica en el sistema y cuenta con los permisos necesarios para ejecutar esta acción. El usuario selecciona en el menú lateral la opción Plan. El usuario selecciona la opción Adicionar Plan.
Flujo de eventos	
Flujo básico Adicionar plan	
1.	Se muestra la ventana Adicionar plan con los siguientes campos para que el usuario introduzca los datos (Ver validación 1): **Denominación** (Nombre que recibe el plan al ser adicionado que, puede ser una combinación de letras, dígitos y caracteres especiales.) **Tipo del plan** (Nomenclador que lista la categoría de los planes. El usuario solo puede seleccionar uno de estas opciones: Plan individual de trabajo, Plan mensual, Plan anual) **Porciento cumplimiento** El usuario especifica el número de porciento de cumplimiento del plan) **Fecha inicio** (El usuario selecciona la fecha en que se inicia el plan) **Fecha fin** (El usuario selecciona la fecha estimada de culminación del plan) **Descripción** (El usuario especifica las características que tendrá el plan)
2.	Se da click en el botón **Aceptar**.
3.	Se validan los datos introducidos (Ver validación 1).
4.	Si los datos introducidos son correctos se registran los datos del plan.
5.	Se muestra un mensaje de información indicando que la acción se ha realizado

	satisfactoriamente.
6.	Concluye el requisito.
Pos-condiciones	
1.	Se adicionó en el sistema el plan con el estado de creado.
2.	Se actualiza el listado de planes. Ver RF Listar planes
Flujos alternativos	
Flujo alternativo 4.a Campos vacíos	
1.	El sistema muestra un mensaje de error “Por favor verifique los datos, existen campos obligatorios vacíos o campos con valores incorrectos”
2.	Volver al paso 1 del flujo básico.
Pos-condiciones	
1.	N/A
Flujo alternativo *.a El usuario cancela la acción	
1.	Concluye el requisito.
Pos-condiciones	
1.	No se registran los datos.
Validaciones	
1	Se validan los datos según lo establecido en el Modelo conceptual Modelo Conceptual de la aplicación propuesta.
1	El sistema valida que la fecha de fin sea igual o mayor que la fecha de inicio.

Conceptos	**Identificador**	**Utilizados internamente**: Identificador
	Denominación	**Visibles en la interfaz**: Denominación
	Categoría del plan	**Visibles en la interfaz**: Categoría del plan
	Fecha inicio	**Visibles en la interfaz**: Fecha inicio
	Fecha fin	**Visibles en la interfaz**: Fecha fin
	Porciento de cumplimiento	**Visibles en la interfaz**: Porciento de cumplimiento
	Descripción	**Visibles en la interfaz**: Descripción

Requisitos especiales	N/A.
Asuntos pendientes	N/A.

Tabla 4 Descripción textual del requisito Modificar Plan

Precondiciones	El usuario se ha autenticado en el sistema y cuenta con los permisos necesarios para ejecutar esta acción. El usuario selecciona en el menú lateral la opción Plan. Existe al menos un plan registrado en el sistema.
Flujo de eventos	
Flujo básico Modificar plan	
1.	El usuario presiona por unos segundos uno de los planes que se listan (Ver requisito **Listar Plan** de esta agrupación de requisitos).
2.	El sistema muestra un menú y permite editar o eliminar los datos del plan.
3.	El usuario selecciona la opción Modificar.
4.	Se muestran en la ventana Modificar plan los siguientes campos para que el usuario modifique sus datos (Ver validación 1): **Denominación** (Nombre que recibe el plan al ser adicionado que, puede ser una combinación de letras, dígitos y caracteres especiales.) **Tipo del plan** (Nomenclador que lista la categoría de los planes. El usuario solo puede seleccionar uno de estas opciones: Plan individual de trabajo, Plan mensual, Plan anual) **Porciento cumplimiento** El usuario especifica el número de porciento de cumplimiento del plan) **Fecha inicio** (El usuario selecciona la fecha en que se inicia el plan) **Fecha fin** (El usuario selecciona la fecha estimada de culminación del plan) **Descripción** (El usuario especifica las características que tendrá el plan)
5.	Se presiona el botón **Aceptar**.
6.	Se validan los datos introducidos (Ver validación 1).
7.	Si los datos introducidos son correctos se registran los datos del plan.
8.	Se muestra un mensaje de información indicando que la acción se ha realizado satisfactoriamente.
9.	Concluye el requisito.
Pos-condiciones	
1.	Se modificó en el sistema el plan con el estado de creado.

Flujos alternativos		
Flujo alternativo 4.a Campos vacíos		
1.	El sistema muestra un mensaje de error “Por favor verifique los datos, existen campos obligatorios vacíos o campos con valores incorrectos”	
2.	Volver al paso 3 del flujo básico.	
Pos-condiciones		
1.	N/A	
Flujo alternativo *.a El usuario cancela la acción		
1.	Concluye el requisito.	
Pos-condiciones		
1.	No se registran los datos.	
Validaciones		
1.	Se validan los datos según lo establecido en el Modelo Conceptual de la aplicación propuesta.	
2	El sistema valida que la fecha de fin sea igual o mayor que la fecha de inicio.	
Conceptos	**Identificador**	**Utilizados internamente**: Identificador
	Denominación	**Visibles en la interfaz**: Denominación
	Categoría del plan	**Visibles en la interfaz**: Categoría del plan
	Fecha inicio	**Visibles en la interfaz**: Fecha inicio
	Fecha fin	**Visibles en la interfaz**: Fecha fin
	Porciento de cumplimiento	**Visibles en la interfaz**: Porciento de cumplimiento
	Descripción	**Visibles en la interfaz**: Descripción
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Requisitos no funcionales

Los requisitos no funcionales, son aquellos requisitos que no refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema.
Los requisitos no funcionales pueden venir de las características requeridas del software (requisitos del producto), de la organización que se desarrolla el software (requisitos organizacionales) o de fuentes externas (Sommerville, 2005).

Requisitos organizaciones: se derivan de políticas y procedimientos existentes en la organización del cliente y en la del desarrollador (Sommerville, 2005).

Requisitos de implementación

RN1. La aplicación requiere un sistema operativo Android con versión 4.4.2 o superior.

RN2. La aplicación requiere una PC servidora que tenga instalado el lenguaje de Programación Python 3, el marco de trabajo Django 1.11, el gestor de bases de datos Postgres 9.4 y el Sistema de Planificación de Actividades en su versión 3.0.

RN3. La aplicación requiere la publicación de los servicios REST del módulo de Planificación, en el Sistema de Planificación de Actividades SIPAC.

RN4. La aplicación requiere que el protocolo de comunicación que se utilice es HTTPS.

RN5. La aplicación se implementará utilizando Android Studio como Entorno de Desarrollo Integrado (IDE), sistema de gestión de bases de datos: SQLite y lenguaje de programación: Kotlin y XML.

Requisitos del producto: especifican el comportamiento del producto (Sommerville, 2005).

Requisitos de usabilidad

RN 6. El diseño visual de la aplicación deberá ser adaptable o adaptativo, de manera que la apariencia de los desarrollos, se adapten a los dispositivos que se estén utilizando para visualizarlos. Se debe tener en cuenta, el reducido tamaño de las pantallas de los dispositivos móviles y su variedad (generalmente entre 4 y 10 pulgadas de diagonal)

RN7. La aplicación requiere un menú lateral izquierdo que permita acceder al módulo de Planificación.

Requisitos externos: se derivan de los factores externos del sistema y de su proceso de desarrollo (Sommerville, 2005).

Requisitos de seguridad

RN8: La aplicación garantizará la autenticidad cuando el usuario se autentique mediante el usuario y la contraseña.

2.3 Disciplina análisis y diseño

En esta disciplina los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo la arquitectura). Además, en esta disciplina se modela el sistema y su forma para que soporte todos los requisitos, incluyendo los requisitos no funcionales. Los modelos desarrollados son más formales y específicos que el de análisis (Sánchez, 2015).

2.3.1 Descripción de los conceptos

Plan: Representa los planes que pueden ser gestionados.

Actividad: Representa conjunto de operaciones o tareas que son ejecutadas por una persona o unidad administrativa como parte de una función asignada.

Objetivo: Representa conjunto de propósitos o metas de un plan específico.

ARC: Representa el área de resultado clave.

Observaciones: Representa las observaciones de un plan específico.

Anexos: Representa los archivos o documentos de un plan.

2.4 Diagrama de clase del diseño

Un diagrama de clases del diseño (DCD) describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Muestra las relaciones existentes entre las clases del sistema. Son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema (Pressman, 2010). A continuación, la Figura 8 representa el diagrama de clases del diseño de la aplicación propuesta.

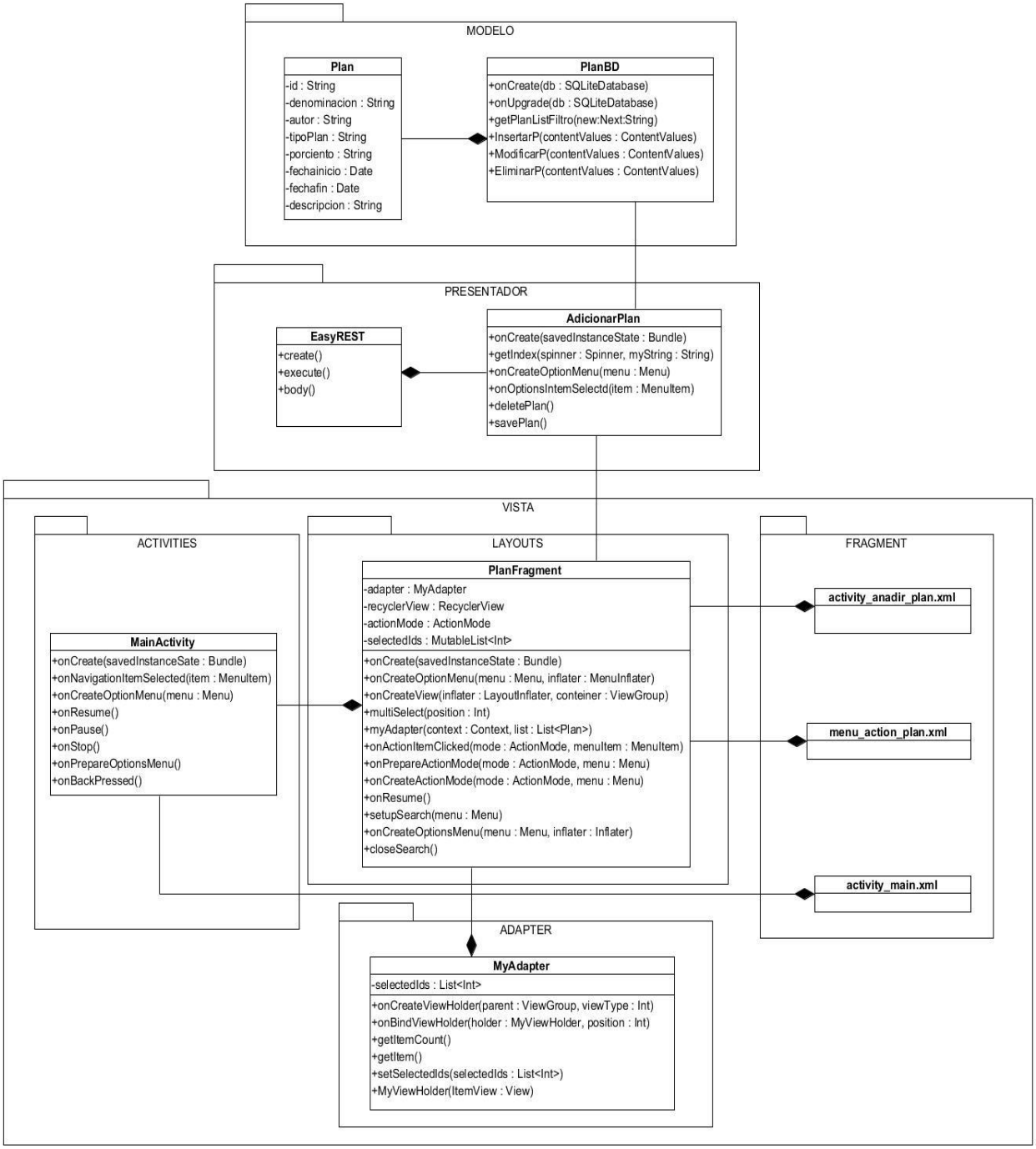


Figura 8. Diagrama de clases del diseño

2.5.3 Patrones Generales de Software para Asignar Responsabilidades (GRASP)

Un patrón describe un problema que ocurre varias veces, así como el núcleo de la solución al inconveniente, de forma que puede utilizarse en ilimitadas ocasiones sin tener que hacer dos veces lo mismo (Larman, 1999).

En la solución del sistema se aplicaron principalmente los siguientes patrones de asignación de responsabilidades:

Controlador*:* es utilizado por todas las clases implicadas en la capa de presentación de la lógica del negocio. Poseen la responsabilidad de controlar el flujo de eventos mediante las actividades correspondientes. En la arquitectura de la aplicación se definió una clase controladora (Inicio) que es la encargada de controlar la lógica del negocio en lo que le respecta.

Creador: Este patrón ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Este patrón se evidencia en la clase *AdicionarPlan.kt* que crea objetos de la clase *PlanFragment.kt*, para acceder a los elementos de esta clase.

Experto*:* Este patrón indica que la clase que cuenta con la información necesaria para cumplir la responsabilidad es la responsable de manejar la información. En el Sistema de Planificación, específicamente en la clase *AdicionarPlan.kt* se evidencia este patrón ya que es la única que tiene la información necesaria para crear la vista *activity_adicionar_plan.xml*.

Alta Cohesión*:* Este patrón plantea que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. El uso de dicho patrón queda evidenciado en la clase *Plan.kt* la cual posee la estructura necesaria para guardar la información de forma coherente y de acuerdo a su responsabilidad.

Bajo acoplamiento*:* Este patrón es utilizado en la creación de clases independientes para la lógica de los diferentes tipos de clases, actividades y entidades. Esto trae como ventaja que solo se realicen acciones sobre el tipo de entidad que se solicite o formulario correspondiente y no sobre todo el conjunto. Estas clases se encargan de la representación de los elementos reales de cada uno respectivamente y las actividades, de manejar los componentes visuales de cada funcionalidad.

2.5 Patrones de diseño de software

La arquitectura de software es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores del software compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema, puesto que establecen la estructura, funcionamiento e interacción entre las partes del software (Gonzalez, 2014).

2.5.2 Patrón arquitectónico Modelo Vista Presentador

El patrón arquitectónico Modelo Vista Presentador (MVP) representado en la Figura 9, es un patrón de diseño que surge para realizar pruebas automáticas de la interfaz gráfica, para ello

la idea es codificar la interfaz de usuario lo más simple posible. En su lugar, toda la lógica de la interfaz de usuario, se hace en una clase separada (que se conoce como Presentador) que no dependa en absoluto de los componentes de la interfaz gráfica y que, por tanto, es más fácil de realizar pruebas. Idealmente el patrón MVP permitiría conseguir que una misma lógica pudiera tener vistas totalmente diferentes e intercambiables.

Básicamente este patrón consiste en tres componentes:

- La vista. Compuesta de las ventanas y controles que forman la interfaz de usuario de la aplicación.
- El modelo. Que es donde se lleva a cabo toda la lógica de negocio.
- El presentador. Es una capa intermediaria entre la Vista (la interfaz gráfica de usuario) y el modelo de datos. Recupera los datos del modelo y se los devuelve a la vista formateados. Pero a diferencia del MVC típico, también decide qué ocurre cuando se interactúa con la vista (MoleQla: revista de Ciencias de la Universidad Pablo de Olavide, Patrón Modelo-Vista-Presentador (MVP), 2015).

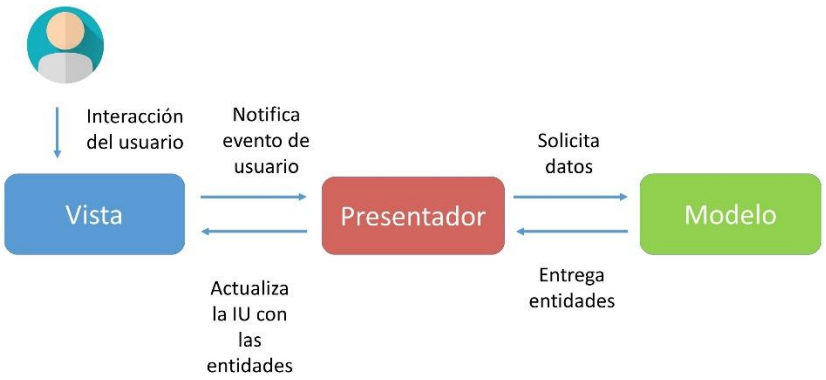


Figura 9. Patrón arquitectónico Modelo-Vista-Presentador

2.5.3 Servicios REST de SIPAC

El marco de trabajo Django REST es un conjunto de herramientas potente y flexible para crear API web, permite crear un API REST sobre Django de forma sencilla ofreciendo una alta gama de métodos y funciones para el manejo, definición y control de los recursos, y es una tecnología utilizada y reconocida por empresas tan importantes como Mozilla y Red Hat (Django REST framework, 2018).

Se propone la utilización del marco de trabajo de Python Django REST framework. El cual, permite la implementación de una API REST e interactúa con la base de datos de SIPAC. Para ello se identifican las tablas con las que se va a interactuar y se registran como recursos en la API, creándose los servicios web que permiten la modificación de los datos. De esta forma, al iniciarse el servidor de SIPAC se inicia automáticamente la aplicación Django REST y se publican los servicios definidos, proporcionando elementos de seguridad para el consumo de los mismos. Para ello deberá implementar un consumidor de servicios, que

utilizando los métodos GET, POST, PUT y DELETE definidos por el protocolo HTTP realice las peticiones, que permitan listar, insertar, modificar y eliminar información en el servidor de SIPAC. Como resultado de estas peticiones recibirá una respuesta de la API REST en formato json, con el resultado de la operación solicitada y ambos sistemas podrán inter operar compartiendo información entre ellos, como se muestra en la Figura 10.

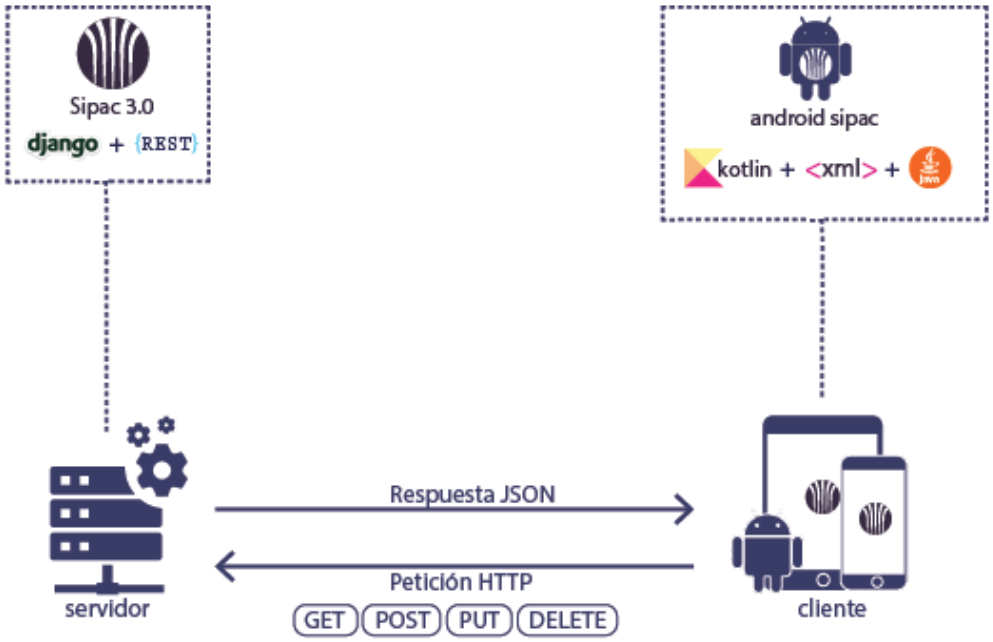


Figura 10. Servicio REST de SIPAC

En el Anexo 1 Django REST framework se muestran las rutas para consumir los servicios del módulo de Planificación. En el anexo dos se muestra un ejemplo del consumo mediante el GET del servicio REST de Plan mostrándose los elementos de un Plan que son: Denominación, Autor, Tipo Plan (Plan de trabajo individual, Plan anual de actividades, Plan mensual), Porciento, Fecha Inicio, Fecha Fin, Create by, Descripción, Observación, Anexos. En el Anexo 3, se muestra un ejemplo del consumo mediante el GET del servicio REST de Objetivo, mostrándose los elementos de un Objetivo que son: Denominación, Estado cumplimiento, Autor, Fecha inicio, Fecha fin, Descripción, Participante, Dirigente, Anexo, Observación.

2.6 Conclusiones parciales

Al finalizar del presente capítulo se arribó a las siguientes conclusiones parciales:

Se elaboraron los artefactos correspondientes a la metodología AUP-UCI en el escenario tres, creando así la documentación necesaria de la investigación que sirvió de guía a los desarrolladores para la implementación de la solución. Se abordaron los principales

conceptos del dominio del problema lo que permitió un mejor entendimiento del módulo de Planificación de SIPAC.

Se definieron los requisitos necesarios para el correcto funcionamiento de la aplicación propuesta del módulo de Planificación de SIPAC. La utilización de los patrones arquitectónicos y de diseño seleccionados proporcionaron un correcto diseño de la aplicación propuesta del módulo de Planificación de SIPAC.

CAPÍTULO 3. IMPLEMENTACIÓN, PRUEBA Y VALIDACIÓN DE LA APLICACIÓN

Introducción

En el presente capítulo se tendrán en cuenta todos los aspectos del diseño del sistema con el fin de llevar a cabo el desarrollo de los flujos de trabajo de implementación y prueba. Se abordan los estándares de codificación empleados durante la implementación del sistema para garantizar un buen entendimiento y legibilidad del código. Se describen las pruebas efectuadas al software que tienen como objetivo: detectar y corregir el máximo de errores en el sistema, antes de su entrega al cliente.

3.1 Modelo de bases de datos de la aplicación Android

El modelo entidad-relación representado en la Figura 11, es un modelo de datos que permite representar cualquier abstracción, percepción y conocimiento en un sistema de información formado por un conjunto de objetos denominados entidades y relaciones, incorporando una representación visual conocida como diagrama entidad-relación (Ochando, 2014).

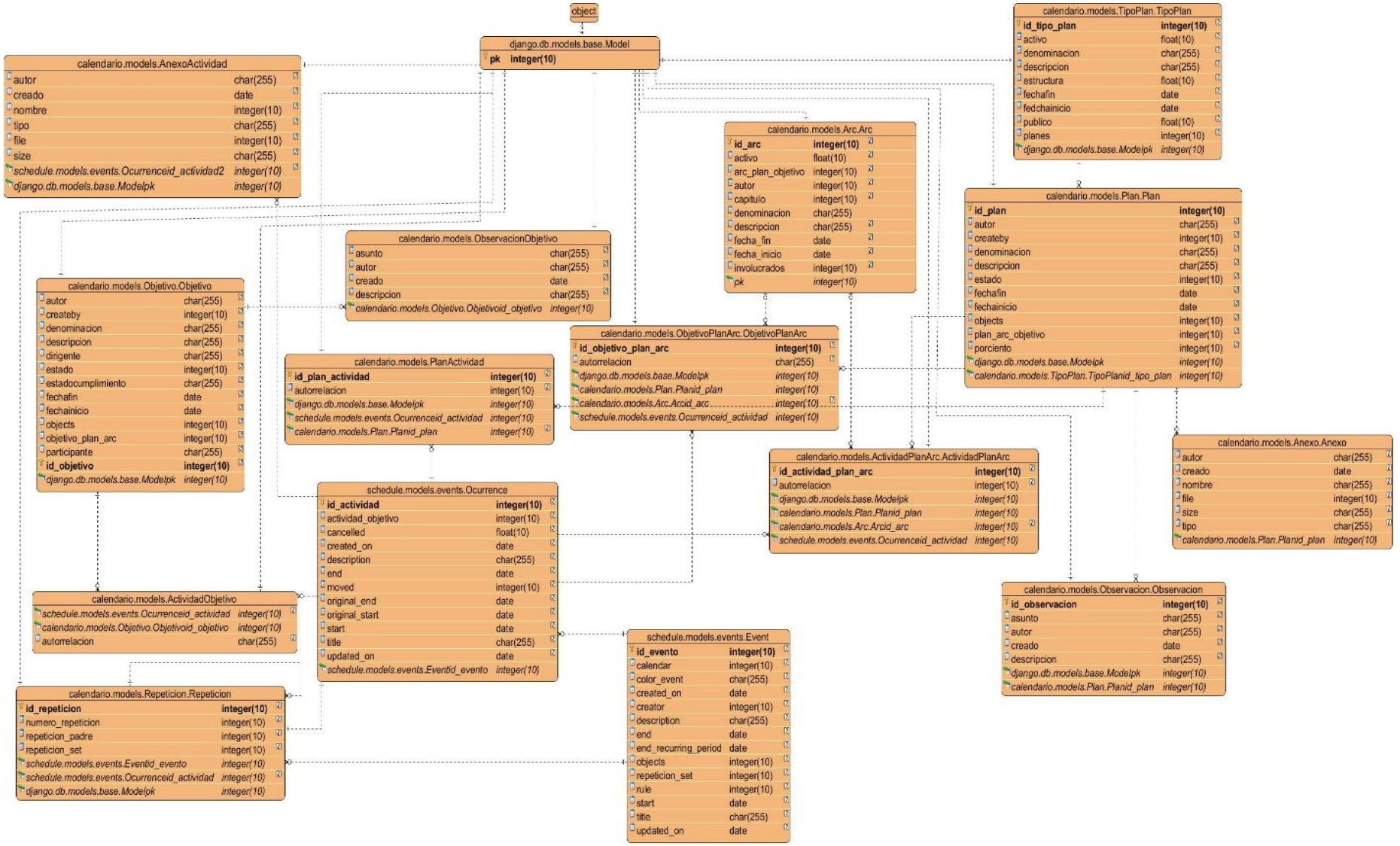


Figura 11. Modelo Entidad-Relación

El diagrama entidad relación de la aplicación Android para brindar y consumir los servicios REST del módulo de Planificación, está caracterizado por la entidad Plan con los atributos: Denominación, Autor, Tipo de Plan, Porciento, Fecha de Inicio, Fecha de Fin, Descripción, Create by. Dicha entidad que va a contener varias actividades relacionadas que se encuentra representada por la entidad Ocurrencia que contiene los datos específicos para cada usuario involucrado en la actividad y tiene los atributos: description, end, created_on, moved, original_en, original_start, start, updated_on. Además de la ocurrencia la actividad está representada por los eventos contenedores de los datos generarles que caracterizan cada actividad, como son: creator, description, color_event, created_on, end, start, titlte, repeticion_set. También esta entidad tiene relación con la entdades: Objetivo, Observación, Área de Resultado Clave (ARC) y Anexo.

3.2 Diagrama de componentes

El diagrama de componentes representado en a Figura 12, muestra la relación entre componentes de software, sus dependencias, su comunicación su ubicación y otras condiciones. Los componentes son agregaciones de alto nivel de las piezas de software más pequeñas y proveen un enfoque de construcción de bloques de “caja negra” para la elaboración de software. Un componente puede ser; tanto un componente de la interfaz de usuario como un servidor de reglas de negocio (Sparx Systems, 2018).

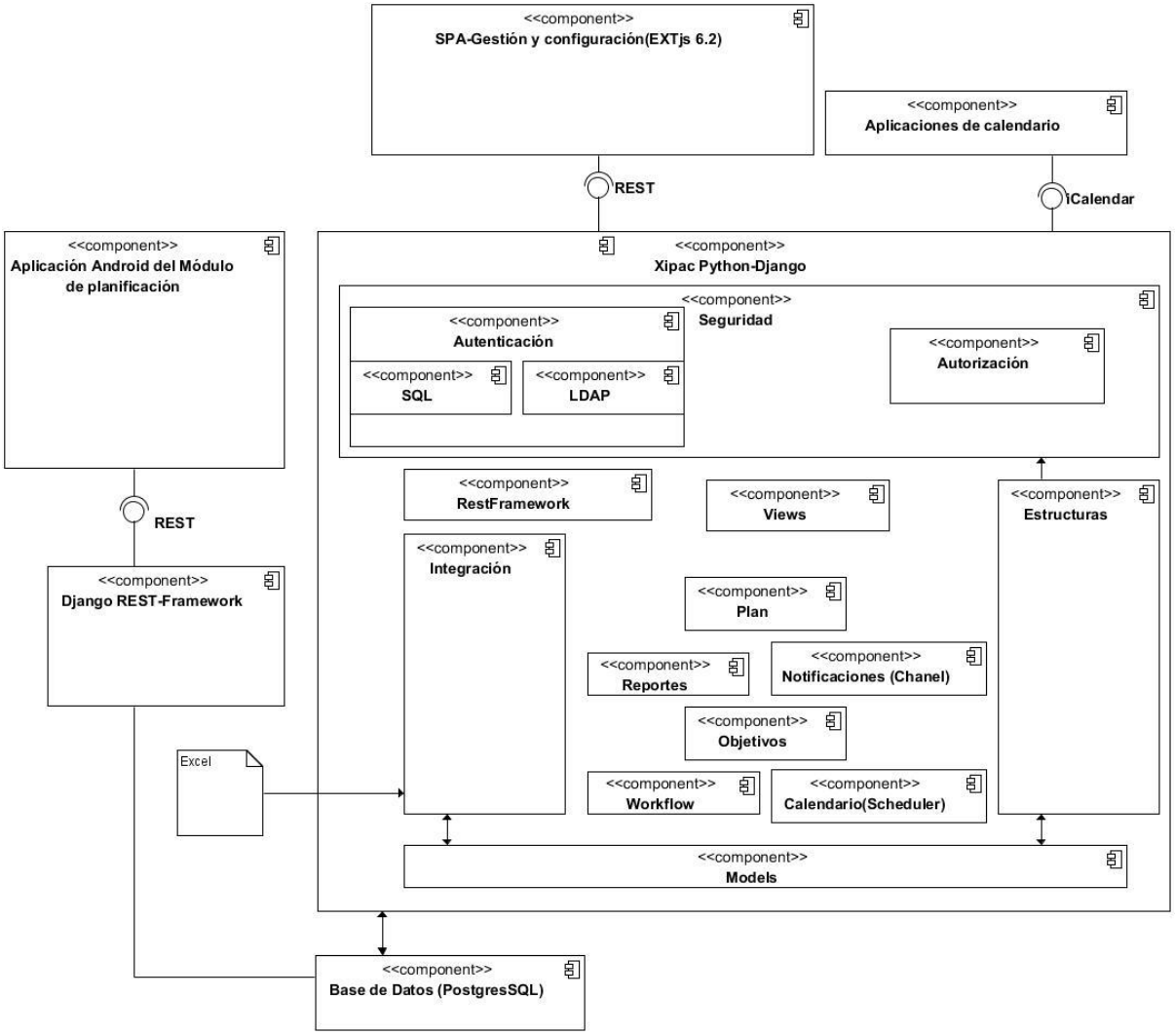


Figura 12. Diagrama de componentes de la aplicación

3.3 Modelo de despliegue de la aplicación

Un Diagrama de Despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos. El objetivo de estos diagramas es mostrar la disposición de las particiones físicas del sistema de información y la asignación de los componentes software a estas particiones. Es decir, las relaciones físicas entre los componentes software y hardware en el sistema a entregar.

El modelo de despliegue suministra la base para la comprensión de la distribución física de un sistema a través de nodos, indicando de qué forma se sitúa el software en el hardware que lo contiene.

A continuación, se presenta el modelo de despliegue de la aplicación a desarrollar:

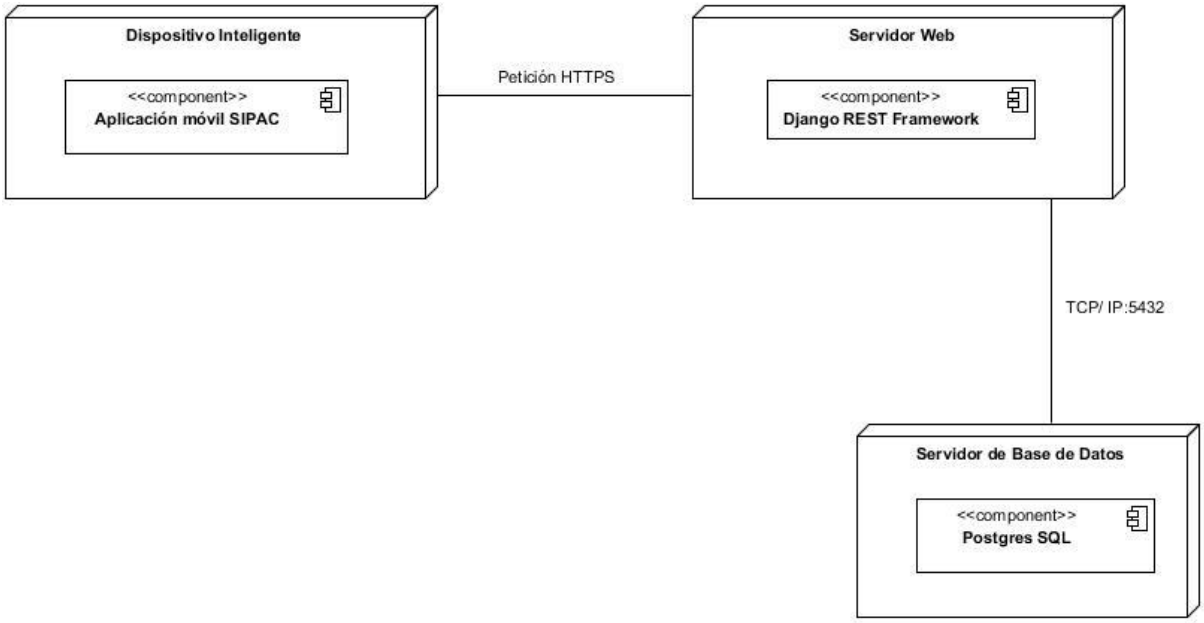


Figura 13. Diagrama de despliegue

3.3.1 Descripción de componentes

Nodo Dispositivo Inteligente: En este nodo (Figura 14) se instala la aplicación implementada desde la cual se consumirán y se brindarán los servicios REST a partir de los recursos que posee la API Django REST Framework.

Figura 14. Nodo Dispositivo Inteligente

Nodo Servidor Web: En este nodo (Figura 15) se guardarán todas las direcciones de los recursos a manejar, mediante los servicios REST.

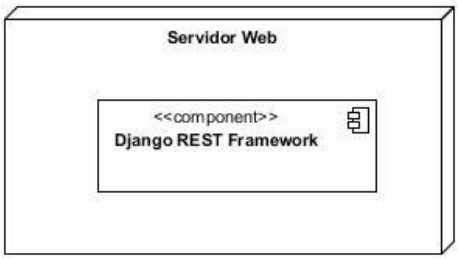


Figura 15. Nodo Servidor Web

Nodo Servido de Base de Datos SIPAC: En este nodo (Figura 16) se almacenan todos los datos referentes al módulo de Planificación.

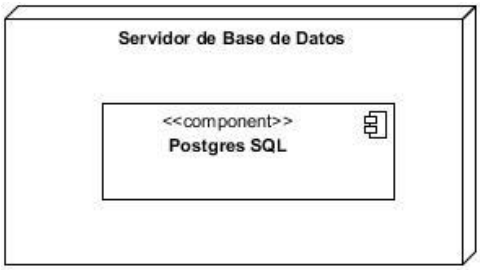


Figura 16. Nodo Servidor de Base de Datos SIPAC

3.4 Estándares de codificación de la aplicación Android

El estándar de codificación comprende los aspectos de generación de código, permite la legibilidad del código fuente, repercute directamente en lo bien que un programador comprende un sistema de software. El mantenimiento del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento.

3.4.1 Convenciones para variables Kotlin

Para la definición de variables se deben tener en cuenta las siguientes consideraciones:

- Las variables pueden ser mutables e inmutables.
- Se declaran indicando *val* o *var* según sean inmutables o mutables.
- Siempre que el compilador lo pueda inferir no es necesario especificar el tipo de dato.
  Ejemplo:

```
var mRepeatRule = 0
val user: Usuario = adapter.fromJson(post.body())
```

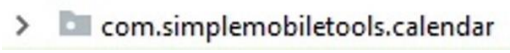
- Siempre que el compilador sea capaz de de detectar que no hay otra opción posible, el casting se hará automáticamente.
  Ejemplo:

```
val tipodplan = findViewById(R.id.PlanTP)
```

3.4.2 Estructura de paquetes

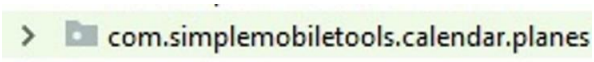
En proyectos de lenguaje mixto, los archivos fuente de Kotlin deben residir en la misma raíz de origen que los archivos fuente de Java y seguir la misma estructura de directorios (cada archivo debe almacenarse en el directorio correspondiente a cada declaración de paquete). Los nombres de los paquetes siempre se escriben con minúsculas y no usan guiones bajos. En los proyectos puros de Kotlin, la estructura de directorios recomendada es seguir la estructura del paquete con el paquete raíz común omitido (por ejemplo, todo el código del proyecto está en el paquete.

Ejemplo:



Sus subpaquetes, y archivos debe colocarse directamente debajo de la raíz de la fuente.

Ejemplo:



3.4.3 Nombres de archivos origen

Si un archivo de Kotlin contiene una sola clase (potencialmente con declaraciones de nivel superior relacionadas), su nombre debe ser el mismo que el nombre de la clase, con la extensión. kt. Si un archivo contiene múltiples clases, o solo declaraciones de nivel superior, elija un nombre que describa qué contiene el archivo y asígnele el nombre correspondiente. Use jorobas de camello con una primera letra mayúscula (por ejemplo, PlanFragment.kt).

3.4.4 Reglas de nombres

Para la definición de constantes debe tener presente las siguientes consideraciones:

- Los nombres de variables declaradas como constante deben ser todas en mayúsculas con palabras separadas por guion bajo.
- Se debe seguir las mismas convenciones que se usan para variables.

Ejemplo:

```
val NOMBRE_TABLA = "Plan"
```

Los nombres de propiedades de alto nivel u objeto que contienen objetos con comportamiento o datos variables deben usar nombres regulares de camello.

Ejemplo:

```
val param1: MutableList<Parametros> = mutableListOf()
```

Los nombres de propiedades que contienen referencias a objetos singleton pueden usar el mismo estilo de denominación object.

Ejemplo:

```
val plan: List<Plan_json>? = adapter.fromJson(ejecutar.body())
```

3.4.5 Nombres de funciones

Los nombres de funciones, propiedades y variables locales comienzan con letra minúscula y sin guiones bajos.

Ejemplo:

```
fun setupStartDate() {
var cancel = false
```

Los nombres de clases y objetos comienzan con una letra mayúscula, luego de los dos puntos las clases de las que hereda.

Ejemplo:

```
class AnadirPlan : SimpleActivity() {
```

3.4.6 Declaración de clases

El constructor principal permite declarar campos de forma dry incluyendo los setters y getters correspondientes.

Ejemplo:

```
data class Plan (
        val id:String?,
        val denominacion:String?,
        val autor:String?,
        val tipoPlan:String?,
        val porciento:String?,
        val fechainicio:String?,
        val fechafin:String?,
        val descripcion:String?,
        val createby:String?)
```

Debido a que Kotlin es capaz de crear campos, getters y setters directamente en el constructor, y que el constructor principal aparece antes de las llaves, hay casos en el que las clases value object, quedarían completamente vacías, así que, para evitar añadir más verbosidad, las llaves son opcionales.

3.4.7 Nomenclatura XML

Al usar vocabulario XML de Android, puedes crear rápidamente diseños de interfaz de usuario y de los elementos de pantalla que contienen, con una serie de elementos anidados.

Cada archivo de diseño debe contener exactamente un elemento raíz, que debe ser un objeto View o ViewGroup. Una vez que se haya definido el elemento raíz, se pueden agregar widgets u objetos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que defina tu diseño.

Ejemplo elemento raíz:

```
<?xml version="1.0" encoding="utf-8"?>
<com.simplemobiletools.calendar.views.MyScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/PlanScrollview"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
</com.simplemobiletools.calendar.views.MyScrollView>
```

Ejemplo de widgets u otros objetos de diseño:

```
<TextView
    android:layout_width="463dp"
    android:layout_height="wrap_content"
    android:layout_weight="3"
    android:text="Tipo de Plan"
    android:textAppearance="@style/AppTheme3"
    android:textSize="@dimen/day_text_size" />
```

Después de declarar el diseño en XML, se guarda el archivo con la extensión *.xml* en el directorio *res/layout/* del proyecto de Android para que pueda compilarse correctamente.
Cuando se compila la aplicación, cada archivo de diseño XML se compila en un recurso View. Se debe cargar el recurso de diseño desde el código de tu aplicación, en tu implementación de callback *Activity.onCreate().* Para hacerlo, llama a *setContentView(),* pásale la referencia a tu recurso de diseño en forma de: *R.layout.*layout_file_name*.*
Ejemplo:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_anadir_plan)
```

Cualquier objeto View puede tener un Id. con número entero asociado a él para identificar de forma exclusiva, por lo que para poder manejar cualquier vista o widget se le debe asignar un id único.
Ejemplo: `android:id="@+id/txtview_ff"`
Luego, crear una instancia del objeto View y capturarlo desde el diseño (generalmente en el método onCreate()):

Ejemplo: `val denominacion = findViewById<EditText>(R.id.PlanDeno)`

3.5 Pruebas de la aplicación Android

En la Figura 17 Contexto de la prueba de software, se muestra el contexto en que se realiza la prueba de software. La prueba de software se puede definir como una actividad en la cual un sistema o uno de los componentes se ejecuta en circunstancias previamente especificadas (configuración de la prueba), registrándose los resultados obtenidos. Seguidamente se realiza un proceso de Evaluación en el que los resultados obtenidos se comparan con los resultados esperados para localizar fallos en el software. Estos fallos conducen a un proceso de Depuración en el que es necesario identificar la falta asociada con cada fallo y corregirla, pudiendo dar lugar a una nueva prueba. Como resultado final se puede obtener una determinada Predicción de Fiabilidad, tal como se indicó anteriormente, o un cierto nivel de confianza en el software probado.

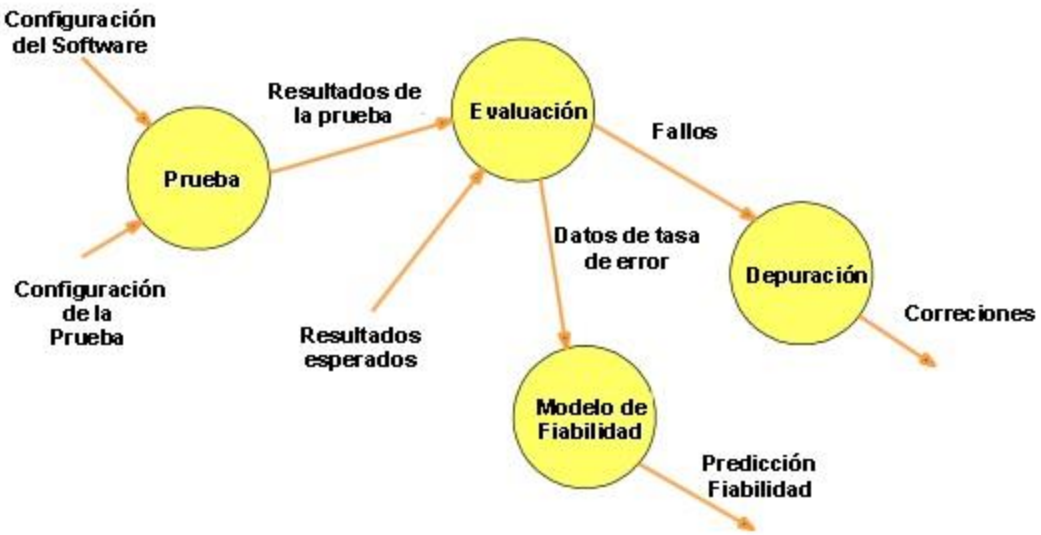


Figura 17. Contexto de la prueba de software

3.5.1 Diseño de las pruebas de la aplicación Android

Diseños de las pruebas: es la identificación de la técnica o técnicas de pruebas que se utilizarán para probar el software. Distintas técnicas de prueba ejercitan diferentes criterios como guías para realizar las pruebas.

Se selecciona para realizar las pruebas a la aplicación Android desarrollada para ejecutar las operaciones de Planificación, la estrategia de Pruebas Unitarias. Además, la librería JUnit desarrollada para probar el funcionamiento de las clases y métodos que componen la aplicación.

La prueba de unidad es la primera fase de las pruebas dinámicas y se realizan sobre cada módulo del software de manera independiente. El objetivo es comprobar que el módulo,

entendido como unidad funcional de un programa independiente, está correctamente codificado. En general, un módulo se entiende como un componente software que cumple las siguientes características: debe ser un bloque básico de construcción de programas, debe implementar una función independiente simples, no deberá tener más de 500 líneas de código.

En los últimos años se han desarrollado un conjunto de herramientas que facilitan la elaboración de pruebas unitarias en diferentes lenguajes. Dicho conjunto se denomina XUnit y JUnit es la herramienta utilizada para realizar pruebas unitarias en Java. El concepto fundamental de estas herramientas es el caso de prueba (test case), y la suite de prueba (test suite). Los casos de prueba son clases o módulos que disponen de métodos para probar los métodos de una clase o módulo concreto. Mediante las suites se organizan los casos de prueba, de forma que cada suite agrupa los casos de prueba de los módulos que están funcionalmente relacionados.

3.5.2 Generación de los casos de prueba

Los casos de pruebas representan los datos que se utilizarán como entrada para ejecutar el software a probar, determinan un conjunto de entradas, condiciones de ejecución y resultados esperados para un objetivo particular.

3.5.3 Definición de los procedimientos de la prueba

Especifica de cómo se va a llevar a cabo el proceso de pruebas.

Pasos para ejecutar las pruebas unitarias

1. Crear al proyecto *PruebaTests.*
2. Abrir el directorio *app*.
3. Abrir el directorio *src*.
4. Abrir el directorio *androidTest.*
5. Abrir el directorio kt
6. Abrir el directorio *com.ejemplo.pruebastest*
7. Crear el directorio tests y colocar el archivo ApplicationTest dentro del directorio tests creado.
8. Configurar en el archivo *build.gradle* para ello es en el código:

Agregar al final de la línea del código defaultConfig :

El identificador de pruebas.

```
testApplicationId "com.simplemobiletools.calendar.planes.test"
```

Clase de Android para ejecutar las pruebas.

```
testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
```

}

// Para generar el reporte la ruta donde se almacena.

testOptions

{

reportDir= "$project. buildDir/results/report" // Se define el directorio del reporte, buildDir es el directorio build que se encuentra en app.

resultsDir = "$project.buildDir/results" // Es un directorio donde se almacena el resultado de la ejecución al momento de compilar las pruebas. Básicamente es un xml que resume los resultados.

}

9. Ejecutar las pruebas y programarlas en el fichero **ApplicationTest** donde se crean las clases o casos de pruebas. Ejemplo:

```
public class AnadirPlanTest extends TestCase {
    public void testOnCreate() throws Exception {
        assertEquals(1,1);
    }
```

10. Compilar las pruebas en la terminar.

Aplicando los casos de pruebas generados previamente e identificando los posibles fallos producidos al comparar los resultados esperados con los obtenidos.

Se ejecutaron un total de 14 casos de pruebas para cada uno de los requisitos funcionales definidos en la aplicación propuesta. Ejemplo, el test AnadirPlantTest.kt, como se muestra en la Figura 18. Los resultados, en una primera iteración fueron satisfactorios, corroborando la correcta implementación de los métodos. Además, que previamente se ejecutaron en la aplicación web, donde se obtiene los datos, 22 casos de pruebas con resultados satisfactorios, utilizando Pytest, como se muestra en la Figura 19.

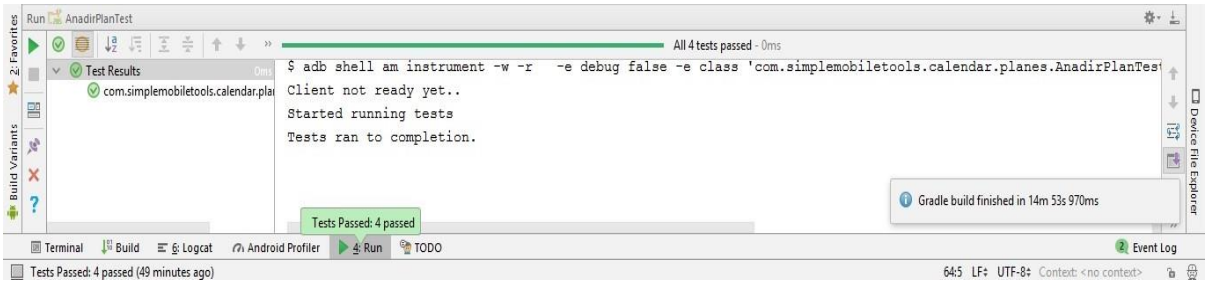


Figura 18. Resultado del test AnadirPlanTest.kt

Coverage for **calendario/models/Plan.py** : 100%

22 statements | 22 run | 0 missing | 0 excluded

```python
from django.db import models

from calendario.models.TipoPlan import TipoPlan
from calendario.models.Concepto import Concepto

from river.models.fields.state import StateField
from river.models.managers.wofkflow_object import WorkflowObjectManager


class Plan(models.Model):
    """Modelo que representa los planes en la aplicacion"""
    denominacion = models.CharField(max_length=100, null=True)
    tipoPlan = models.ForeignKey(TipoPlan, related_name='planes', on_delete=models.CASCADE)
    concepto = models.ForeignKey(Concepto, null=True, related_name='plan', on_delete=models.CASCADE)
    porciento = models.IntegerField(null=True)
    createby = models.IntegerField(null=True)
    fechainicio = models.DateField(null=True)
    fechafin = models.DateField(null=True)
    descripcion = models.CharField(max_length=500, null=True, blank=True)
    estado = StateField()

    objects = WorkflowObjectManager()
    autor = models.CharField(max_length=500, null=True, blank=True)

    class Meta:
        db_table = 'cal_plan'
        default_permissions = ('add', 'change', 'delete', 'read')

    def __str__(self):
        return self.denominacion
```

« index coverage.py v4.4.1, created at 2018-06-14 12:35

Figura 19. Resultados de las pruebas en la aplicación web.

3.6 Validación de la investigación de la aplicación Android

La portabilidad según la ISO 9126-1 es la Capacidad de transferencia de un producto de software de un entorno a otro. El entorno puede ser de tipo organizacional, hardware o software. Diferente software, diferente plataforma, código portátil o componentes cuyo reemplazo no afecta al resto de la aplicación.

Para evaluar la variable dependiente portabilidad, se utiliza el estándar internacional para la evaluación del software ISO/IEC 25023. La norma ISO/IEC 25023 - Measurement of system and software product quality: define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.

3.6.1 Medidas de portabilidad

Las medidas de portabilidad se usan para evaluar el grado de efectividad y eficiencia con el cual un sistema, producto o componente se puede transferir de un hardware, software u otro entorno operacional o de uso a otro.

3.6.2 Medidas de adaptabilidad

Las medidas de adaptabilidad se utilizan para evaluar el grado en que un producto o sistema puede adaptarse eficaz y eficientemente a un hardware, software u otros entornos operativos de uso diferentes o en evolución.

Tabla 5 Medidas de adaptabilidad

ID	Nombre	Descripción	Función de medición
PAd-1-G	**Adaptabilidad ambiental del hardware**	¿El software o el sistema es lo suficientemente capaz para adaptarse a diferentes entornos de hardware?	X = 1 - A / B A = Número de funciones que no se completaron o resultados insuficientes para cumplir con los requisitos durante las pruebas B = Número de funciones que han sido probadas en diferentes entornos de hardware
PAd-2-G	**Adaptabilidad ambiental del software del sistema**	¿El software o el sistema es lo suficientemente capaz para adaptarse a diferentes entornos de software del sistema?	X = 1 - A / B A = Número de funciones que no se completaron o resultados insuficientes para cumplir con los requisitos durante las pruebas B = Número de funciones que han sido probadas en diferentes entornos de software del sistema
NOTA 1: Cuando un usuario tiene que aplicar un procedimiento de adaptación que no haya sido previamente proporcionado por el software para una necesidad de adaptación específica, el esfuerzo del usuario requerido para adaptarse debe ser medido. NOTA 2: El software del sistema puede incluir sistemas operativos, middleware, sistema de gestión de bases de datos, compilador, sistema de gestión de red, etc.			
PAd-3-S	**Adaptabilidad del entorno operativo**	¿Con qué facilidad puede realizarse la prueba de funcionamiento desde el punto de mantenimiento?	X = 1 - A / B A = Número de funciones que no se completaron o resultados insuficientes para cumplir con los requisitos durante las pruebas operativas con el entorno del usuario B = Número de funciones que han

			sido probadas en diferentes entornos operacionales

3.6.3 Evaluación de las medidas de adaptabilidad

Tabla 6 Evaluación de las medidas de adaptabilidad

Aplicación Android para realizar las operaciones sobre los elementos de la planificación Total de requisitos probados: 14.				
No	**Ambiente del software**	**Ambiente del hardware**	**Adaptabilidad del entorno operativo**	**Resultado**
1	Versión de Android 4.4.2	Modelo: SCH-R530M Marca: Galaxy SII, metro PCS	Compilar la aplicación con los cambios realizados.	Se adapta el software y hardware, se puede compilar la aplicación.
2	Versión de Android 4.4.2	Modelo: CAPHG23-01 Marca: Selllure	Compilar la aplicación con los cambios realizados.	Se adapta el software y hardware, se puede compilar la aplicación.
3	Versión de Android 6.0	Modelo: ALE-L21 Marca**:** Huawei P8 Lite	Compilar la aplicación con los cambios realizados	Se adapta el software y hardware, Se puede compilar la aplicación pero no te muestra el listado de los errores.
4	Versión de Android 4.4.4	Modelo: GT-I9060M Marca: Samsung	Compilar la aplicación con los cambios realizados	Se adapta el software y hardware. Se puede compilar la aplicación
5	Versión de Android 5.1.1	Modelo: SCL-AL00 Marca: Huawei Honor	Compilar la aplicación con los cambios realizados	No se adapta el software y hardware. Se puede compilar la aplicación

6	Versión de Android	Modelo: K371 Marca: LG	Compilar la aplicación con los cambios realizados	No se adapta el software y hardware. Se puede compilar la aplicación
	Función de medición X = 1 - A / B X = 1 - 4/ 6 X = 0,34	Función de medición X = 1 - A / B X = 1 - 4/ 6 X = 0,34	Función de medición X = 1 - A / B X = 1 - 5/ 5 X = 0	Ambiente del software = 0,34 Ambiente del hardware = 0,34 Adaptabilidad del entorno operativo = 0

3.6.4 Medidas de instalabilidad

Las medidas de instalación se usan para evaluar el grado de efectividad y eficiencia con que un producto o sistema puede ser instalado y/o desinstalado con éxito en un entorno específico.

Tabla 7 Medidas de instalabilidad

ID	Nombre	Descripción	Función de medición
Pln-1-G	**Eficiencia del tiempo de instalación**	¿Qué tan eficiente es el tiempo real de instalación en comparación con el tiempo esperado?	X=$\sum_{i=1}^{n}(A_i/B_i)/\ n$ A_i = Tiempo de trabajo total dedicado a la instalación i B_i = Tiempo esperado para realizar una instalación i *n*= Número de instalaciones medidas
NOTA 1: X mayor que 1 representa una instalación ineficiente, y X menos de 1 representa una instalación muy eficiente. NOTA 2: El tiempo esperado para realizar una instalación puede basarse en datos históricos o promedios de la industria.			
Pln-2-G	**Facilidad de instalación**	¿Pueden los usuarios o encargados de mantenimiento personalizar el procedimiento de instalación para su conveniencia?	X = A / B A = Número de casos en los que un usuario logra personalizar el procedimiento de instalación B = Número de casos en los que un usuario intentó personalizar el

			procedimiento de instalación para la conveniencia del usuario
NOTA Estos cambios de procedimiento de instalación pueden ser reconocidos como personalización de la instalación por el usuario.			

3.6.4 Evaluación de las medidas de instalabilidad

Tabla 8 Evaluación de las medidas de instalabilidad

Aplicación Android para realizar las operaciones sobre los elementos de la planificación Total de requisitos probados: 14.		
Sistema operativo PC	**Eficiencia del tiempo de instalación**	**Facilidad de instalación**
Marca Toshiba Procesador: AMD A8 Memoria: 3 GB	X=$\sum_{i=1}^{n}(A_i/B_i)$/ n X = ((2m 856ms/4m) /1 X= 0,5 m con 856 ms Resultado: eficiente	X = A / B X= 14/14 X=1 Resultado: eficiente
Marca Lenovo Memoria: 7.6 GB Procesador: Intel Core i5-3230M	X=$\sum_{i=1}^{n}(A_i/B_i)$/ *n* X = (1m 32s/4m)/n X= 0,25 con 32s Resultado: muy eficiente	X = A / B X= 14/14 X=1 Resultado: eficiente
Marca Asus Procesador: Core i3-2120	X=$\sum_{i=1}^{n}(A_i/B_i)$/ *n* X = (3m 447ms)/4m)/1 X= 0,75 con 447ms Resultado: poco eficiente	X = A / B X= 14/14 X=1 Resultado: eficiente

3.6.5 Medidas de reemplazabilidad

Las medidas de reemplazabilidad se usan para evaluar el grado en que un producto puede reemplazar a otro producto de software especificado para el mismo propósito en el mismo ambiente.

Tabla 9 Medidas de reemplazabilidad

ID	Nombre	Descripción	Función de medición
PRe-1-G	**Similitud de uso**	¿Qué proporción de funciones de usuario del producto reemplazado se puede realizar sin ningún aprendizaje adicional o solución?	X = A/B A = Número de funciones de usuario que se pueden realizar sin ningún aprendizaje adicional o solución alternativa B = Número de funciones de usuario en el producto de software reemplazado
NOTA Las funciones de usuario son aquellas que el usuario puede llamar y utilizar para realizar las tareas previstas, incluidas las interfaces de usuario.			
PRe-2-S	**Equivalencia de la calidad del producto**	¿Qué proporción de las medidas de calidad se satisface después de reemplazar el producto de software anterior por éste?	X = A / B A = Número de medidas de calidad del nuevo producto que son mejores o iguales al producto sustituido B = Número de medidas de calidad del producto de software sustituido que son relevantes
NOTA Algunas de las cualidades de los productos críticos correspondientes a la intercambiabilidad son la interoperabilidad, la seguridad y la eficiencia del rendimiento.			
PRe-3-S	**Inclusión funcional**	¿Pueden las funciones similares ser usadas fácilmente después de substituir el producto de software anterior por éste?	X = A/B A = Número de funciones que producen resultados similares a los anteriores B = Número de funciones que deben utilizarse en el producto de software sustituido
PRe-4-S	**Capacidad de reutilización/importación de datos**	¿Pueden usarse los mismos datos después de reemplazar el producto de software anterior por éste?	X = A / B A = Número de datos que se pueden utilizar continuamente como antes

			B = Número de datos que se utilizarán continuamente en el producto de software sustituido

Aplicación Android para realizar las operaciones sobre los elementos de la planificación.			
Similitud de uso	**Equivalencia de la calidad del producto**	**Inclusión funcional**	**Capacidad de reutilización/importación de datos**
X = A/B X= 14/14 X= 1	X = A/B X= 14/4 X= 3,5 Relevantes: Listar plan Adicionar plan Modificar plan Eliminar plan	X = A/B X= 14/14 X=1	X = A/B X= 14/7 X=2 Nota: En el sistema Android no se carga toda la información contenida en el sistema web, siempre es el 50% de la misma.

3.6.6 Evaluación de la portabilidad aplicando la norma ISO/IEC 25023

Tabla 10 Evaluación de la portabilidad aplicando la norma ISO/IEC 25023

Características	**Ponderación**	**Resultado**
Adaptabilidad	Ambiente del software = 0,34 Ambiente del hardware = 0,34 Adaptabilidad del entorno operativo = 0	**Aceptable**: los resultados de la variable x siempre fue menor que 1. **Recomendación** Requiere un plan de acción para fortalecer los aspectos críticos, del ambiente del software y hardware.
Instalabilidad	**Eficiencia del tiempo de instalación =** Eficiente, muy eficiente, poco eficiente. **Facilidad de instalación =** Eficiente	**Bueno**: la eficiencia del tiempo siempre fue menor que 1 y la facilidad de la instalación 1. Con resultados de eficiente para la instalación. **Recomendación** Fortalecer algunas de las características de la aplicación.
Reemplazabilidad	**Similitud de uso**= 1 **Equivalencia de la calidad del producto** = 3,5 **Inclusión funcional** =1 **Capacidad de reutilización/importación de datos** =2	**Poco aceptable**: Es poco aceptable porque en la aplicación web se muestra toda la información mientras que en la aplicación Android por las características solamente se muestran el 50% de la misma. Aunque se implementan todos los requisitos contenidos en la aplicación Web.

3.6.7 Conclusiones de la Evaluación de la portabilidad aplicando la norma ISO/IEC 25023

Para obtener los valores de las características de portabilidad, después de la aplicación de las medidas de evaluación, se procede adaptar la fórmula para el cálculo de las subcaracterísticas: adaptabilidad, instalabilidad y reemplazabilidad, y la característica de portabilidad; propuestas en la norma ABNT NBR ISO/IEC 14598-6 Anexo C (ISO, 2001) **Vc = ∑ Vsc/nsc y Vsc = ∑ m/ (n-nd).** En donde: Vc es el valor medio de la característica; Vsc es el valor medido de la subcaracterística; nsc es el número de subcaracterísticas; m es 1, si la respuesta es positiva, en caso contrario es 0; n es el número total de medidas; nd es el número de preguntas descartadas.

Tabla 11 ABNT NBR ISO/IEC 14598-6 Anexo C (Informativo)

Subcaracterísticas	Vc	Vsc	nsc	m	n	nd
		0,25		1	6	2
		0,25		1	6	2
		0,16		1	6	0
Adaptabilidad	0,22	0,66	3			
Instalabilidad	0,3	0,3	1	1	3	0
		0,07		1	14	0
		0,25		1	14	10
		0,07		1	14	0
		0		0	14	7
Reemplazabilidad	0,09	0, 39	4			
Portabilidad	0,61					

El resultado obtenido basado en la evaluación para la subcaracterística propuesta en la norma ABNT NBR ISO/IEC 14598-6 Anexo C (Informativo), aplicada a la característica portabilidad, es de un valor Vc=0,61, representando un 61%. El valor significa que la portabilidad del sistema SIPAC, en dispositivos móviles con sistema operativo Android en versiones 4.4.2 o superior, está representada por un 61%. A partir de la situación antes descrita, se evidencia que se ha mejorado la portabilidad del sistema SIPAC, contribuyendo en un 61% en portabilidad, para dispositivos móviles con sistema operativo Android.

3.7 Conclusiones parciales

Luego de la implementación y validación de la aplicación propuesta se arriba a las siguientes conclusiones:

Mediante el modelo de componentes se representó una vista estática de la aplicación, mostrando la organización y dependencias que existe entre los componentes físicos que se necesitan para ejecutar la aplicación propuesta.

El modelo de base de datos elaborado, permitió representar la abstracción, percepción y conocimiento en un sistema de información formado por un conjunto de objetos denominados entidades y relaciones. El uso de estándares de codificación y estilos de programación, permitió el entendimiento del código por otros programadores que no sean del equipo de desarrollo, para el mantenimiento de la aplicación propuesta. El modelo de despliegue permitió comprender la distribución física.

La ejecución de las pruebas utilizando la herramienta JUnit, permitió demostrar que las funciones de la aplicación propuesta son operativas, mediante los casos pruebas ejecutados. La aplicación de las medidas de las subcaracterísticas; adaptabilidad, instalabilidad y reemplazabilidad de la norma ISO/IEC 25023, para la evaluación de la portabilidad en dispositivos móviles, obtuvo que la aplicación propuesta mejora la portabilidad del sistema SIPAC.

CONCLUSIONES GENERALES

Con la investigación e implementación descritas, se logró dar cumplimiento a las tareas de investigación, así como al objetivo planteado inicialmente obteniendo como resultado una aplicación que cumple con todos los requisitos y la calidad requerida, lo cual permite arribar a las siguientes conclusiones:

- Se analizaron los principales conceptos relacionados con la planificación, estableciendo los elementos fundamentales regidos en la Instrucción no.1 del Presidente de los Consejos de Estado y de Ministros, para los diferentes tipos de planes: anual, mensual e individual. Se identificaron las subcaracterísticas de la norma ISO 25000, para la característica de la portabilidad, que permitió comprender las medidas en la validación de la investigación. Se identificaron los servicios REST, que se utilizaron en la implementación de la aplicación. Se selecciona Android como sistema operativo para el desarrollo de la solución propuesta, considerando sus características, según el estudio realizado de la cuota de mercado del SO móvil en Cuba.
- Se analizaron las diferentes tecnologías que realizan las operaciones de planificación de actividades, que permitió obtener elementos para el diseño de la interfaz visual, así como la distribución de la información y las funcionalidades principales de una aplicación de este tipo. Se analizaron diferentes metodologías ágiles por las características que presenta este tipo de desarrollo de software, seleccionando Variación AUP para la UCI, en correspondencia con los lineamientos de desarrollo de software de la universidad. Se identificaron las herramientas y lenguajes de programación que permitieron el desarrollo de la aplicación propuesta.
- El modelo conceptual elaborado, permitió comprender los conceptos asociados al dominio del negocio y sus relaciones. Los requisitos funcionales identificados, a través del análisis de la documentación existente y las entrevistas realizadas, permitió definir lo que el sistema debe hacer, así como las características requeridas del software (requisitos del producto), de la organización que se desarrolla el software (requisitos organizacionales) o de fuentes externas. El diseño de la aplicación, permitió representar las clases con sus atributos y las relaciones entre las mismas, utilizadas en la implementación de la aplicación propuesta. La arquitectura definida, permitió describir la estructura general de la aplicación propuesta, así como la implementación de un consumidor de servicios utilizando los métodos GET, POST, PUT y DELETE definidos por el protocolo HTPP.
- La implementación de la aplicación para dispositivos móviles desarrollada, permitió obtener un producto para brindar y consumir los servicios REST del módulo de Planificación en el SIPAC. La ejecución de las pruebas unitarias con la herramienta JUnit, con resultados satisfactorios, permitió corroborar que los requisitos funcionales están

correctamente implementados, para un conjunto de entradas, condiciones de ejecución y resultados esperados. El resultado obtenido basado en la evaluación para las subcaracterísticas; adaptabilidad, instalabilidad y reemplazabilidad, propuesta en la norma ABNT NBR ISO/IEC 14598-6, aplicada a la característica portabilidad; evidencia que se ha mejorado la portabilidad del sistema SIPAC. Contribuyendo la aplicación propuesta, en un 61% a la portabilidad para dispositivos móviles, con sistema operativo Android.

RECOMENDACIONES

Se recomienda el desarrollo del desglosar actividades del plan y del relacionar las actividades a un plan en el módulo de calendario.

BIBLIOGRAFÍA

About SQLite. 2018. About SQLite. SQLite. [En línea] 2018. [Citado el: 15 de mayo de 2018.] https://www.sqlite.org/about.html.

Abrahamsson, Pekka, y otros. 2004. *Mobile-D: an agile approach for mobile application development.* 2004.

Alonso, Arturo Baz, Ferreira Artime, Irene y Álvarez Rodríguez, María. 2016. *Dispositivos móviles.* s.l. : Universidad de Oviedo, 2016.

Android Developers. 2018. Android Developers. [En línea] 2018. [Citado el: 10 de abril de 2018.] https://developer.android.com/.

Camarero, Julio, y otros. 2003. *Metodología de desarrollo ágil para sistemas móviles Introducción al desarrollo con Android y el iPhone.* 2003.

CECM. 2009. *Intrucción 11.* 2009.

Clements, Mark, Kazman, Rick y Klein, Mark. 2001. *Evaluation the Software Architecture.* 2001.

2018. Diagrama de Despliegue - manuel.cillero.es. [En línea] 2018. [Citado el: 17 de mayo de 2018.] https://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-despliegue/. .

Django REST framework. 2018. Django REST framework. [En línea] 2018. [Citado el: 6 de enero de 2018.] http://www.django-rest-framework.org/.

2018. Evernote. [En línea] 2018. [Citado el: 3 de marzo de 2018.] evernote.com.

2018. Exes Cursos oracle,cursos java,master oracle,master java,formación en informatica. [En línea] 2018. [Citado el: 10 de abril de 2018.] http://www.mundolinux.info/que-es-xml.htm..

Gonzalez, Mónica. 2014. SlideShare. [En línea] 2014. https://es.slideshare.net/MonicaGlez1/ingeniera-de-software-30069503.

Gradle Inc. 2018. Gradle Docs. [En línea] 2018. [Citado el: 10 de mayo de 2018.] https://docs.gradle.org/current/userguide/userguide.html#introduction..

IAVILAE. 2013. Proyecto Simio. *Proyecto Simio.* [En línea] 2013. http://www.proyectosimio.com/es/programacion-android-base-de-datos-i-modelo-vista-.

ISO. 2001. *ISO/IEC 14598-6,Software engineering — Product evaluation — Part 6.* 2001.

2017. Kotlin Blog. [En línea] 2017. https://blog.jetbrains.com/kotlin/.

Kotlin.es. 2018. Kotlin.es La primera comunidad de Kotlin en español. [En línea] 2018. [Citado el: 7 de diciembre de 2018.] https://kotlin.es.

Kotlinlang. 2018. Kotlin. [En línea] 2018. [Citado el: 7 de abril de 2018.] https://kotlinlang.org/.

BIBLIOGRAFÍA

Kotonya, Gerald y Sommerville, Ian. 1998. *Requirements Engineering: Processes and Techniques.* 1998.

Larman, Craig. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 1999.

Método ágil híbrido para desarrollar software en dispositivos móviles. **Mundaca, Ignacio Leiva y Villalobos Abarca, Marcos. 2015.** 2015.

MoleQla: revista de Ciencias de la Universidad Pablo de Olavide, Patrón Modelo-Vista-Presentador (MVP). **Diéguez, D.S. 2015.** 20, 2015.

Norma ISO 25000. 2004. *Norma ISO 25000.* 2004.

Ochando, Dr. Manuel Bláquez. 2014. Fundamentos y diseño de Base de Datos. [En línea] 2014. http://ccdoc-basesdedatos.blogspot.com/2013/02/modelo-entidad-relacion-er.html..

OMG. 2018. OMG. *OMG.* [En línea] 2018. http://www.omg.org/technology/readingroom/XML.htm.

Paradigm, Visual. 2018. Visual Paradigm. *Visual Paradigm.* [En línea] 2018. https://www.visual-paradigm.com/.

Patrón Modelo-Vista-Controlador. **Romero, Yanisleydis Fernandez y Díaz Gonzalez, Yanette . 2012.** 1, La Habana : Revista Digital de las Tecnoligías de la Informacón y las Comunicaciones, 2012, Vol. 11.

Patrón Modelo-Vista-Presentador (MVP).

Pressman, Roger S. 2010. *Ingeniería de software un enfoque práctico Séptima edición.* 2010.

Sánchez, Tamara Rodríguez. 2015. *Metodología de desarrollo para la Actividad productiva de la UCI.* 2015.

Solución para lograr la interoperabilidad de Odoo con otros sistemas. **Clauda Bravo Batista, Janier Treto Portal, Nemurys Silega Martínez. 2017.** La Habana : s.n., 2017.

Sommerville, Ian. 2005. *Ingenieria del Software Séptima Edición.* 2005.

2018. Sparx Systems - Tutorial UML 2 - Diagrama de Despliegue. [En línea] 2018. [Citado el: 2 de abril de 2018.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html. .

Sparx Systems. 2018. El Modelo de Componenetes UML por Sparx Systems. [En línea] 2018. [Citado el: 13 de mayo de 2018.] http://www.sparxsystems.com.ar/resources/tutorial/component_model.html..

Statcounter. 2018. statcounter GlobalStats. [En línea] 2018. [Citado el: 3 de mayo de 2018.] http://gs.statcounter.com/os-market-share/mobile/cuba.

Studio, Android. 2017. Android Studio. [En línea] diciembre de 2017. https://developer.android.com.

BIBLIOGRAFÍA

Technology Radar . **ThoughtWorks, Comite Consultor de Tecnología de. 2017.** s.l. : ThoughtWorks, 2017, Vol. Vol.17.

2018. Todoist. [En línea] 2018. [Citado el: 6 de abril de 2018.] https://es.todoist.com/.

2018. Toodledo. [En línea] 2018. [Citado el: 6 de abril de 2018.] https://www.toodledo.com/.

UCI. *Metodología de desarrollo para la actividad productiva de la UCI.* La Habana : s.n.

Visual Paradigm . 2018. Visual Paradigm - Leading UML, BPMN, EA, Agile and Project Management Software. [En línea] 2018. https://www.visual-paradigm.com/.

XML | Object Management Group. 2018. XML | Object Management Group. [En línea] 2018. [Citado el: 1 de abril de 2018.] http://www.omg.org/technology/readingroom/XML.htm.

GLOSARIO DE TÉRMINOS

Actividades: conjunto de operaciones o tareas, propias de una persona o entidad, destinadas para cumplir determinado(s) objetivo(s).

Acuerdo: decisión tomada en común por dos o más personas, por una junta o asamblea. También se denomina así a una resolución de organizaciones, instituciones, empresas.

Anexo: fichero que se anexa a un elemento de la planificación, el mismo puede contener cualquier objeto digitalizado de los siguientes: documentos (PDF, EXCEL, DOC) o imágenes.

Área de Resultados Clave (ARC): término empleado para definir un resultado cuyo logro implica la participación de dos o más elementos (departamento, dirección, grupo de trabajo, etc.) de la estructura funcional de una entidad. Tal logro es esencial para que la organización cumpla su misión.

Categoría de la actividad: elemento de clasificación para el ordenamiento de las actividades con el fin de mejorar su organización y control.

Criterio de medida: condición que debe cumplir una o varias medidas para que el indicador al que tributan sea evaluado con un resultado determinado.

Entidad: colectividad considerada como unidad, especialmente, cualquier corporación, compañía, institución, etc. En el campo de la planificación se aborda genéricamente como organización; referida a los disímiles grupos humanos conformados para cumplir sus correspondientes objetivos explícitos e implícitos.

Factores que Intervienen en la Planificación (FIP): documento que contiene elementos que desencadenan cambios en los planes, el mismo es emitido por el nivel superior o propio.

Indicador: puntos de referencia, que brindan información cualitativa o cuantitativa, conformada por una o varias medidas, que permiten seguir el desenvolvimiento de un proceso y su evaluación, y que deben guardar relación con el mismo.

Plan: modelo sistemático que se elabora antes de realizar una acción, con el propósito de dirigirla y encausarla. En este sentido, un plan también es un documento que precisa los detalles necesarios para realizar una misión.

Planificación: acciones llevadas a cabo para realizar planes y proyectos de diferente índole. La planificación ejecuta los planes desde su concepción y si es el caso, se encarga de la operación en los diferentes niveles y amplitudes de la planeación.

GLOSARIO DE TÉRMINOS

Planificar: acción de crear o proyectar metas donde se interrelacionan los factores tiempo, espacio, recursos y personas.

Principio de consecución: condición que debe cumplir uno o varios indicadores para que el objetivo al que tributan sea evaluado con un resultado determinado.