Universidad de las Ciencias Informáticas

Centro de Gobierno Electrónico

Facultad 3

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

Sistema para la gestión de Acuerdos, Decisiones e Indicaciones versión móvil (XADIM).

Autor:

Asiledy Manzanares Rodríguez

Tutores:

Ing. Carlos Miguel Morera Pérez

Ing. Heidy Infante Morales

Co-tutores:

Ing. Michel Álvarez Cancio

Ing. Michel Sariol Fernández

La Habana, Cuba

Junio de 2017

"Año 59 de la Revolución"

Declaración de auditoría:

Declaro ser autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas, para que haga el uso que estime pertinente.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2017.

Autora:

Asiledy Manzanares Rodríguez.

_____ Firma

Tutores:

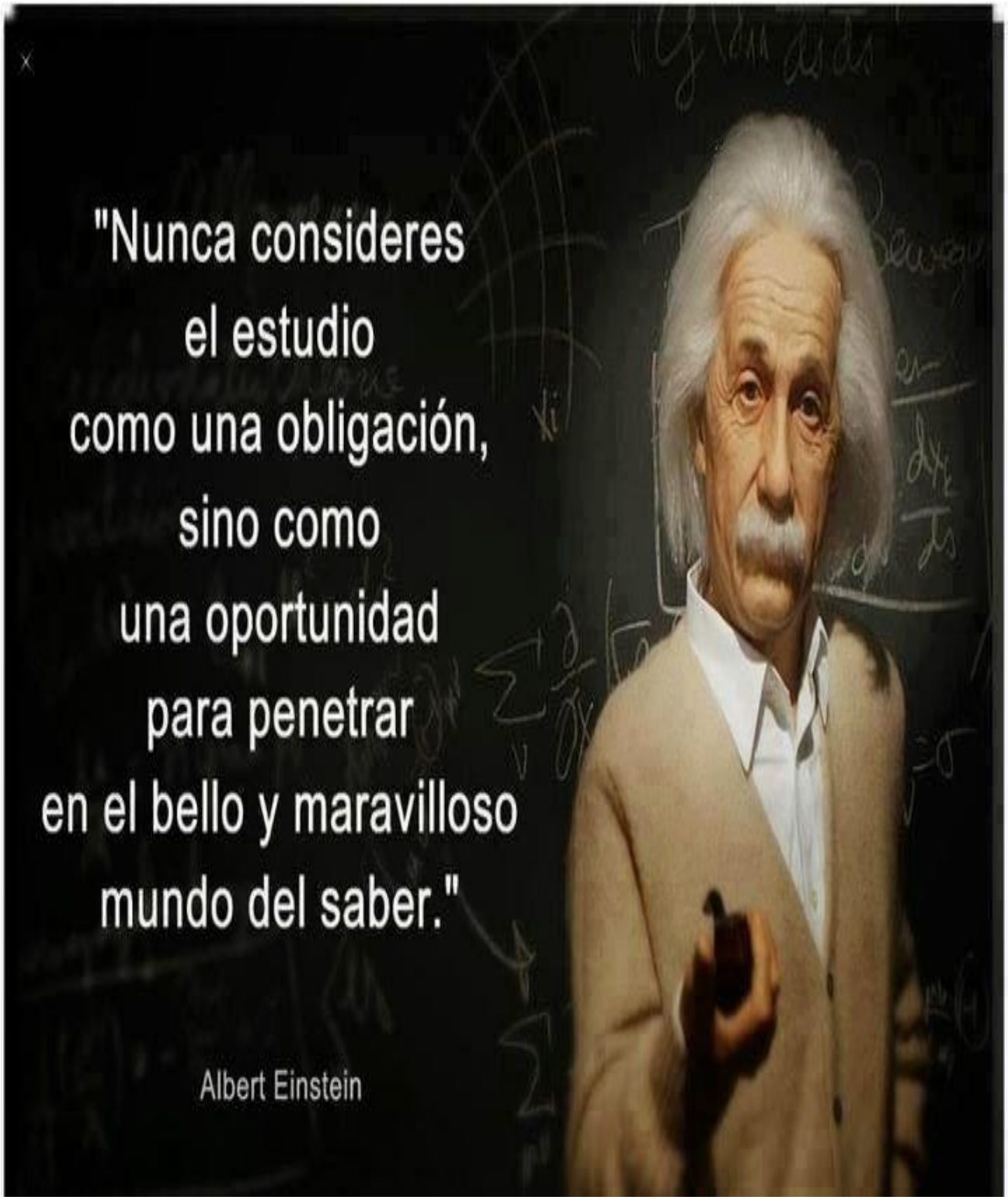
Ing. Heidy Infante Morales

_____ Firma

Ing. Carlos Miguel Morera Pérez

_____ Firma

Frase


"Nunca consideres
el estudio
como una obligación,
sino como
una oportunidad
para penetrar
en el bello y maravilloso
mundo del saber."
Albert Einstein

Agradecimientos

A mis padres por su apoyo incondicional y por permitirme llegar a donde estoy y ser quien soy, en especial a mi madre por guiarme en mis estudios y demostrarme que siempre se puede ser mejor cada día, por contentarse cada vez que aprobaba una prueba como si fuera ella quien estuviera cursando la universidad.

A mis profesores, a todos y cada uno de ellos por influir de una manera u otra en mi formación como profesional y como persona, en especial a mi profesor de introducción a la programación Juniel.

A los miembros de mi tribunal por sus consejos y ayuda durante este agitado proceso.

A mi tutor por soportar mi carácter y mis pensamientos que en la mayoría de las ocasiones son diferentes a los de él.

A mi tutora que, aunque no estuvo hasta al final me fue de gran ayuda al comienzo de este proyecto.

A mi amor por soportarme en esos días difíciles que con la “tesis” aumentaron y que, a pesar de no estar ahí físicamente en estos últimos tiempos cerquita mío como antes, tuvo paciencia para apoyarme emocionalmente, te amo.

A mis amigas y compañeras de apartamento de primer año que, aunque ahora no compartamos casa siempre las tengo presente, Arisleydis y las Claudias. Creo que el mejor año de la universidad fue nuestro primer año en la UCI, hacíamos que la estancia a pesar de tanto estudio y diversiones fuera un lugar especial, nuestro hogar. Por no hablar de las fiestas, por dios en la vida había dado tanto chucho, divertido tanto con solo escuchar una música y coger el sereno. Cuando empecé en la universidad muchas personas me dijeron que las amistades hechas en el pre-universitario eran para toda la vida, por eso pensé que ya con ellas tenía suficiente, pero entonces llegaron ustedes con esas características que hacen que cualquiera se sienta querido por ustedes, que cualquiera se divierta a su lado, cierto chicas son las mejores!!!. Gracias por ser mis amigas y por compartir conmigo ese encuentro con la universidad, las he echado mucho de menos estos años.

A toda mi familia por ayudarme en lo que sea que me hiciera falta, por tal de permitir que me concentrara en mis estudios y no me preocupara por otra cosa que no fuera graduarme y ser una buena profesional, para lograrlo y convertirme en el orgullo de la familia. Sin ellos no estaría ni a la mitad del camino, es más sin ellos no sería nadie, pese a las diferencias, a los caracteres, a los momentos y circunstancias

difíciles, aunque a la familia no nos permiten escogerla si a mí me dieran esa posibilidad, los volvería a escoger, gracias por todo, los amo.

Dedicatoria

Este trabajo de diploma se lo dedico principalmente a mi madre Idelisa y a mi papá Víctor Manuel, todas mis metas serán dedicadas a ustedes, este es solo el principio...

A mi familia y amigos por siempre estar presente.

Resumen

El proceso de informatización en Cuba, ha permitido un mayor acceso a la información y la automatización de los procesos en las empresas. Uno de estos procesos es la gestión de los acuerdos, decisiones, tareas o indicaciones que se toman por parte de los ejecutivos de una empresa. Para ello el Centro de Gobierno Electrónico desarrolla el Sistema para la gestión de Acuerdos, Decisiones e Indicaciones.

Luego de un análisis del funcionamiento de este sistema, se identificaron problemas de disponibilidad y visibilidad de la información. Debido a la gran cantidad de contenido de valor a mostrar y funcionalidades que proporciona, además de lograr el acceso a los datos de forma *offline*[1]. Por lo que se implementa una versión móvil del mismo para sistemas operativos Android. La cual permite consultar, buscar, visualizar detalles, mostrar notificaciones y anular los acuerdos que estén registrados. Para ello se escogen las herramientas y tecnologías que fueron empleadas en la implementación del software. Se aplicaron las pruebas de caja blanca (técnica del camino básico) y caja negra (partición de equivalencia) a la propuesta de solución tanto a nivel de diseño como de implementación. También se validan un conjunto de indicadores de visibilidad y disponibilidad mediante la técnica de juicio de expertos, demostrando el cumplimiento de los objetivos trazados.

Palabras claves: Android, XADIM, gestión de acuerdos, aplicaciones móviles.

[1] Offline: Se le denomina al hecho de no estar conectado, fuera de red.

Índice

Índice de ilustraciones

Índice de tablas

Introducción

La evolución constante de las Tecnologías de la Información y las Comunicaciones (TIC) ha significado a escala mundial un salto acelerado, evidenciándose en las distintas esferas de la sociedad, tales como la salud, la educación, las comunicaciones, el comercio, la industria, entre otras. Ejemplo del uso y evolución de estas tecnologías se encuentran los dispositivos móviles (1), quienes compiten con las computadoras de escritorio y los portátiles o laptops. Siendo estos de diferentes tamaños y permitiéndole a los usuarios realizar las mismas funciones, incluso hasta más rápido, pero de manera remota y más cómodamente, ya sea desde su propio teléfono inteligente o tableta, sin necesidad de estar atado a una computadora.

De igual manera avanzan los sistemas operativos, quienes permiten el funcionamiento de los dispositivos móviles. En la actualidad existen varios sistemas operativos para dispositivos móviles, entre los más destacados están: Blackberry OS, Windows Phone, iPhone OS y Android (2). Estos sistemas son más simples que los empleados en las computadoras y están orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos. Siendo Android, el sistema operativo más utilizado, según se evidencia en las gráficas del 2016 (Ver figura 1) (3) (4), el cual se ha mantenido en la mayor cuota desde el año 2011. Android ha ido evolucionando como sus propias versiones lo indican, hasta convertirse en un sistema operativo rápido, fácil de usar, así como de una apariencia variada y agradable a la vista de todos (5).

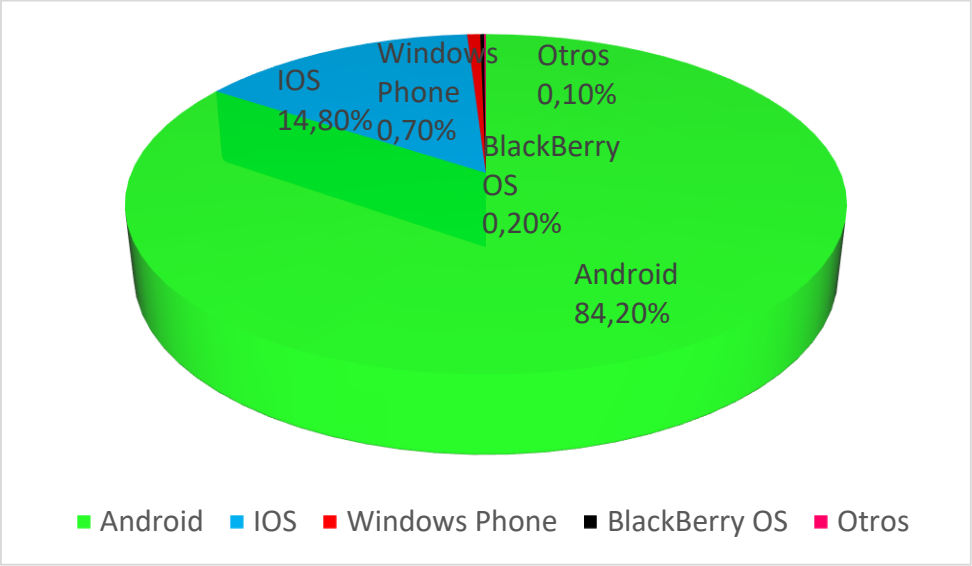


Ilustración 1: Cuota en el mercado del uso de los sistemas operativos para dispositivos móviles 2016. (3)

En Cuba, la utilización de la tecnología móvil ha crecido considerablemente en los últimos años (6). Donde este avance ha permitido un mayor acceso a la información y la automatización de los procesos en las empresas.

Uno de estos procesos es la gestión de los acuerdos, decisiones, tareas o indicaciones que se toman por parte de los ejecutivos de una empresa. La decisión o tarea posee un emisor, un receptor o receptores y un coordinador. Este último es uno de los receptores y tiene como función mediar entre el emisor y el receptor para hacer posible el cumplimiento correcto y en tiempo de la tarea asignada. Además, las tareas poseen una fecha de vencimiento que indica el tiempo límite para llevar a cabo el cumplimiento de las mismas.

Para ello el Centro de Gobierno Electrónico (CEGEL), perteneciente a la Universidad de las Ciencias Informáticas ha desarrollado, a solicitud del Fiscal General de la República de Cuba, un sistema para gestionar los acuerdos, decisiones e indicaciones que son emitidas a diario por el Fiscal General y fiscales jefes. Como resultado, se obtuvo una aplicación web que brinda estos servicios.

Con el objetivo de que un sistema similar sea utilizado por los cuadros administrativos y políticos de Cuba se realiza, también por CEGEL, el producto Sistema para la gestión de Acuerdos, Decisiones e Indicaciones (XADI) versión 1.0. Este es un sistema web que incluye las funcionalidades y características del sistema realizado para la Fiscalía, pero de manera genérica y adaptable a las demás entidades, empresas e instituciones. Este sistema permitirá a los usuarios mantener un control de los acuerdos que se toman y un flujo de información entre el emisor y los ejecutores de las tareas. Las búsquedas de la documentación resultarán más sencillas y los proveerá de una herramienta de evaluación y análisis.

Los acuerdos son tomados en reuniones, asambleas y en disímiles lugares donde no siempre se tienen las herramientas y condiciones para su gestión en el sistema XADI, por lo que los usuarios no pueden consultar e insertar los datos del acuerdo. Dado esto deben utilizar medios alternativos para salvar la información que más tarde será transmitida. De esta manera también se dificulta poder consultar el estado en que estos se encuentran y la posibilidad de notificar los vencimientos de término.

Además, al acceder al actual (XADI) desde dispositivos móviles (aunque el sistema cuenta con un diseño adaptable) se dificulta el acceso y la visibilidad de los datos, debido a la cantidad de contenidos de valor a mostrar y funcionalidades que proporciona. Puede resultar compleja para el usuario la navegación por la plataforma, pues en ocasiones es necesario acercar y reducir la interfaz del sitio web para leer o acceder a un elemento, o utilizar una barra de desplazamiento debido a la densidad de contenido. Como

consecuencia de esto, la visibilidad queda reducida ya que el usuario en cuestión puede demorarse en la búsqueda de un elemento o resultar complejo el acceso a este.

La situación problemática antes descrita genera el siguiente **problema de investigación:**

¿Cómo elevar la disponibilidad y visibilidad de la información del sistema de gestión de acuerdos, decisiones e indicaciones XADI?

Teniendo en cuenta el problema antes mencionado se define como **Objeto de estudio:** el proceso de gestión de la información.

Para ello se identifica como **campo de acción:** el proceso de gestión de la información de los acuerdos, decisiones e indicaciones desde dispositivos móviles.

Se propone como **Objetivo general:** desarrollar una aplicación informática para dispositivos móviles que eleve la disponibilidad y visibilidad de la información del sistema de gestión de acuerdos, decisiones e indicaciones XADI.

Desglosado en los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación para fundamentar la metodología, herramientas y lenguajes a utilizar.
- Desarrollar el Sistema para la gestión de Acuerdos, Decisiones e Indicaciones versión móvil, teniendo en cuenta los elementos de diseño para móviles.
- Validar las variables de la investigación mediante panel de expertos y verificar el correcto desarrollo del sistema mediante pruebas de caja blanca y caja negra.

Se define como **idea a defender**:

Con el de desarrollo de una aplicación informática para dispositivos móviles, se elevará la disponibilidad y visibilidad de la información del sistema de gestión de acuerdos, decisiones e indicaciones XADI.

Métodos de investigación (7)**:**

Dentro de los métodos teóricos utilizados se encuentra el **histórico-lógico**, se utiliza para estudiar los sistemas de gestión de la información, así como el impacto y la aceptación que estos tuvieron,

tomándose como referencia para apoyar la realización de la solución y en qué medida facilitaría la investigación.

El método **análisis-síntesis** posibilita el análisis de las teorías, documentos y materiales, permitiendo así la extracción de los elementos más importantes sobre el proceso de gestión de la información en los dispositivos móviles.

Dentro de los métodos del nivel empírico del conocimiento se destacan los siguientes métodos:

Entrevista: Proporciona datos importantes acerca de las necesidades de los usuarios en el sistema, además sobre su satisfacción una vez terminado el producto.

Sistematización de experiencias: Posibilitó la obtención de valoraciones empíricas de la factibilidad en la aplicación de esta herramienta, para la gestión de acuerdos y decisiones.

El presente trabajo de diploma está estructurado por los siguientes capítulos:

Capítulo 1: En este capítulo se realiza la **fundamentación teórica** de los sistemas sobre la gestión de la información, relacionada con los acuerdos y decisiones tomados por los directivos. En el mismo se definen cada uno de los aspectos y conceptos relacionados con el sistema a implementar y se realiza un análisis de soluciones similares existentes. Se definen las herramientas y lenguajes de programación necesarios para la implementación de la aplicación. Finalmente, se detalla la elección de la metodología de desarrollo a utilizar, teniendo en cuenta los elementos y fases que la componen.

Capítulo 2: En este capítulo se aborda la **descripción de la propuesta de solución**, en el mismo se concretan las características generales del sistema, se procede a realizar la captura de requisitos funcionales y no funcionales a los que se debe dar cumplimiento. Se lleva a cabo, además, el análisis y diseño del Sistema para la Gestión de Acuerdos, Decisiones e Indicaciones versión móvil “XADIM”, para obtener como principal resultado el diagrama de clases vinculado. Se detalla también la arquitectura del sistema, los patrones utilizados, y se describe el estándar de codificación utilizado.

Capítulo 3: En este capítulo se presenta la verificación y validación de la solución propuesta, aplicando la estrategia de pruebas de caja blanca y caja negra definidas para valorar el funcionamiento del sistema implementado y se validan las variables de la investigación.

Capítulo 1. Fundamentación teórica

1.1 Introducción.

En el presente capítulo se esboza el marco teórico, abordando los principales conceptos y herramientas usadas en la investigación. Se realiza una breve descripción de los sistemas existentes relacionados a la investigación, ya sea en el escenario nacional como en el ámbito internacional para luego realizar una valoración de los mismos. Se elabora un estudio general de características de las metodologías ágiles para desarrollo de software y uno más específico de la seleccionada. Finalmente quedan definidas la arquitectura, el lenguaje de programación y las herramientas a utilizar para el desarrollo del sistema.

1.2 Conceptos fundamentales.

Acuerdo:

- Solución negociada al conflicto, tarea a realizar por parte de uno o varios miembros de una organización (8).
- Reunión de una autoridad gubernamental con sus inmediatos colaboradores para tomar conjuntamente decisiones sobre asuntos determinados. Resolución tomada por una o por varias personas (9).
- Según Franco Aguilar, un acuerdo es “Resolución que se toma en los tribunales, sociedades, comunidades u otros órganos” (10).

Por lo que, teniendo en cuenta los conceptos antes expuestos por varios autores, para la presente investigación se toma ***acuerdo*** como una decisión, tomada sobre algo, que debe ser llevada a cabo por alguien en un tiempo determinado; véase también como una decisión, indicación o tarea asignada.

Decisión: Resolución o determinación acerca de algo dudoso (11). Una Decisión es una respuesta con voluntad con la que se resuelve un conflicto o se determina el destino de una cosa o situación (12).

Indicación: Comunicación o explicación mediante indicios y señales. Cosa con la que se indica algo (11).

Sistema: Conjunto de elementos relacionados entre sí y que funcionan como un todo (13).

Un ***sistema informático*** es como cualquier sistema, un conjunto de funciones interrelacionadas, de hardware, software y personal informático. Estos emplean un sistema que utiliza dispositivos que permiten programar, almacenar y procesar programas, datos e información. (14)

Disponibilidad: Es la probabilidad de que un sistema esté en disposición de funcionar para proporcionar los servicios a los usuarios que lo soliciten. (15)

Visibilidad: Capacidad de que un sistema pueda ser observado correctamente y de manera agradable a la vista de los usuarios. (15)

Aplicación: Es un tipo de programa informático (puede ser visto como un sistema informático) diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos. (15)

1.3 Sistemas existentes relacionados con la investigación.

El resultado de toda investigación se basa en los estudios realizados sobre el problema planteado. Es por ello que a continuación se presentan algunos ejemplos de sistemas semejantes a lo que se propone en dicho trabajo.

1.3.1 Escenario Internacional:

A nivel internacional, desde el surgimiento de la tecnología móvil se han desarrollado aplicaciones para dispositivos con el fin de mejorar la gestión empresarial y de la información, en diferentes aspectos. A continuación, se describen ejemplos de dichos sistemas:

- "Diseño e implementación de una aplicación Android para recordatorios". Sistema desarrollado por el Departamento de Informática en la Universidad de Carlos III de Madrid, España (2011) (16). Este proyecto propone desarrollar una aplicación para un dispositivo móvil que facilite al usuario la organización de su tiempo, permitiéndole acceder a su correo electrónico y poder gestionar sus citas mediante eventos dentro de uno de los mayores servicios más utilizados que ofrece Google, que es su aplicación Google Calendar. A estas funcionalidades se le suma una extensión que permite almacenar unos recordatorios en forma de notas a los contactos que posee el usuario dentro de su dispositivo. Como herramientas de desarrollo fueron utilizadas, Eclipse 3.6 con las herramientas de Android en el plug-in ADT rev.11, así como bibliotecas de desarrollo: Java SDK 6, Android SDK rev.11, JavaMail para Android y el API cliente de Google.

Dicho sistema es de gran utilidad ya que le brinda al usuario una ayuda de cómo organizar su tiempo de acuerdo a las tareas que tenga de manera muy sencilla a través de su dispositivo móvil.

- "Diseño e implementación de una aplicación distribuida de gestión de inventario para dispositivos móviles" llevado a cabo por el Departamento de Informática en la Universidad de Carlos III de Madrid, España (2011) (17). El objetivo principal que persigue este proyecto es la gestión del proceso de inventario de una compañía, de manera eficiente, segura, sencilla e intuitiva para los usuarios, ahorrando tiempo y recursos. Asimismo, se desarrolló una aplicación servidor que atienda las peticiones desde el cliente a través de servicios Web, y que acceda a la base de datos para insertar o recuperar la información de inventario de manera transparente para el usuario.

  Este sistema es de gran utilidad debido a que utiliza la tecnología Android para el uso rápido, remoto y sencillo de la gestión de inventario; pero a su vez no coincide con el propósito de la investigación: gestión de la información relacionada con los acuerdos, decisiones e indicaciones.

1.3.2 Escenario Nacional:

Teniendo en cuenta que en el contexto nacional no se ha encontrado ninguna aplicación para la gestión de información que emplee la tecnología móvil, se investigaron otros sistemas que fueron desarrollados con el mismo fin: la gestión de acuerdos, aunque con otras tecnologías.

- Propuesta "SICODI" (18), la cual sugiere un sistema para la gestión de información de reuniones y su empleo en la Unión CubaPetróleo. Dicha propuesta tuvo como objetivos: estudiar el flujo de trabajo para la gestión de las reuniones de los órganos de dirección; realizar el análisis y diseño de un sistema informático que garantizara el acceso, actualización y seguridad de la información de las reuniones de estos órganos de dirección, basados en el flujo de trabajo. Además de implementar el sistema y analizar los resultados de su puesta en explotación. En el desarrollo del mismo se utilizó una arquitectura Cliente-Servidor, se empleó el lenguaje de programación C# y el Gestor de Base de Datos MS SQL Server, para una posible integración, ya que es genérico para las principales aplicaciones cubanas que se emplean en la rama del petróleo.

  La concepción inicial del SICODI se basó en automatizar los flujos de trabajo del Consejo de Dirección y del Consejo de Directores Generales, lo que facilitó el acceso y el uso de la información referente a estas reuniones, a partir de diferentes roles de usuarios.

- Sistema “SUPERVISA” aplicación comercial desarrollada por la división de Guantánamo de la empresa DESOFT. (18) Supervisa es un sistema basado en plataforma distribuida que permite almacenar, dar seguimiento y controlar los acuerdos que se adopten en las reuniones. Los acuerdos son almacenados según los diferentes tipos de reuniones, y se puede llevar el seguimiento cronológico de un acuerdo hasta que este sea concluido. Es utilizado actualmente en el Buro Central del Comité del Partido. Este sistema no es web y sólo registra los datos, por lo que no existe un flujo de datos directo.
- “Sistema de Gestión de Decisiones para la Fiscalía General de la República” (SGD). Esta aplicación fue realizada en la Universidad de las Ciencias Informáticas, específicamente para la institución: Fiscalía General de la República. Este fue realizado utilizando las siguientes herramientas: Symfony 2.3 como marco de trabajo, Doctrine 2.0 para el mapeo objeto-relacional (ORM), PHP 5.3.10, HTML 5 y CSS 3 como lenguaje, Twitter Bootstrap 3.0, jQuery, 1.10.2 para las interfaces, Apache 2.0 como servidor web, y PostgreSQL 9.1 como Sistema Gestor de Base de Datos.

  Dicho sistema brinda su servicio desde un servidor web, que puede ser accedido desde cualquier dispositivo que esté conectado a la red, lo que hace que sea imposible la consulta de dichas decisiones en caso de no estar conectado. Además, está personalizado a las características de las decisiones de los directivos de la fiscalía.

1.3.3 Valoración de las soluciones existentes:

Después de estudiar y analizar las aplicaciones existentes se llega a la conclusión de que no se satisfacen las demandas que genera la situación problemática que da origen a la presente investigación. Además, las funcionalidades implementadas en dichos sistemas varían en dependencia del problema dado en cada caso. Se estudian las características a fines que tienen con el problema planteado, tomándose como referencia para apoyar la realización de la solución.

Como parte del estudio de las soluciones nacionales existentes se toma el proyecto (SGD), como punto de partida para la actual investigación, teniendo en cuenta sus puntos en común con el problema abordado. Con dicho propósito se crea el Sistema para la Gestión de Acuerdos y Decisiones (XADI), cuyo resultado se integra con el nuevo sistema que se propone en el presente trabajo de diploma: “Sistema para la Gestión de Acuerdos y Decisiones versión móvil” (XADIM).

1.4 Metodología de desarrollo.

En la actualidad no está definida una metodología genérica para desarrollar en cualquier proyecto de desarrollo de software, sino que estas tienen prácticas específicas que las hacen adaptables o no, dependiendo de las características de cada proyecto. La metodología se basa en una filosofía, distinguiéndose de los métodos, ya que estas marcan unos pasos a seguir. Entre sí, las metodologías difieren por la cantidad de fases, las técnicas de cada fase, el contenido de la fase o en su base filosófica; todo esto se aplica, dependiendo del contexto de desarrollo, tamaño del proyecto o del equipo de trabajo, cultura organizacional, entre otros aspectos. (19)

En todo proceso de desarrollo de software se debe elegir la metodología de desarrollo que mejor se adapte a la necesidad del producto final y permita entregar una solución de calidad. En la actualidad existen varias metodologías ampliamente usadas en la industria, pero al momento de elegir la adecuada para una solución móvil se debe tener en cuenta que los escenarios y los requerimientos cambian con respecto al desarrollo de software tradicional. Esto se debe a que el software móvil debe poseer condiciones especiales tales como:

- **Canal radio**: La disponibilidad, desconexiones, la variabilidad del ancho de banda, la heterogeneidad de redes o los riesgos de seguridad.
- **Movilidad**: La migración de direcciones, alta latencia[2] debido a cambio de estación base o la gestión de la información dependiente de localización.
- **Portabilidad**: Implica una serie de limitaciones físicas directamente relacionadas con el factor de forma de los mismos, como el tamaño de las pantallas (algo que ha variado sustancialmente con la popularización de las pantallas táctiles), o del teclado, limitando también el número de teclas y su disposición.

Bajo este escenario, la industria de desarrollo móvil ha adoptado las metodologías ágiles como las apropiadas para ser aplicadas (19) (20) (21), esto debido a que las propiedades de la metodología ágil se alinean a las necesidades de la construcción de software móvil. Además, estas metodologías tienen

[2] Latencia: retraso entre el momento que se inicia el proceso y el momento en que uno de sus efectos comienza.

como punto fuerte la adaptación a cualquier cambio en un proyecto para aumentar sus posibilidades de éxito. (22) Utilizando como principios:

- Los individuos y sus interacciones son más importantes que los procesos y las herramientas.
- El software que funciona es más importante que la documentación exhaustiva.
- Colaboración con el cliente en lugar de negociación de contratos.
- No hay que seguir un plan cerrado, sino adaptarse al cambio. (19)

Tomando como guía los principios anteriormente expuestos queda seleccionada el uso de un enfoque ágil para el desarrollo de XADIM, ya que el equipo de trabajo y el producto son pequeños, de pocos desarrolladores, los cuales están bien unidos y colaborativos. Además, los requerimientos están bien definidos y poseen una arquitectura diseñada específicamente para estos; enfocándose en las personas y los resultados. Cumpliendo de esta forma con las características que rigen las metodologías antes mencionadas.

Metodología seleccionada:

Se determina como metodología ágil a utilizar Scrum, de acuerdo a las características del software a desarrollar. Scrum no se trata únicamente de un método para desarrollo de software, sino que puede ser aplicado teóricamente a cualquier contexto en donde un grupo de personas (equipo de trabajo) necesitan trabajar juntas para lograr una meta común.

Más que una metodología de desarrollo, es una herramienta para gestionar proyectos. Está basada en el hecho de que el trabajo es realizado por equipos auto-organizados y auto-dirigidos, logrando motivación, responsabilidad y compromiso. Es un proceso constructivo, iterativo e incremental donde las iteraciones tienen una duración fija. Scrum, al ser una metodología de desarrollo ágil tiene como base la idea de creación de ciclos breves para el desarrollo, conocidos como sprints. Además, la metodología contiene definición de roles, prácticas y productos de trabajo escritas de forma simple y soportada en un conjunto de valores y principios. (23)

Elementos de Scrum:

- Roles:
  - Propietario del producto.
  - Líder del proyecto.

- Equipo de desarrollo.
- Interesados en el producto.

❖ Pila del producto: Es el inventario en el que se almacenan todas las funcionalidades o requisitos en forma de lista priorizada de las necesidades del cliente. Estos requisitos serán los que tendrá el producto o los que irá adquiriendo en sucesivas iteraciones, se suelen usar historias de usuarios para expresarlos.

  - Se describen las historias de usuarios.

❖ Pila de sprint: Es la lista de tareas que debe realizar el equipo durante el sprint para generar el incremento previsto. Este es una forma de registrar y organizar el trabajo pendiente para el producto (actividades y requerimientos). Es un documento dinámico que incorpora constantemente las necesidades del sistema. Por lo tanto, nunca llega a ser una lista completa y definitiva. Se mantiene durante todo el ciclo de vida y es responsabilidad del propietario del producto.

❖ *Sprint*[3]: Constituye el núcleo de Scrum, que divide de esta forma el desarrollo de un proyecto en un conjunto de pequeñas "carreras".

Los elementos antes mencionados, en conjunto con el **incremento,** conformarían los **artefactos** generados por la metodología Scrum, donde el incremento es la parte de producto realizada en un sprint, y tiene como característica el estar completamente terminada y operativa, en condiciones de ser entregada al cliente. (24)

Dentro del marco teórico, otro elemento importante a definir, es la arquitectura del sistema operativo.

1.5 Arquitectura de Android.

Android es un sistema operativo considerado como una plataforma emergente para dispositivos móviles que está obteniendo cada vez mayor aceptación. Su núcleo está basado en una versión modificada de Linux v2.6 (25) y su código fuente está disponible para la comunidad de desarrolladores para su libre

[3] Sprint: es el período de tiempo durante el que se desarrolla un incremento de funcionalidad.

estudio y modificación. La empresa desarrolladora de esta plataforma es Google Inc., que con su proyecto *Android Open Source Project* (AOSP) (26) tiene como objetivo evolucionar este sistema proporcionando una plataforma de excelencia para el desarrollo de aplicaciones que mejore las experiencias de usuario de dispositivos móviles. Este proyecto está auspiciado por la *Open Handset Alliance* (27).

La plataforma Android está constituida por una serie de capas software que cooperan entre sí para obtener una funcionalidad específica dando lugar a lo que se denomina software *stack*[4]. Cada una de estas capas utilizan elementos de la capa inferior para realizar sus funciones. La ilustración 2 evidencia la jerarquía de cooperación de las diferentes capas que conforman el software *stack*, siendo las capas superiores dependientes de las inferiores. (28) (29)

Capas o componentes que forman parte de la arquitectura o pila de Android (5): (Ver ilustración 2)

Núcleo Linux: El núcleo del sistema operativo Android es un kernel Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, solo que adaptado a las características del hardware en el que se ejecutará Android. Proporciona una capa de abstracción para los elementos hardware y el resto de la pila de software. Esto permite que se pueda acceder a esos componentes sin necesidad de conocer el modelo o características precisas de los que están instalados en cada teléfono.

Runtime [5]de Android: Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik.

Bibliotecas: La componen las bibliotecas nativas de Android, que están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono, tarea que normalmente realiza el fabricante.

Marco de trabajo de aplicaciones (Armazón de aplicaciones): Aunque se apoya en las bibliotecas enumeradas anteriormente, no se considera una capa en sí mismo, dado que también está formado por

[4] *Stack:* pila o acumulación de funciones.

[5] *Runtime*: entorno de ejecución de aplicaciones utilizado por el sistema operativo móvil Android.

bibliotecas. En concreto, las bibliotecas esenciales de Android, que incluyen la mayoría de la funcionalidad de las bibliotecas habituales de Java, así como otras específicas de Android. El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android.

Además, las aplicaciones Android se ejecutan cada una en su propia instancia de la máquina virtual Dalvik, evitando así interferencias entre ellas, y tienen acceso a todas las bibliotecas mencionadas antes y, a través de ellas, al hardware y al resto de recursos gestionados por el kernel.

Aplicaciones: Está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones y que se apoyan en las bibliotecas y en el entorno de ejecución ya detallado. En esta capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C o C++) como las administradas (programadas en Java), tanto las que vienen de serie con el dispositivo como las instaladas por el usuario.

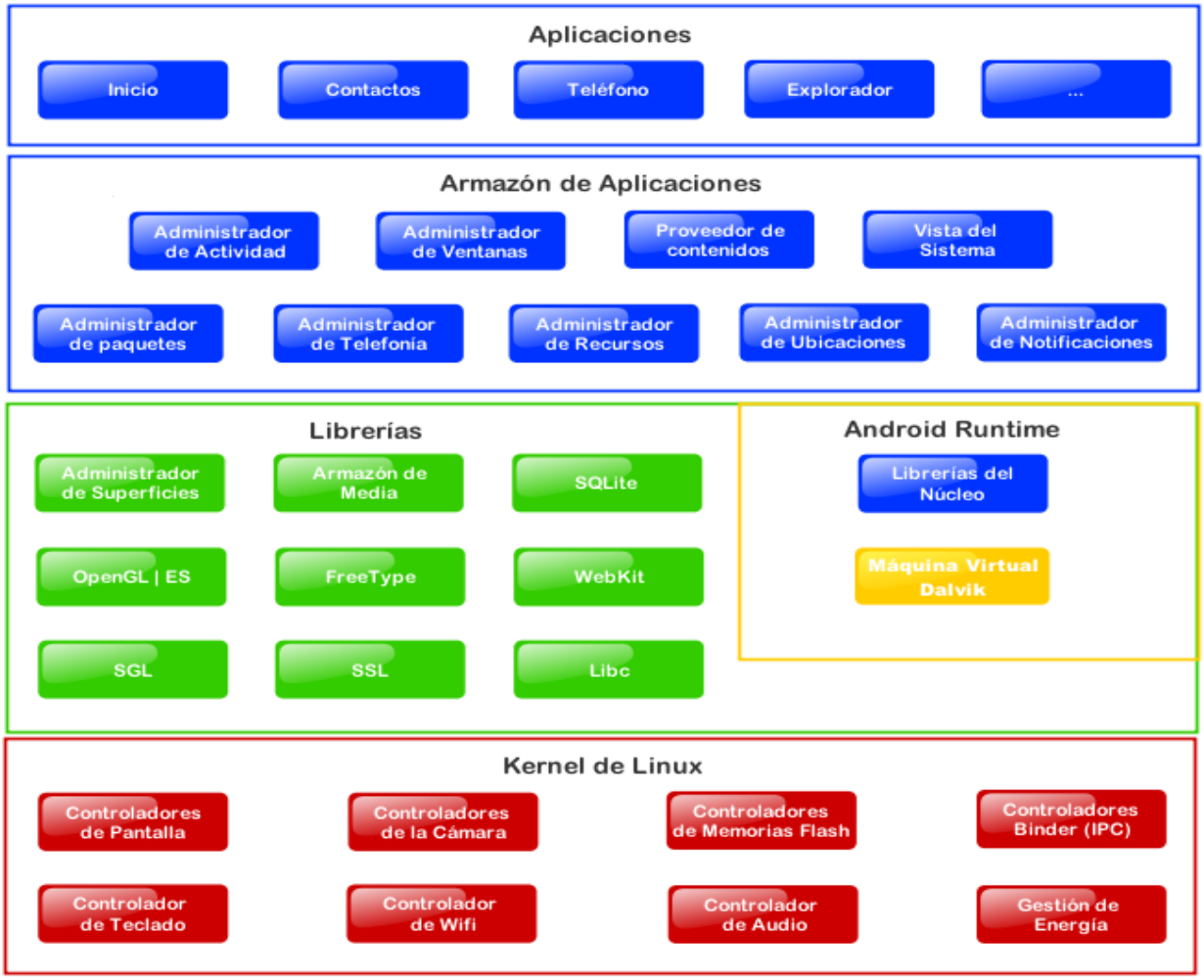


Ilustración 2: Diagrama de los componentes de la arquitectura Android. (5)

Desarrollar aplicaciones sobre la plataforma Android requiere un entorno basado en ciertas tecnologías integradas. Se necesitan las bibliotecas y herramientas de los propietarios, trabajadas sobre los lenguajes definidos a continuación:

1.6 Lenguajes.

Para el desarrollo de aplicaciones Android es preciso el conocimiento de los lenguajes necesarios a utilizar durante la implementación de las aplicaciones realizadas sobre esta plataforma. Dichos lenguajes se describen a continuación:

1.6.1 Lenguaje de modelado UML.

Lenguaje Unificado de Modelado (UML por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo. Es uno de los lenguajes de modelado de sistemas de software más conocido y utilizado en la actualidad respaldado por el Grupo de Administración de Objetos (OMG, por sus siglas en inglés, *Object Management Group*). Este permite la modelación de sistemas con tecnología orientada a objetos fundamentalmente mediante el uso de los diagramas. (30).

1.6.2 Lenguaje Marcado Extendido (*XML*, por sus siglas en inglés).

Es un metalenguaje extensible de etiquetas utilizado para el diseño de interfaces a través de *Parsing*[6], el cual permite definir la gramática de lenguajes específicos. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, principalmente se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. Entre sus principales ventajas se pueden mencionar (31):

- Es extensible: una vez diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas.
- El analizador: es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML posibilitando el empleo de cualquiera de los analizadores disponibles.
- Transforma datos en información, pues se le añade un significado concreto y los asocia a un contexto, con lo cual se tiene flexibilidad para estructurar documentos.

1.6.3 Lenguajes de Programación.

Un lenguaje de programación es un lenguaje utilizado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un conjunto de

[6] *Parsing:* analización gramática del código.

instrucciones, operadores y reglas de sintaxis que pone a disposición del programador para que este pueda comunicarse con los dispositivos de hardware y software del sistema. (32)

Se utiliza como lenguaje de programación Java para construir las aplicaciones. Java es un lenguaje orientado a objetos que permite representar los conceptos de la vida real como entidades de programación. También posee otras características importantes**:**

- Es un lenguaje que genera ficheros de clases, las que son en realidad interpretadas por la máquina virtual de Java. Siendo esta la que mantiene el control sobre las clases que se estén ejecutando.
- Es un lenguaje multiplataforma: el mismo código escrito en Java que funciona en un sistema operativo, funcionará en cualquier otro que tenga instalada la máquina virtual de Java.
- Es un lenguaje seguro: la máquina virtual, al ejecutar el código escrito en Java, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.
- Gracias al API de java se puede ampliar el lenguaje para que sea capaz de, por ejemplo, comunicarse con equipos mediante red, acceder a bases de datos, crear páginas HTML dinámicas, crear aplicaciones visuales.

Teniendo en cuenta las características antes expuestas se decide utilizar UML para la construcción de los artefactos necesarios para el desarrollo de la solución, XML para el diseño de las interfaces y finalmente Java como el lenguaje de programación. Siendo los dos últimos, los lenguajes recomendados para el desarrollo de aplicaciones en Android e integrados perfectamente al IDE Android Studio (33).

1.7 Herramientas de desarrollo.

Para el desarrollo de aplicaciones en Android, su plataforma dispone de una serie de herramientas y recomendaciones para facilitar las implementaciones; estas serán las utilizadas para el desarrollo de la aplicación. Además, se definen otras herramientas para la realización del prototipado y para el modelado del negocio. A continuación, se comentan las diferentes herramientas a emplear:

AndroidStudio 2.2: Es el entorno de desarrollo (IDE) recomendado por Google actualmente para la implementación de aplicaciones Android (34). Se utiliza la versión android-studio-ide-2.2 para Windows.

Java SE JDK 1.8: Es el conjunto de recursos Java, o plataforma, que conforman la denominada *Java Standard Edition*, fundamental para el desarrollo de aplicaciones Java y requerido por el entorno de

desarrollo. Este conjunto de recursos no es propio de la plataforma Android, pero debe ser instalada en el ordenador para poder desarrollar aplicaciones. (35)

Android SDK 24.4.1: Es el conjunto de recursos fundamentales que dispone la plataforma Android imprescindible para el desarrollo de aplicaciones Java. Posteriormente se debe configurar el IDE AndroidStudio para que localice el SDK de Android y disponga de sus recursos en el entorno de desarrollo. Un SDK de Android contiene las bibliotecas, el emulador de dispositivos: para probar los desarrollos sin necesidad de realizar instalaciones sobre un dispositivo físico, documentación, ejemplos de código, tutoriales y una versión de la plataforma. (2)

Gradle 2.10: Es un sistema de compilación basado en JVM (Máquina Virtual de Java, del inglés *Java Virtual Machine*) a su vez es un plugin[7], lo que facilita su actualización y su exportación de un proyecto a otro. Gradle permite reutilizar código fácilmente, hace sencilla la tarea de configurar y personalizar la compilación, permite la distribución sencilla de código al resto del mundo y gestiona las dependencias de forma potente y cómoda ya que está basado en Maven[8]. Esta herramienta emplea al *javac* (Compilador de Java o *Java Compiler*) para programar mediante "*Scripting*[9]" el funcionamiento de la integración modular de la aplicación y permite compilar la aplicación mientras es desarrollada. (2)

Balsamiq Mockups 2.1.13: Herramienta para la creación de los prototipos de interfaz de usuario no funcionales para dispositivos móviles. Balsamiq Mockups es una aplicación para crear maquetas para interfaces gráficas para usuario. Le permite al diseñador diagramar widgets pre construidos utilizando un editor WYSIWYG (lo que ves es lo que obtienes, por sus siglas en inglés). (36)

Visual Paradigm 5.0: Es la herramienta a utilizar para modelado UML, ideal para Ingenieros de Software, Analistas de Sistemas y Arquitectos de sistemas que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos. (37) A través de esta se construyen los artefactos necesarios para llevar a cabo el desarrollo de la solución.

---

[7] *Plugin*: aplicación que añade una funcionalidad o una nueva característica al software.

[8] *Maven*: repositorio de Android.

[9] *Scripting:* secuencias de comandos.

Json: (*JavaScript Object Notation* - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos, es un formato de texto que es completamente independiente del lenguaje. Json está constituido por dos estructuras: una colección de pares de nombre/valor y una lista ordenada de valores. (38)

1.8 Conclusiones parciales.

Del estudio de los sistemas homólogos existentes se concluye que ninguno responde a las exigencias que deben ser tenidas en cuenta para dar respuesta al problema planteado, tomando como punto de partida para la actual investigación a XADI que gestiona las decisiones, pero se le identifican problemas de visibilidad y disponibilidad.

Teniendo en cuenta la dimensión de la problemática planteada, que solo hay un desarrollador, que los requerimientos están bien definidos y que es la recomendada por la empresa Google para el desarrollo de aplicaciones para Android, se selecciona la metodología ágil de desarrollo de software Scrum.

Capítulo 2. Características del sistema

2.1 Introducción.

En el siguiente capítulo se especifica la solución propuesta para resolver la problemática planteada. Se describirá el proceso sobre el cual se soportará el sistema, haciendo uso de las metodologías, arquitecturas, lenguajes de programación y herramientas seleccionadas. Se guiará el desarrollo del software a través de los requisitos funcionales y no funcionales, así como mediante sus prototipos de interfaz de usuarios. Además de realizar el diseño de la arquitectura del sistema y sus estándares de codificación.

Scrum se puede dividir de forma general en 3 **fases**, también tratadas como reuniones, las que se describen a lo largo del presente trabajo.

2.2 Fase I: Planificación de la pila de sprint.

En esta fase se define un documento en el que se reflejarán los requisitos del sistema por prioridades. Además, se puntualiza la planificación del sprint 0, en la que se decidirá cuáles van a ser los objetivos y el trabajo que hay que realizar para esa iteración. Se obtendrá igualmente en esta reunión una pila de Sprint que es la lista de tareas y que es el objetivo más importante del Sprint. (39)

2.2.1 Propuesta del sistema.

Se propone el desarrollo de un sistema móvil que permita consultar la información relacionada con los acuerdos, decisiones e indicaciones tomados por los cuadros administrativos y políticos que ha sido gestionada en el sistema XADI, pero a través de un dispositivo móvil. En dicha aplicación el usuario podrá consultar la información relacionada con las decisiones en cualquier momento o lugar. Además, el usuario puede conocer el estado en el que se encuentran las decisiones vinculadas a él, ya sean decisiones emitidas o asignadas, en término, al límite o fuera de término, así como anular una decisión o consultar los detalles de estas.

2.2.2 Roles del proyecto:

Teniendo en cuenta la metodología seleccionada en el capítulo anterior se definen los siguientes roles.

Tabla 1: Roles que define Scrum.

Roles	
Propietario del producto	Universidad de las Ciencias Informáticas
Equipo de desarrollo	Asiledy Manzanares Rodríguez
Cliente	Ing. Heidy Infante Morales
Interesados	Centro de Gobierno Electrónico de la Facultad 3 (CEGEL)

2.2.3 El Sprint:

El núcleo de Scrum es el Sprint, es un bloque de tiempo de un mes o menos de duración durante el cual se crea un incremento de producto “terminado” utilizable y potencialmente desplegable. Es más conveniente si la duración de los Sprint es consistente a lo largo del esfuerzo de desarrollo. Cada nuevo Sprint comienza inmediatamente después de la finalización del Sprint anterior. (24)

2.2.4 Especificaciones de requisitos del sistema.

Los requisitos son propiedades o restricciones determinadas de forma precisa que deben satisfacerse. Los requisitos de un sistema describen los servicios que ha de ofrecer y las restricciones asociadas a su funcionamiento, se pueden clasificar en: funcionales y no funcionales (40).

A continuación, se especifican los requisitos funcionales y no funcionales sujetos al desarrollo del sistema.

2.2.4.1 Requisitos funcionales.

Los requisitos funcionales se obtienen de los requisitos de XADI, para el desarrollo de una primera versión de la aplicación para dispositivos móviles. A continuación, se muestra, en la tabla 2, la lista de requisitos funcionales asociados cada uno de ellos a su Sprint:

Tabla 2: Pila de Sprint.

# Sprint	Duración (semanas)	Requisito(s) funcional(es) asociado(s) al Sprint
Sprint 1	2	1.Crear una estructura JSON en la aplicación.
Sprint 2	4	2.Autenticar Usuario. 3.Notificar al Usuario la cantidad de notificaciones. 4.Cerrar Sesión .
Sprint 3	4	5.Listar todas las decisiones. 6.Listar todas las decisiones emitidas. 7.Listar todas las decisiones asignadas. 8.Mostrar Notificaciones.
Sprint 4	2	9.Buscar Decisión.
Sprint 5	4	10.Listar decisiones en término. 11.Listar decisiones al límite. 12.Listar decisiones fuera de término.
Sprint 6	1	13.Visualizar detalles de la decisión.
Sprint 7	2	14.Anular decisión.

2.2.4.2 Requisitos no funcionales.

Los requisitos no funcionales se refieren a funciones específicas que proporciona el sistema. Son restricciones de los servicios o funciones ofrecidas por el sistema (fiabilidad, tiempo de respuestas,

capacidad de almacenamiento). Generalmente se aplican a este en su totalidad. Surgen de las necesidades del usuario (restricciones de presupuesto, políticas de la organización, necesidad de interoperabilidad). En muchos casos los requerimientos no funcionales son fundamentales en el éxito del producto. Están vinculados a requisitos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. (40)

A continuación, son especificados los requisitos no funcionales con que debe cumplir el sistema que se propone, agrupados en las siguientes categorías (41) (42):

- Requisitos de usabilidad: Las interfaces gráficas implementadas por el sistema deben concebirse con un ambiente sencillo y de navegación fácil para el usuario, debido a que la aplicación podrá ser usada por usuarios con conocimientos básicos de dispositivos móviles.
- Restricciones de diseño:
  - La apariencia de la interfaz de la aplicación debe poseer los colores rojo, blanco y gris según la estrategia marcaria empleada para el producto XADI.
- Requisitos de sistema (hardware):
  Dispositivo móvil:
  - El dispositivo debe soportar tecnología Wi-Fi 802.11 b/g/n.
  - Se requiere un microprocesador con una velocidad de 650 MHz o superior.
  - Se requiere un mínimo de 256 Mb de memoria RAM.
  - Dispositivo con una resolución igual o superior a 320 x 480 píxeles, para una correcta visualización de la aplicación.
- Requisitos de sistema (software): Se debe disponer en el dispositivo móvil una versión 4.0 o superior del sistema operativo Android.

2.2.4.3 Descripción de Historias de usuarios (HU).

Las historias de usuario corresponden a una de las técnicas utilizadas para especificar los requisitos del software. Se trata de formatos en los cuales el cliente describe brevemente las características que el sistema debe poseer. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. (43)

La estructura de una HU está compuesta por:

Número: Se especifica un identificador para cada HU.

Nombre del requisito: Se especifica el nombre del requisito funcional asociado a la HU.

Nombre del programador: Se especifica el nombre del programador que va a desarrollar el requisito asociado a la HU.

Iteración Asignada: En el caso de la metodología utilizada (SCRUM) se le asigna a cada HU el Sprint al que pertenecen.

Prioridad: Se especifica la prioridad del requisito asociado a la HU que puede ser alta, media o baja en dependencia del negocio.

Tiempo estimado: Se especifica el tiempo que se estima demorará la realización del requisito asociado a la HU.

Riesgo en desarrollo: Se especifican los riesgos que pudiera tener la realización del requisito asociado a la HU.

Tiempo real: Se especifica el tiempo real que demora la realización del requisito asociado a la HU.

Descripción: Se describe con lujo de detalle en que consiste el requisito asociado a la HU.

Observación: Se planta alguna observación que sea de relevancia.

Prototipo elemental de interfaz gráfica de usuario: Se muestra una imagen de cómo quedaría la interfaz gráfica del requisito asociado a la HU.

Se definen un total de 14 historias de usuarios, de las cuales se muestra un ejemplo de una con prioridad alta, el resto de las HU se pueden observar en documento Historias_de_usuario XADIM.

Tabla 3: Historia de usuario Visualizar detalles de la decisión.

Número: 10	**Nombre del requisito:** Visualizar detalles de la decisión
Programador: Asiledy Manzanares Rodríguez	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 56h
Riesgo en Desarrollo:	**Tiempo Real:**
Descripción: Esta funcionalidad proporciona al usuario los detalles de la decisión seleccionada, tales como su número, la fecha de vencimiento, el emisor, si es priorizada o no, así como su procedencia interna y externa, además se muestra el asunto de la decisión.	
Observaciones: Botones de la interfaz: ❖ **Cerrar:** Retorna a la interfaz anterior del sistema.	
Prototipo elemental de interfaz gráfica de usuario:	

2.3 Fase II: Seguimiento del Sprint.

En esta fase se hacen reuniones diarias en las que las tres preguntas principales para evaluar el avance de las tareas serán (39):

- Que trabajo se realizó desde la reunión anterior.
- Que trabajo se hará hasta una nueva reunión.
- Inconvenientes que han surgido y que hay que solucionar para poder continuar.

2.3.1 Técnicas de validación de requisitos.

Como técnicas de validación de requisitos adoptadas en este trabajo se tienen:

Casos de prueba

Los casos de prueba son un conjunto de condiciones o variables bajo las cuáles un analista determinará si una aplicación o sistema software es parcial o completamente satisfactoria. Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber al menos un caso de

prueba para cada requisito a menos que un requisito tenga requisitos secundarios. En ese caso, cada requisito secundario deberá tener por lo menos un caso de prueba. Lo que caracteriza un escrito formal de caso de prueba es que hay una entrada conocida y una salida esperada, los cuales son formulados antes de que se ejecute la prueba. La entrada conocida debe probar una precondición y la salida esperada debe probar una postcondición (40).

Donde se establecieron un conjunto de decisiones “fuera de término”, “al límite” y “en término”, de las que el usuario puede obtener sus características y anularlas. También se verifica la ejecución del buscador de decisiones y el gestor de notificaciones. Determinándose la implementación de 14 casos de prueba correspondientes a las 14 HU.

Prototipado

El prototipado de interfaz de usuario es una técnica de representación aproximada de la interfaz de usuario de un sistema software que permite a clientes y usuarios entender más fácilmente la propuesta de los ingenieros de requisitos para resolver sus problemas de negocio. Los dos tipos principales de prototipos de interfaz de usuario son (44):

- Desechables: que se utilizan sólo para la validación de los requisitos y posteriormente se desechan.
- Evolutivos: una vez utilizados para la validación de los requisitos, se mejora su calidad y se convierten progresivamente en el producto final.

Se realizaron un total de 14 prototipos no funcionales para cada HU. Para ello se utilizó la herramienta Balsamiq Mockups. Esto permitió una representación inicial a los usuarios finales de las interfaces del sistema.

2.3.2 Diseño de la Arquitectura de Software.

Como parte del diseño de la solución se abordan algunos elementos tenidos en cuenta para la descripción y organización en la fase de implementación.

2.3.3 Patrón arquitectónico.

Una meta del diseño del software es obtener una aproximación arquitectónica de un sistema. Esta sirve como estructura a partir de la cual se realizan las actividades de diseño más detalladas. Un conjunto de

patrones arquitectónicos permite que el ingeniero de software resuelva problemas de diseño comunes. (40)

Teniendo en cuenta el creciente uso de la programación orientada a objeto en la concepción e implementación de este tipo de aplicaciones y la gran actualidad que tiene el uso de patrones internacionalmente aceptados, se propone un análisis del patrón Modelo Vista Presentador (MVP).

MVP es un patrón de arquitectura de software cuya finalidad es la de separar la lógica de negocio de la Vista y facilitar el proceso de prueba del código. El MVP es una derivación del patrón arquitectónico modelo–vista–controlador (MVC) y es utilizado mayoritariamente para construir interfaces de usuario. Aunque ambos modelos parecen bastante similares ofrecen importantes diferencias (45):

1. En el MVC, el modelo notifica a la vista cualquier cambio que sufra el estado del modelo. La información puede pasarse en la propia notificación, o después de la notificación, la vista puede consultar el modelo directamente para obtener los datos actualizados. Por el contrario, en el MVP, la vista conoce sobre el modelo y la función del presentador es la de mediar entre ambos, enlazando los datos con la vista.
2. En el modelo MVC, la vista tiende a tener más lógica porque es responsable de manejar las notificaciones del modelo y de procesar los datos. En el modelo MVP, esa lógica se encuentra en el presentador, haciendo a la vista "simple". Debido a que su única función es representar la información que el presentador le ha proporcionado.
3. En MVC, el modelo tiene lógica extra para interactuar con la vista. Mientras que, en el MVP, esta lógica se encontraría en el presentador.

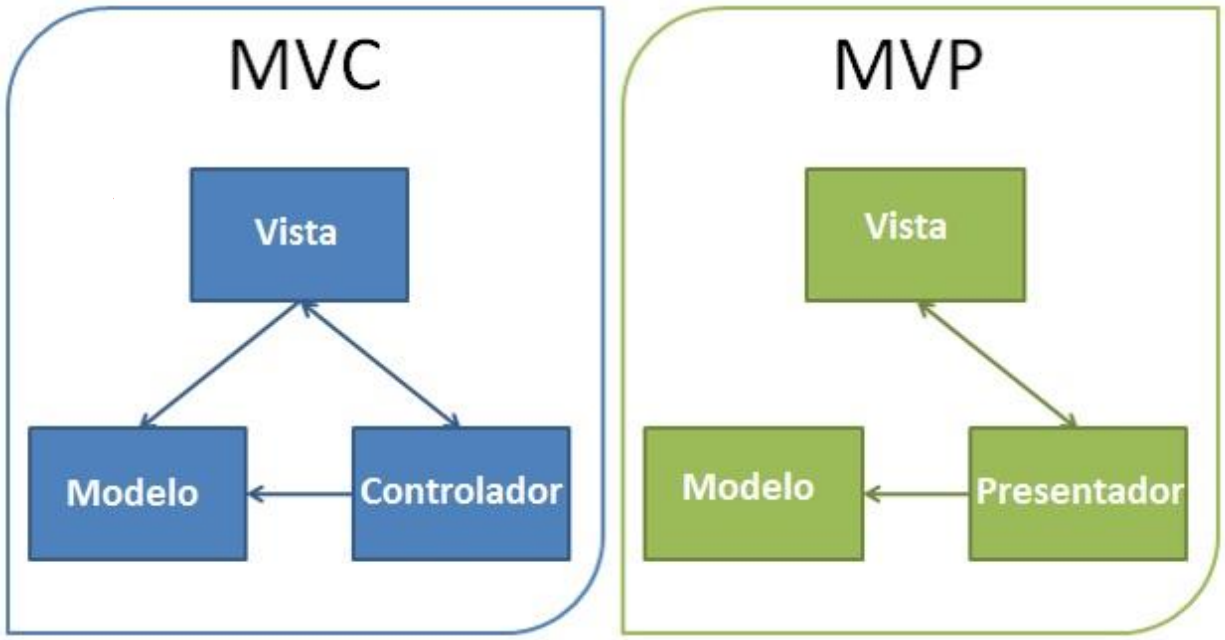


Ilustración 3: Patrones arquitectónicos: Modelo vista controlador y Modelo vista presentador.

Al usar MVP, se consigue que las Activities[10] y Fragments[11] queden completamente desacoplados de la parte de negocio, de tal forma que pueden ser reutilizadas con gran facilidad, además de dotar de una mayor legibilidad al código, ya que éste queda dividido y, al conocer las responsabilidades de cada clase, se conoce en qué parte diferenciada va a estar aquello que hace lo que se desee buscar.

Capas del patrón MVP (46):

- **Vista**: Suelen ser los Fragments o Activities correspondientes, quienes implementan una interfaz con todos los métodos que permiten modificar y rellenar con datos los layouts[12], que fueron definidos haciendo uso de los métodos correspondientes que dota Android.

[10] Activities: clases que expresan las actividades que se realizan en una aplicación.

[11] Fragments: fragmentos de clases que se incluyen dentro de una activity.

[12] Layouts: clases que definen el diseño de la interfaz en XML, manejada por una clase java.

- **Presentador**: Es una clase que se encarga de modificar la Vista, por lo que tiene una referencia a su interfaz desde donde lo hace. De esta forma, da igual la representación de esa Vista, el Presentador solo se encarga de decidir qué se muestra y no cómo se muestra.
- **Modelo**: Su responsabilidad es la de proporcionar, desde clases externas, los datos que necesita el Presentador y dotarle de métodos para poder modificarlos.

El uso de esta arquitectura mejora en gran medida la calidad del código, brindando las siguientes ventajas:

- Este queda dividido, según sus responsabilidades, en diferentes clases; fraccionadas, a su vez, en métodos que se encargan de acciones únicas. De esta forma, encontrar "qué hace qué se convierte en una tarea mucho más sencilla. Además, se hace un estudio de los artefactos de XADI para elaborar el diagrama de clases de XADIM.
- Facilita los cambios, tanto a nivel funcional como a nivel de diseño, gracias a su desacoplamiento, ya que sólo es necesario crear una Vista que implemente la interfaz para cambiarla por completo o cambiar la parte de la lógica en el Presentador, sin tener que cambiar nada de la misma.
- Permite la reutilización de código. Creando una vista que implemente la interfaz, se puede reutilizar el Presentador; e implementando un nuevo Presentador que haga uso de la interfaz, además de poder reutilizar la Vista.

2.3.4 Diagrama de Paquetes.

Para un mejor entendimiento de cómo este patrón se refleja en el desarrollo de la aplicación se expone en la Ilustración 4 un diagrama de paquetes, donde se expone una vista detallada de los paquetes Modelo, Vista, Controlador y los componentes que contiene cada uno en particular.

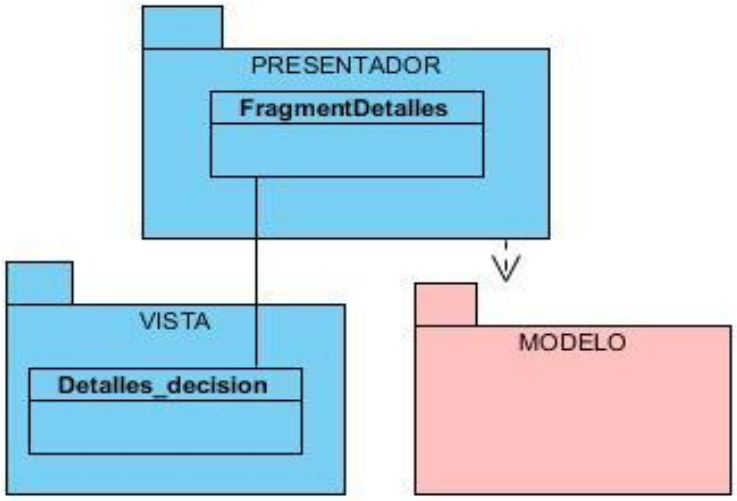


Ilustración 4: Diagrama de paquetes de la HU Detalles de la decisión.

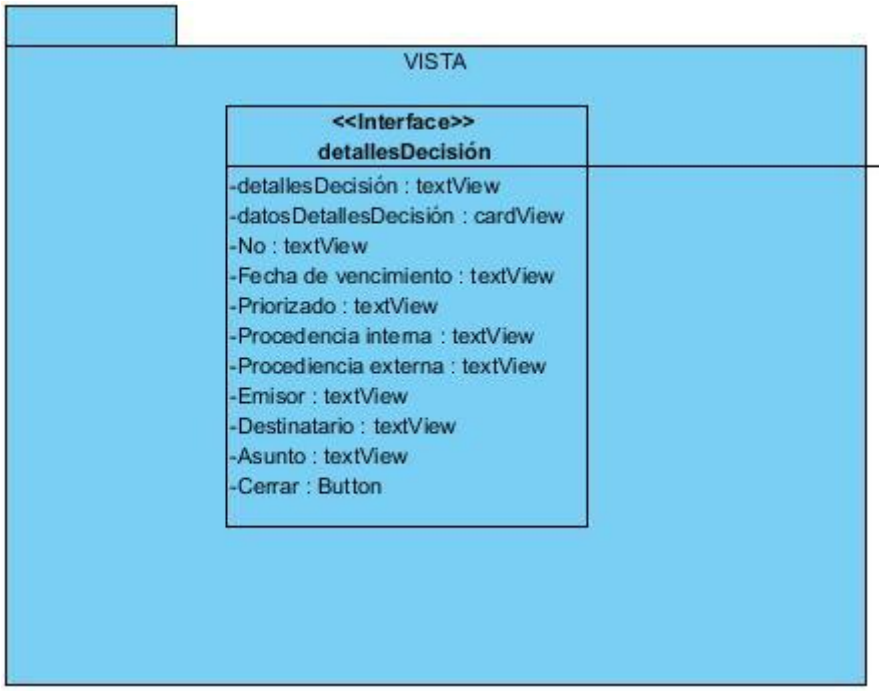


Ilustración 5: Contenido correspondiente al paquete Vista.

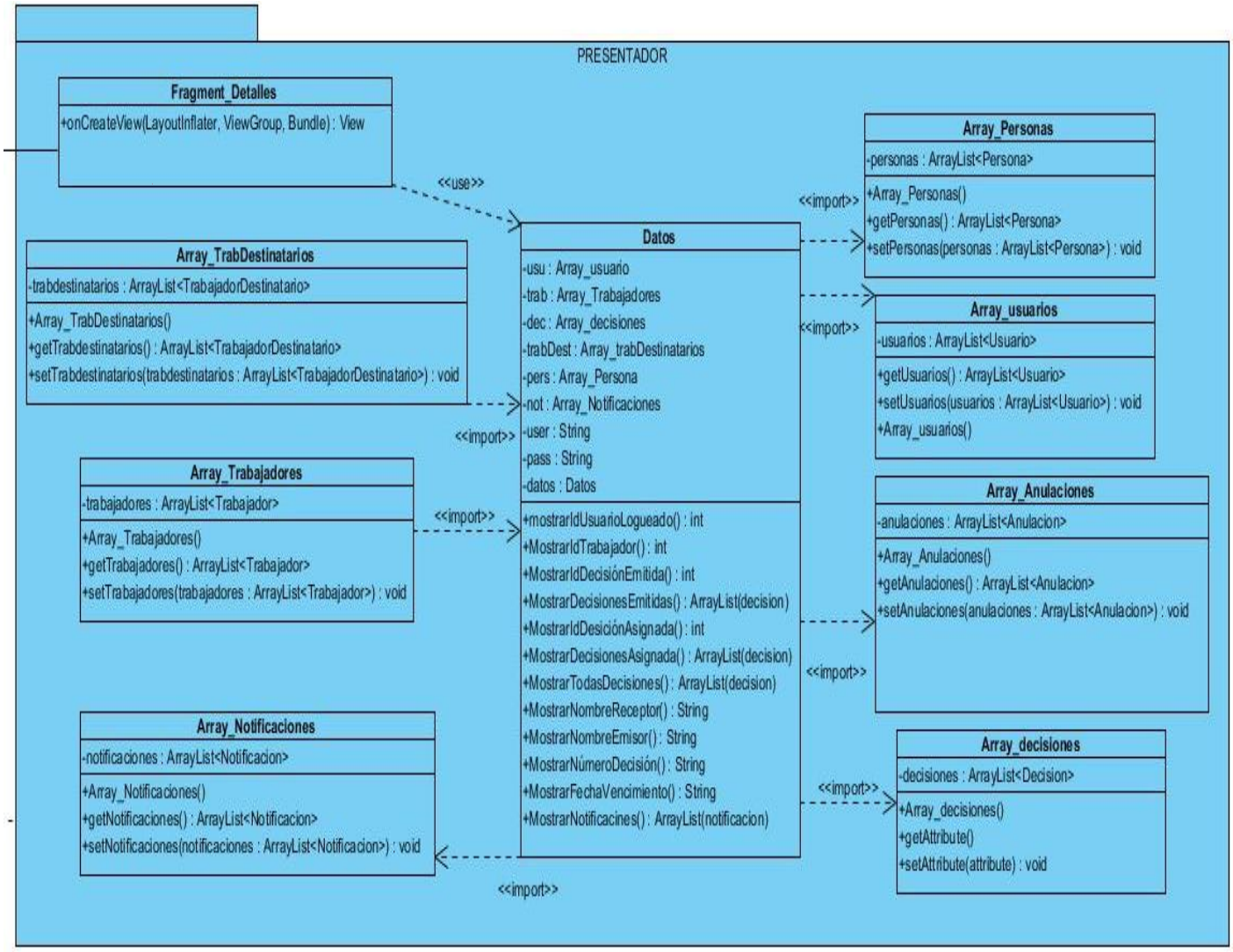


Ilustración 6: Contenido correspondiente al paquete Presentador.

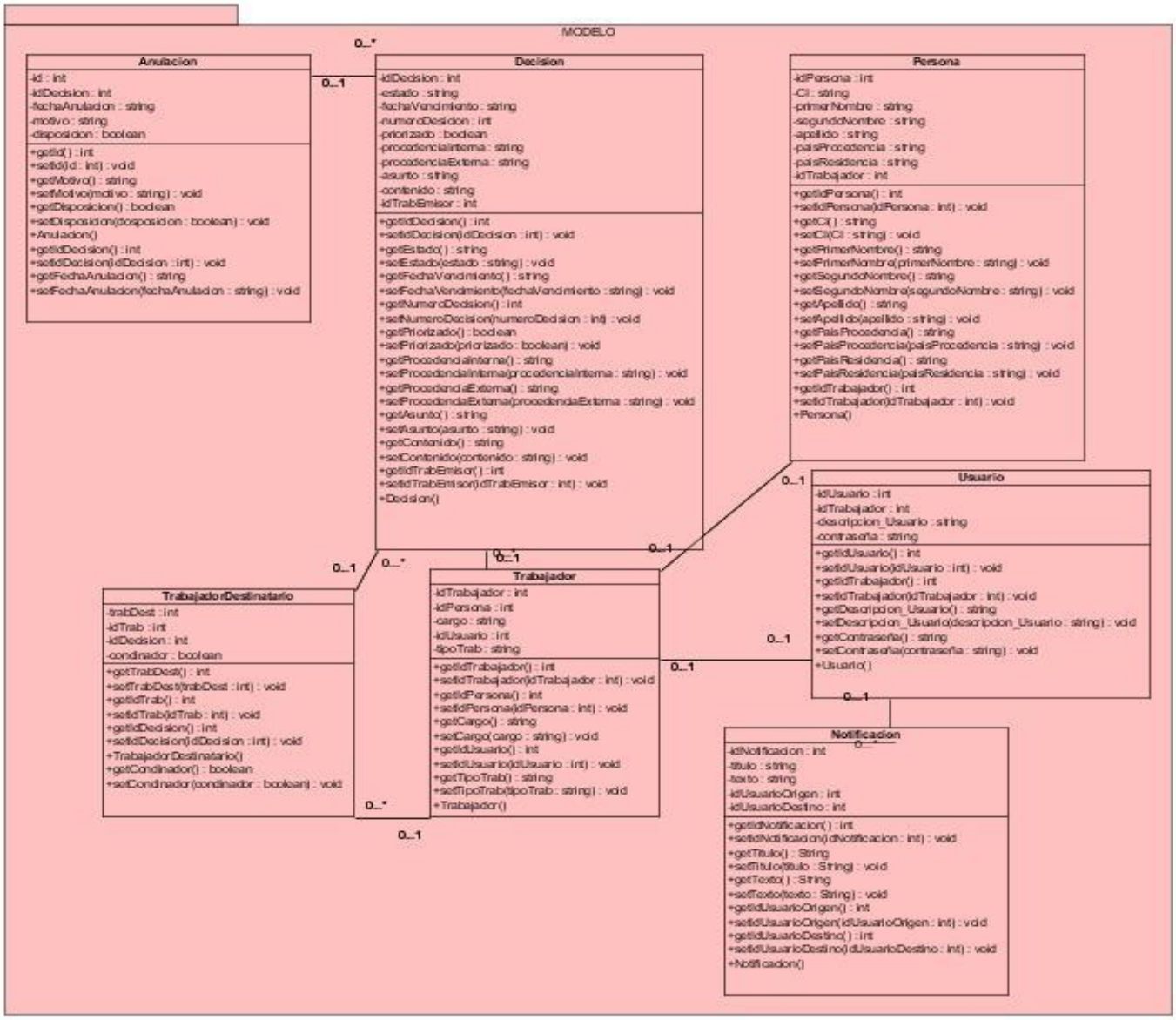


Ilustración 7: Contenido correspondiente al paquete Modelo.

2.3.5 Diagrama de clases persistentes.

Las clases persistentes son, en una aplicación, las clases que implementan las entidades del problema empresarial. La persistencia de una clase está definida por la propiedad de los objetos de trascender su estado en el tiempo y el espacio: una clase persistente existirá durante la ejecución de un programa, y deberá sobrevivir incluso a la eliminación o colapso del mismo. (47) A continuación se representa el diagrama de clases persistentes de la solución:

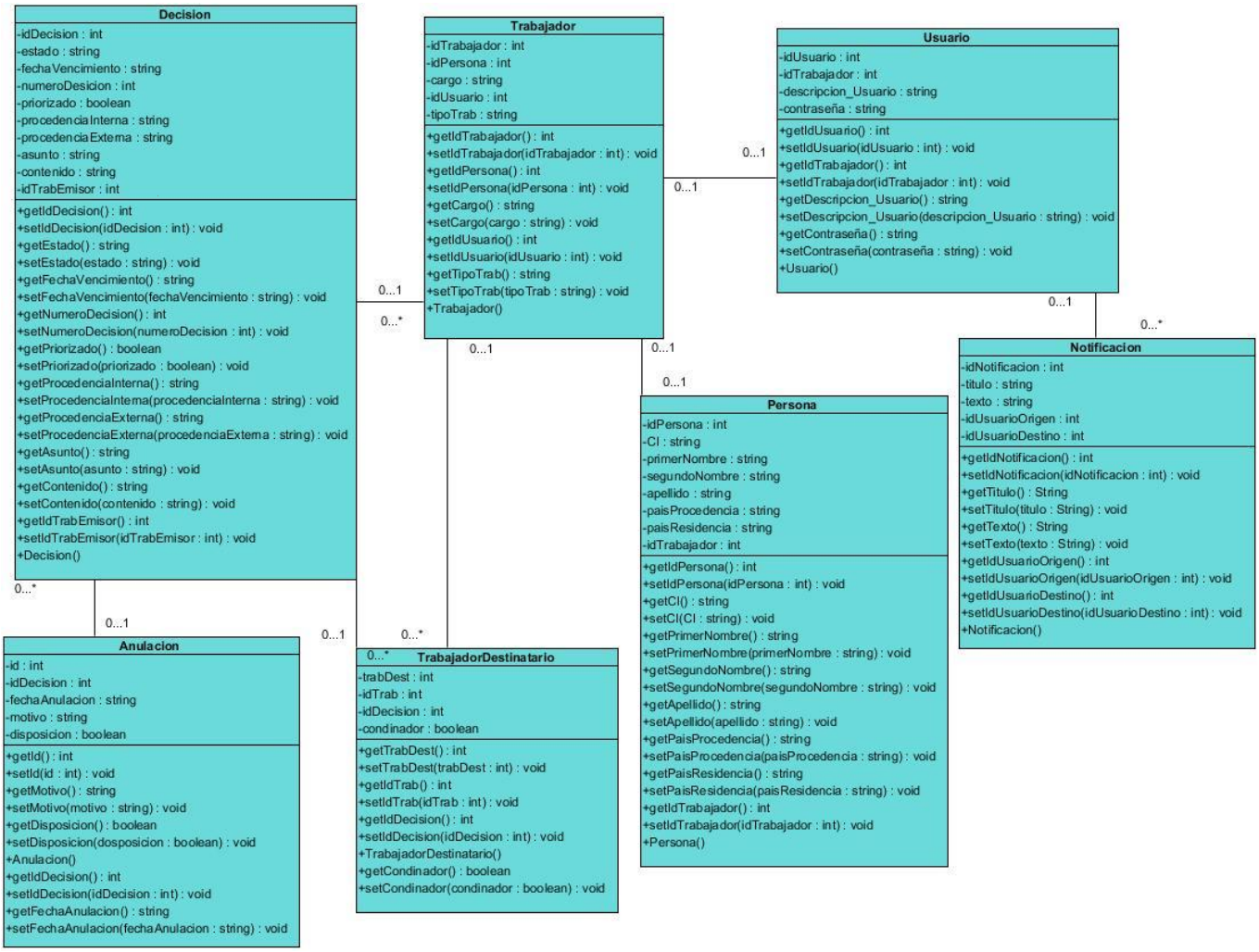


Ilustración 8: Diagrama de clases persistentes.

2.3.6 Patrones de diseño.

Los patrones de diseño son descripciones de clases y objetos relacionados que están particularizados para resolver un problema de diseño general en un determinado contexto. Un patrón de diseño nomina, abstrae e identifica los aspectos clave de una estructura de diseño común, lo que los hace útiles para crear un diseño orientado a objetos reutilizable. El patrón de diseño identifica las clases e instancias participantes, sus roles y colaboraciones, y la distribución de responsabilidades. (48) Concluyendo, los patrones de diseño son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de software. (49)

Una de los aspectos de complejidad en la Programación Orientación a Objeto consiste en elegir las clases adecuadas y decidir cómo estas clases deben interactuar. Aún en las metodologías ágiles, es inevitable elegir cuidadosamente las responsabilidades de cada clase en la primera codificación y, fundamentalmente, en la refactorización (continua) del programa. (50) es por ello que se utilizan como buena práctica los patrones de diseño GRASP.

Patrones GRASP: (Patrones de Software de Asignación de Responsabilidad General, del inglés *General Responsibility Assignment Software Patterns*) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en formas de patrones. A continuación se explican los principales patrones utilizados. (50) (51)

- Experto: Asignar una responsabilidad a una clase de manera que esta posea o pueda tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.

  Este patrón se evidencia en la clase decisión, la que contiene toda la información necesaria sobre las decisiones, otras clases donde se evidencia este patrón son las correspondientes al Modelo: Anulacion.java, Decision.java, Notificacion.java, Persona.java, Trabajador.java, TrabajadorDestinatario.java y Usuario.java, donde ellas tienen todos los datos de los métodos que realizan.

```
public class Decision {

    @SerializedName("idDecision")
    private int idDecision;

    @SerializedName("estado")
    private String estado;

    @SerializedName("fechaVencim")
    private String fechaVencim;

    @SerializedName("noDecision")
    private int noDecision;

    @SerializedName("priorizado")
    private boolean priorizado;

    @SerializedName("procedenciaInterna")
    private String procedenciaInterna;

    @SerializedName("procedenciaExterna")
    private String procedenciaExterna;

    @SerializedName("asunto")
    private String asunto;

    @SerializedName("contenido")
    private String contenido;

    @SerializedName("idTrabEmisor")
    private int idTrabEmisor;
}
```

Ilustración 9: Clase Decisión del paquete Modelo.

- Creador: Asignar a la clase B la responsabilidad de crear una instancia de clase A si se cumple alguno de los puntos siguientes:

1. B contiene a A.

2. B registra a A.

3. B agrega a A.

4. B utiliza estrechamente a A.

5. B tiene los datos de inicialización de A.

Este patrón se evidencia dentro de la clase Array_decisiones.java, donde cada instancia de decisión creada se registra en el *array*[13] que las contiene a todas. Otras clases donde se evidencia el empleo del patrón son las correspondientes al paquete Presentador: ArrayAnulaciones.java, Array_Notificaciones.java, Array_Personas.java, Array_Trabajadores.java, Array_TrabDestinatarios.java y Array_usuarios.java.

```
public class Array_decisiones {

    @SerializedName("Decisiones")
    private ArrayList<Decision> decisiones;

    public Array_decisiones(ArrayList<Decision> decisiones) { this.decisiones = decisiones; }

    public ArrayList<Decision> getDecisiones() { return decisiones; }

    public void setDecisiones(ArrayList<Decision> decisiones) { this.decisiones = decisiones; }
}
```

Ilustración 10: Clase Array_decisiones del paquete Presentador.

- Controlador: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión.

  Este patrón se evidencia en la clase principal Datos.java, la cual maneja el flujo de eventos del sistema, implementándose en estas las funcionalidades más críticas del negocio.

[13] Array: colección, lista, serie.

```
public class Datos {
    public static Array_usuarios usu = LoginActivity.usu;
    public static Array_Trabajadores trab = LoginActivity.array_trabajadores;
    public static Array_decisiones dec = LoginActivity.array_decisiones;
    public static Array_TrabDestinatarios trabDest = LoginActivity.array_trabDestinatarios;
    public static Array_Personas pers = LoginActivity.array_personas;
    public static Array_Notificaciones not = LoginActivity.notificaciones;
    public static Array_Anulaciones anul = LoginActivity.anulaciones;
    public static Fragment_Anular FA;
    public static JsonActivity jsonA;
    private String user;
    private String pass;
    private Datos datos;

    public Datos(String user, String pass) {
        this.user = user;
        this.pass = pass;
    }
```

Ilustración 11: Clase Datos del paquete presentador.

- Bajo Acoplamiento: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo, o sea debe existir poca dependencia entre las clases. Al programar o diseñar debe lograrse un acoplamiento lo más bajo posible entre dos unidades de software cualesquiera. Por supuesto, es imposible lograr un desacoplamiento total entre las unidades. Sin embargo, manteniendo lo más bajo posible el acoplamiento se logrará que las distintas "clases" del software funcionen sin depender demasiado unas de otras. Eso redunda en una mejora considerable en la detección y corrección de errores, en una mayor facilidad de mantenimiento y, sobre todo, en la reutilización de esas "clases" en el software (52). El diseño de clases expertas en información brinda soporte a un bajo acoplamiento ya que para responder a una determinada solicitud se accede solo a las funcionalidades de ese experto. Ninguna otra clase podrá responder a la petición mejor que ella, así lo que se hace dependiendo de muchas clases, este experto lo gestiona con su propia información.
- Alta cohesión: Asignar una responsabilidad de manera que la cohesión permanezca alta, o sea, cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. Con este patrón se espera que una clase tenga un número moderado de responsabilidades dentro de un área funcional y colabore con las otras para llevar a cabo una tarea (50). En el caso del diseño propuesto, utilizando el patrón experto se hacen definiciones de clases “sencillas” y más cohesivas que son más fáciles de comprender y de mantener, que contienen las responsabilidades que les corresponden. Se crearon

controladores para cada una de las funcionalidades necesarias y además para las funcionalidades comunes, de esta forma se aumenta la cohesión ya que cada una de ellas se especializa y es coherente con su función específica.

Una especificación formal que permita definir en forma precisa la semántica de patrones de diseño, en particular patrones de Gamma o GoF, puede ser la base para definir una herramienta que ayude a desarrollar software más seguro. (53)

Patrones GOF: (El grupo de los cuatro, del inglés *The Gang of Four*) proponen soluciones a problemas concretos, ya que no son teorías genéricas. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje. Además, se utilizan en situaciones frecuentes, debido a que se basan en la experiencia acumulada al resolver problemas reiterativos.

A continuación, se describe el patrón utilizado en la solución:

Adaptador (del inglés *Adapter*)**:** convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permite que cooperen clases que de otra manera no podrían por tener interfaces incompatibles. Este patrón se utilizó en el diseño e implementación de las clases ListaAdaptador_Buscar.java, Lista_adaptador_A.java, Lista_adaptador_E.java, Lista_adaptador_Notificaciones.java, Lista_adaptador_TD.java, Lista_adaptador_Termino.java, Lista_entrada_datos_adapter.java. La ilustración 11 muestra el ejemplo de la clase Lista_adaptador_TD donde se convierte a adaptador_entrada_td, evidenciándose el uso de este patrón.

Ilustración 12: Clases adaptador_td y adaptador_entrada_td.

Con el aprovechamiento de estos patrones quedan asignadas correctamente las responsabilidades entre clases, para de esta forma obtener un diseño más claro, con alta cohesión, bajo acoplamiento y clases reutilizables que mantienen el encapsulamiento, logrando un código normalizado y estandarizado. Además, la utilización de patrones de diseño, permite ahorrar grandes cantidades de tiempo en la construcción de software, obteniendo un producto más fácil de comprender, mantener y extender.

2.3.7 Validación del diseño propuesto

Con el objetivo de validar el diseño expuesto en el epígrafe anterior se aplican las métricas TOC y RC cuyos resultados se describen a continuación.

2.3.7.1 Métricas orientadas a clases para evaluar el diseño (54).

El diseño propuesto fue validado a partir de las métricas orientadas a clases, concebidas como instrumentos para evaluar la calidad del diseño, definidas por Lorenz y Kidd, Tamaño operacional de las clases (TOC) y las de Chidamber y Kemerer, Relaciones entre clases (RC), que proponen validar el diseño mediante la evaluación de los siguientes atributos de calidad:

- Tamaño operacional de las clases:

Responsabilidad: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.

Complejidad de implementación: Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

- Relación entre clases:

Acoplamiento: Consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy ligada a la característica de Reutilización.

Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta pero fuertemente en los costos y la planificación del proyecto.

Cantidad de pruebas: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto diseñado.

2.3.7.2 Resultados del instrumento de evaluación de la métrica Tamaño operacional de clase (TOC).

Al aplicar la métrica se tuvieron en cuenta las clases Presentadoras, Modelos y las Vistas reflejadas en el diseño de acuerdo a la propuesta planteada, así como la cantidad de procedimientos (métodos) por cada una de ellas.

Para determinar el valor de los atributos, se calcula el promedio de la columna cantidad de procedimientos y este promedio es el que se emplea en la columna criterio (El instrumento se muestra en el Anexo 1).

Resultados en % de los atributos de calidad afectados:

Ilustración 13: Representación del valor en % del atributo responsabilidad.

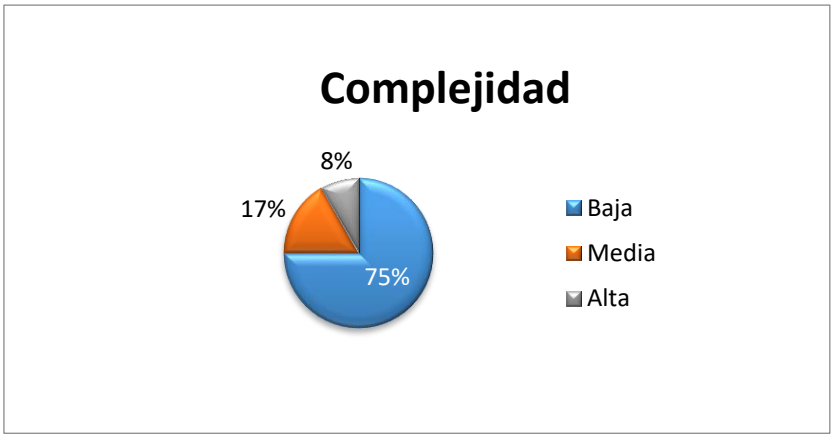


Ilustración 14: Representación del valor en % del atributo complejidad.

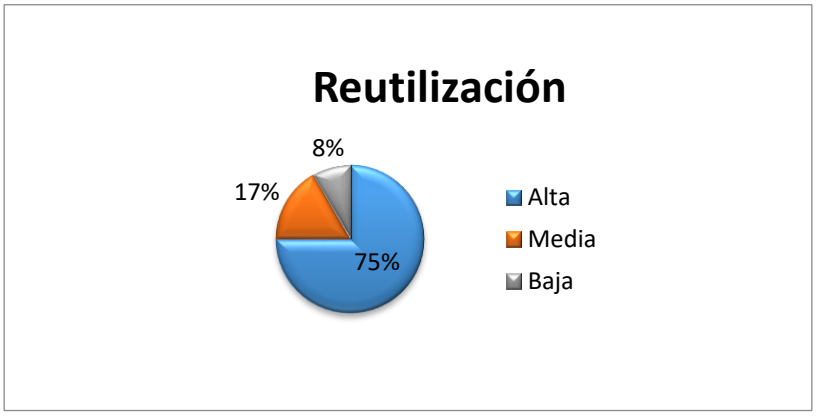


Ilustración 15: Representación del valor en % del atributo reutilización.

Análisis de los resultados obtenidos en la evaluación de la métrica TOC:

Considerando que el 75% de las clases contienen un número de funcionalidades inferior a la media de procedimientos por clases (cuyo valor asciende a 3.305), dando como resultado una elevada reutilización (Ilustración 15), baja responsabilidad y complejidad de implementación en el diseño propuesto (Ilustraciones 14 y 15). Solamente un 8% de las clases tienen pocas posibilidades de reutilización y una gran complejidad de implementación. Estos valores demuestran que los indicadores de reutilización, complejidad y responsabilidad son aceptables.

2.3.7.3 Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC).

La métrica Relaciones entre Clases está dada por el número de relaciones de uso de una clase con otra(s). Se tuvieron en cuenta las clases Presentadoras, Modelos y Vistas reflejadas en el diseño de acuerdo a la propuesta planteada. (El instrumento se muestra en el Anexo 2).

Resultados en % de los atributos de calidad afectados:

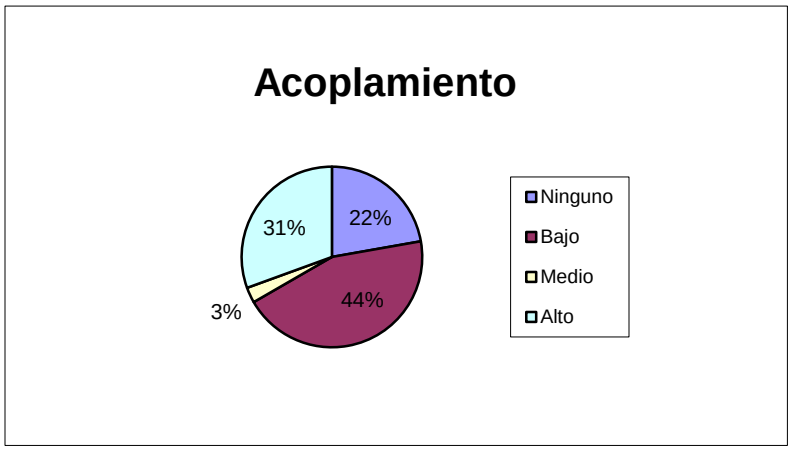


Ilustración 16: Representación del valor en % del atributo Acoplamiento.

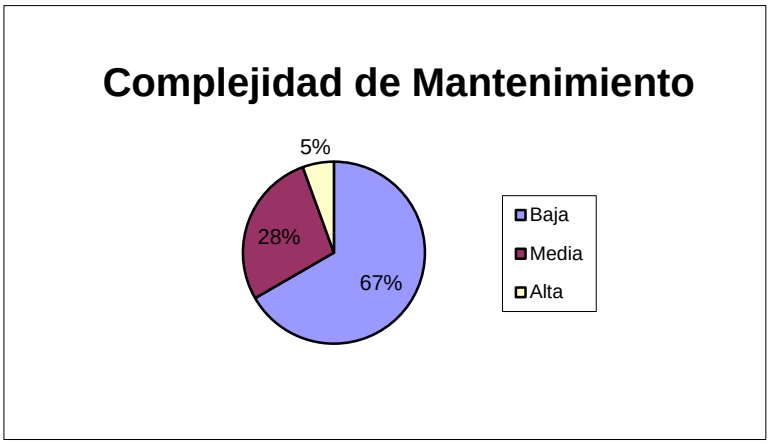


Ilustración 17: Representación del valor en % del atributo Complejidad de mantenimiento.

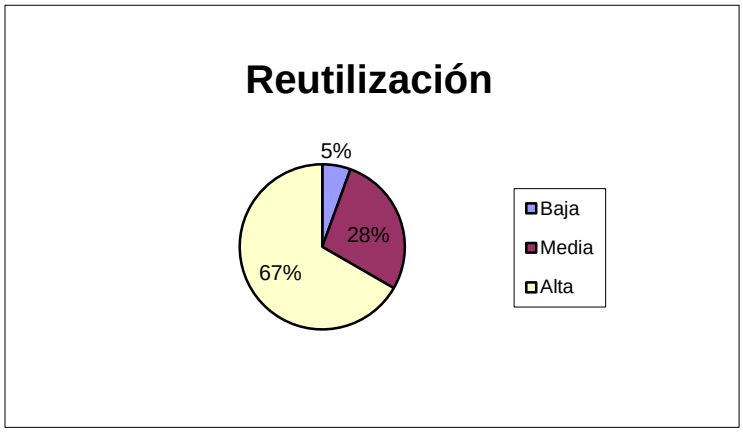


Ilustración 18: Representación del valor en % del atributo Reutilización.

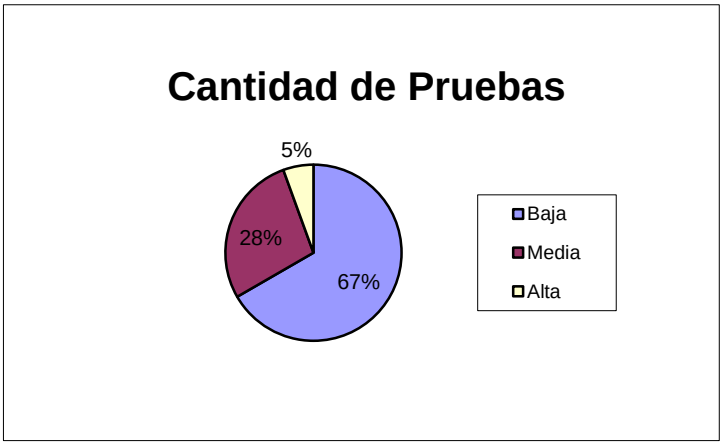


Ilustración 19: Representación del valor en % del atributo Cantidad de pruebas.

Los resultados obtenidos durante la evaluación de las clases y sus respectivas relaciones, mediante los atributos de la métrica RC, demuestran que las clases del diseño poseen un bajo acoplamiento, ya que para este atributo las categorías ninguno y bajo sumaron un 66% del total, mostrando por otra parte en las categorías alta y media del atributo reutilización un total de 95%. Los atributos complejidad de mantenimiento y cantidad de pruebas, sumaron un 95% igualmente en las categorías baja y media, lo que demuestra que no es necesario un elevado esfuerzo en el momento de realizar cambios, rectificaciones y pruebas al software.

2.3.8 Estándar de codificación.

Un estándar de codificación completo comprende todos los aspectos de la generación de código y es propio de cada programador. Para la realización de XADIM se definió el siguiente estándar de codificación:

Generales: Todo el código para este sistema se desarrolló en el lenguaje Android usando la librería gson-2.2.4.

Indentación: Todo el código desarrollado tendrá una indentación de 8 espacios ya que es la utilizada en el kernel de Linux, ver ilustración 20.

Métodos: Se escriben en mayúsculas cada vez que empiece una nueva palabra, ver ilustración 20, línea 1.

Variables: Tienen que ser nombres descriptivos acorde con la función que desempeñan, se definen en minúsculas inicialmente, el resto de las palabras comienzan con mayúsculas; ver ilustración 20, línea 2.

Espacio dentro del código: Para una mejor lectura del código este se agrupa acorde con las acciones que realizan, haciendo uso de los espacios para concentrar las líneas que se relacionan con similares funcionalidades; ver ilustración 20, líneas 2 y 3.

```
//para las emitidas
public int MostrarIdDecisionEmitida() {
    int idDec = 0;

    for (int i = 0; i < dec.getDecisiones().size(); i++) {
        if (MostrarIdTrab() == dec.getDecisiones().get(i).getIdTrabEmisor()) {
            idDec = dec.getDecisiones().get(i).getIdDecision();
        }
    }
    return idDec;
}
```

Ilustración 20: Ejemplo de uso de los estándares de codificación.

Asignación de nombres: Cada tipo de elemento debe nombrarse con una serie de reglas determinadas.

Clases e interfaces: Las clases que poseen el formato .java se escriben comenzando por mayúsculas, en caso de las Activities ya sea simple o compuesta su nombre, ver Figura 21, mientras que los Fragments utilizan guion bajo para separar una palabra de otra cuando el nombre es complejo, ver Figura 22.

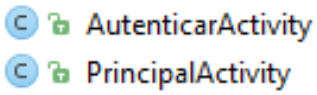


Ilustración 21: Ejemplo de uso de los estándares de codificación en las clases de tipo Activity.

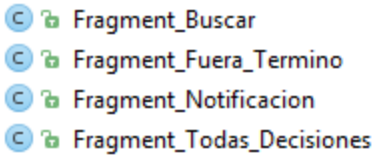


Ilustración 22: Ejemplo de uso de los estándares de codificación en las clases de tipo Fragment.

Número de declaraciones por línea: Se debe declarar cada variable en una línea distinta, de esta forma cada variable se puede comentar por separado.

```
int cantFueratermino=0;
int canttermino=0;
int cantlimite=0;
cantFueratermino=datos.CantidadFueraTermino();
canttermino=datos.CantidadTermino();
cantlimite=datos.CantidadLimite();

TextView textft = (TextView) rootView.findViewById(R.id.textView1);
textft.setText("Fuera de Término "+cantFueratermino);
TextView textt = (TextView) rootView.findViewById(R.id.textView3);
textt.setText("En Término "+canttermino);
TextView textl = (TextView) rootView.findViewById(R.id.textView2);
textl.setText("Al límite "+cantlimite);
```

Ilustración 23: Ejemplo de uso de los estándares de codificación número de declaraciones por líneas.

Código robusto: Es necesario comprobar todos los datos, para evitar errores por valores inesperados y de manera que el sistema cumpla con un correcto funcionamiento.

```
if(d.buscar(query).isEmpty() ){
    //si no enceuntra ninguna decision mostrar un sms!
    String result= "No se ha encontrado ningún resultado";
   Toast.makeText(getActivity(), result, Toast.LENGTH_SHORT).show();

} else {
ArrayList<Decision> listaB = d.buscar(query);
listaB.add(0,new Decision());
Lista_adaptador_Buscar adaptador = new Lista_adaptador_Buscar(getActivity(), listaB, user, pass);
lista.setAdapter(adaptador);
    /*listaB = d.mostarTodasDecisiones();
    listaB.add(0, new Decision());
    Lista_adaptador_TD adaptadortd = new Lista_adaptador_TD(getActivity(), listaB, user, pass);
    lista.setAdapter(adaptadortd);*/
registerForContextMenu(lista);
}
```

Ilustración 24: Ejemplo de uso de los estándares de codificación código robusto.

2.4 Conclusiones parciales.

Luego del desarrollo del presente capítulo se puede llegar a las siguientes conclusiones:

Del estudio de XADI se obtuvieron 14 requisitos funcionales y 7 no funcionales, los cuales se especificaron debidamente según las pautas definidas en la metodología seleccionada y fueron aprobados acorde a las necesidades del cliente.

Como parte del diseño de la solución se logró un modelo que empleara buenas prácticas y los patrones mencionados anteriormente que posibilitaron una organización adecuada del desarrollo y un mayor entendimiento de su arquitectura.

Se propuso una aplicación para dispositivos móviles, basada en el uso de los patrones GRASP y GOF, estándares de codificación y se valida el diseño con el uso de las métricas TOC y RC; demostrando que los indicadores de reutilización, complejidad y responsabilidad son aceptables.

Capítulo 3. Resultados y validación del sistema.

3.1 Introducción.

En el capítulo se generan las pruebas y validaciones de la solución, las cuales constituyen un instrumento para comprobar el nivel de calidad del producto. Las pruebas se dividen en dos grupos: pruebas de caja blanca, encargadas de verificar el código, la cual es realizada por los programadores, y pruebas de caja negra, destinadas a evaluar si al terminar una iteración se consiguió la funcionalidad requerida diseñadas por el cliente. Se muestran los resultados de la validación realizada mediante una encuesta donde se muestra la satisfacción del cliente en cuanto a elementos como: visibilidad y disponibilidad.

3.2 Fase III: Revisión del Sprint.

Cuando se finaliza el Sprint se realizará una revisión del incremento que se ha generado. Se presentarán los resultados finales y una versión, esto ayudará a mejorar el *feedback* [14]con el cliente. (39)

3.2.1 Pruebas.

Scrum como metodología ágil propone la realización constante de pruebas en cada una de las iteraciones realizadas a lo largo de la implementación del software. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones. Scrum divide las pruebas del sistema en dos grupos: pruebas de caja blanca, las cuales se concentran en los componentes individuales asegurándose que funcionen de manera apropiada como unidad y pruebas de caja negra o pruebas funcionales destinadas a evaluar si al concluir un *sprint* se consiguió la funcionalidad requerida diseñadas por el cliente final. De esta forma se logra que el producto finalmente concluido satisfaga en gran medida lo que el cliente necesita, obteniendo como resultado un producto entregable. (55)

[14] Feedback: retroalimentación.

3.2.1.1 Pruebas de Caja Blanca.

Las pruebas de caja blanca del software se basan en un examen cercano al detalle procedimental. Se prueban las rutas lógicas del software y la colaboración entre componentes, al proporcionar casos de pruebas que ejerciten conjuntos específicos de condiciones, bucles o ambos. (40)

Resultados de las pruebas de Caja Blanca:

A continuación, se muestra el resultado de aplicar la técnica del camino básico al método PosicionNoDecision de la clase inicial.java, que se encarga de avanzar a la vista que muestra la lista de las decisiones en término tras pulsar la imagen correspondiente.

En la siguiente figura se muestra el grafo dirigido del flujo asociado al algoritmo PosicionNoDecision.

```
public int PosicionNoDecision(String no) {
    for (int i = 0; i < dec.getDecisiones().size(); i++) {
        if (Integer.valueOf(no) == dec.getDecisiones().get(i).getNoDecision()) {
            return i;
        }
    }
    return -1;
}
```

Ilustración 25: Método PosicionNoDecision.

En la siguiente figura se muestra el grafo dirigido del flujo asociado al algoritmo PosicionNoDecision.

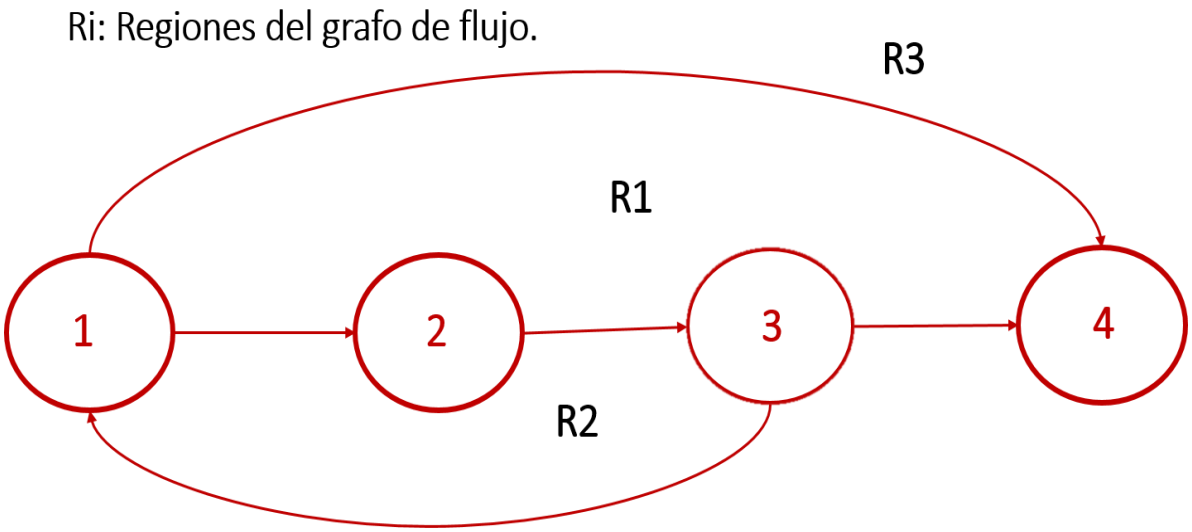


Ilustración 26: Grafo del flujo asociado al método PosicionNoDecision..

Tomando como referencia al grafo dirigido del flujo se procede a calcular la complejidad ciclomática empleando las tres variantes definidas.

Existen 3 regiones.

V(G) = 5 aristas – 4 nodos + 2 = 3

V(G) = 2 nodos predicados +1 = 3

La complejidad ciclomática es igual a 3, lo que significa que existen 3 posibles caminos linealmente independientes y representa el mínimo número de casos de prueba para el algoritmo, a continuación, se muestran los caminos existentes.

Tabla 4: Caminos por donde el flujo puede circular

Número	Camino
1	1-2-4
2	1-4
3	1-2-3-1-4

El próximo paso es ejecutar los casos de pruebas para cada camino y se compara con los resultados esperados, verificando que las instrucciones se ejecuten por lo menos una vez. A continuación, se muestran los casos de prueba para los caminos básicos identificados.

Tabla 5: Caso de prueba para el camino básico 1.

Caso de prueba para el camino básico 1	
Descripción	Comprueba que exista al menos una decisión en la lista de decisiones.
Condiciones de ejecución	El identificador, pasado por parámetro, no debe corresponderse con ninguna decisión de la lista.

Entrada	Cadena que identificaría a una decisión.
Resultado esperado	-1 → No existe la decisión.

Tabla 6: Caso de prueba para el camino 2.

Caso de prueba para el camino básico 2	
Descripción	Comprueba que no existe ninguna decisión en la lista de decisiones.
Condiciones de ejecución	La lista de decisiones debe estar vacía.
Entrada	Cadena que identificaría a una decisión.
Resultado esperado	-1 → No existe la decisión.

Tabla 7: Caso de prueba para el camino 3.

Caso de prueba para el camino básico 3	
Descripción	Comprueba que exista al menos una decisión en la lista de decisiones.
Condiciones de ejecución	El identificador, pasado por parámetro, debe corresponderse con alguna decisión de la lista.
Entrada	Cadena que identificaría a una decisión.
Resultado esperado	Posición de la decisión a la cual le corresponde el identificador pasado parámetro.

Después de ejecutar los casos de prueba diseñados se comprobó que cada instrucción se ejecuta al menos una vez, teniendo en cuenta todas las condiciones lógicas en sus variantes verdaderas y falsas; donde para cada camino se realiza un caso de prueba y se verificó que los resultados obtenidos no contienen errores.

3.2.1.2 Pruebas de Caja Negra.

Las pruebas de caja negra se centran en los requisitos funcionales del software (56). Mediante las técnicas de prueba de caja negra se obtiene un conjunto de casos de prueba que intentan encontrar errores de las siguientes categorías: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a bases de datos externas, errores de rendimiento y errores de inicialización y de terminación.

Resultados de las pruebas de Caja Negra:

Como resultado de realizar la prueba de Caja Negra, mediante el método de partición de equivalencia se encontraron varias no conformidades. La no conformidad en un sistema de calidad, se origina cuando se está incumpliendo con los requisitos del manual de calidad. A continuación, se muestra un gráfico con las no conformidades encontradas en XADIM.

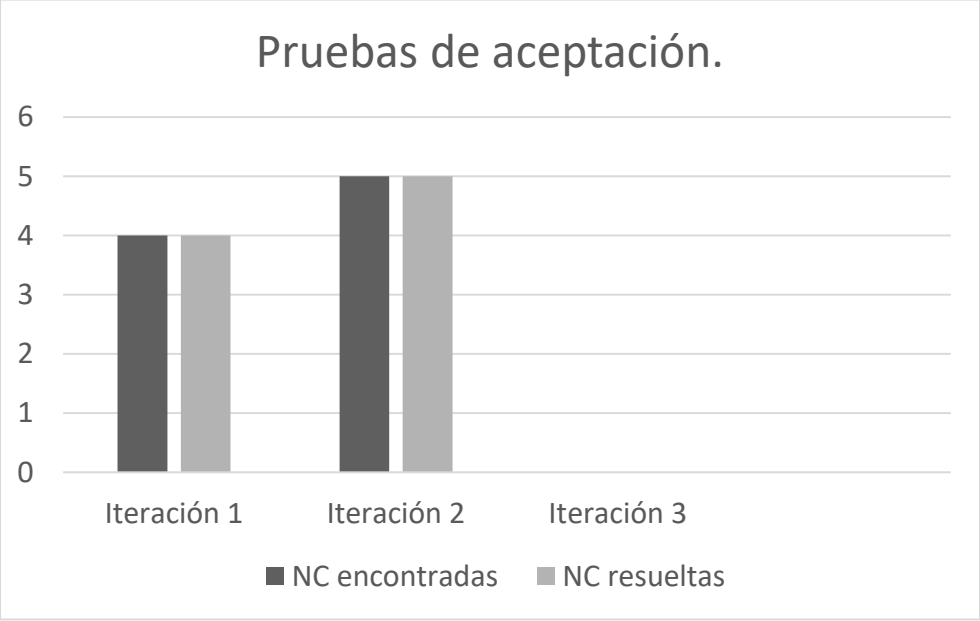


Ilustración 27: Resultados de aplicar las pruebas de caja negra.

En la primera iteración se encontraron cuatro no conformidades:

- Se quedaba guardado el usuario y contraseña afectando la seguridad de los datos contenidos en la aplicación.
- La funcionalidad buscar mostraba todas las decisiones y no las correspondientes con la búsqueda.
- Faltas de ortografía en los detalles de la decisión.
- Al presionar el botón de atrás del teléfono salía automáticamente a la vista de autenticación.

Se le dio solución a las cuatro, en la segunda iteración se encontraron cinco no conformidades:

- La funcionalidad buscar obtiene los resultados por fecha completa, pero no por día, mes o año, ni por emisor o destinatario.
- No se anulan las decisiones.
- Al ir atrás en Detalles no redirecciona a la vista anterior.
- En el listado de decisiones emitidas no tiene sentido mostrar el emisor, ya que es el propio usuario.
- La cantidad de decisiones en término, al límite y fuera de término, no estaba mostrando la cantidad que debería según la fecha actual.

Finalmente se resolvieron 9 no conformidades y en la tercera iteración no se encontraron no conformidades.

3.2.2 Validación de la investigación.

Como parte de la validación de las variables de la investigación (visibilidad y disponibilidad) se desarrolló un estudio de indicadores definidos a partir de las necesidades de los clientes. A continuación, se listan cada uno de estos indicadores:

1. Consulta de los datos del acuerdo.
2. Notificar los vencimientos de término.
3. Visualización de contenidos y funcionalidades.
4. Navegación por la plataforma.
5. Demora en la búsqueda de una decisión.

Con el objetivo de lograr una comparación en dos momentos, un antes con el uso de la aplicación XADI desde el navegador del dispositivo móvil y un después utilizando XADIM, a partir de los indicadores antes definidos como se visualiza en la tabla 8.

Tabla 8: Comparación de las necesidades del cliente antes y después del uso del sistema.

Necesidades del cliente	Antes	Después
Consulta de los datos del acuerdo.	❖ Se necesita de una conexión con la plataforma web para poder consultar los datos del acuerdo. ❖ La rapidez de esta consulta depende de la conexión.	❖ Se puede acceder a los datos relacionados con los acuerdos en cualquier momento o lugar desde su dispositivo móvil. ❖ La consulta se hace de manera rápida y fácil.
Notificar los vencimientos de término.	❖ Se necesita de una conexión para poder recibir y enviar las notificaciones pertinentes a los acuerdos.	❖ La primera vez que se entra al sistema quedan todos los datos guardados y recibe una notificación de cuantas notificaciones tiene.
Visualización de contenidos y funcionalidades.	❖ En el sistema web los componentes están ubicados horizontalmente de manera que visualizarlo en el móvil resulta incómodo.	❖ Debido a que es una aplicación realizada específicamente para un dispositivo móvil posee los componentes acomodados a la vista del usuario, así como los contenidos de los acuerdos.
Navegación por la plataforma.	❖ Es imprescindible la existencia de una conexión para la navegación y uso de la plataforma.	❖ No se necesita una conexión para acceder y navegar por la aplicación.

Demora en la búsqueda de una decisión.	❖ La demora en la búsqueda depende primeramente de la conexión existente y seguidamente de la cantidad de información.	❖ La búsqueda se realiza de manera rápida debido a que no influye ningún tipo de conexión, ni la cantidad de datos, debido a que estos son almacenados en Json, una biblioteca rápida para la consulta y almacenamiento de datos.

Como parte de las pruebas realizadas al sistema para comprobar la veracidad de esta comparación se aplicó la técnica de juicio de experto, ya que es necesario la valoración de los especialistas para verificar la mejora de estos indicadores en el sistema realizado, dando una valoración de cada uno de los resultados obtenidos.

Juicio de expertos

El juicio de expertos es un método de validación útil para verificar la fiabilidad de una investigación que se define como una opinión informada de personas con trayectoria en el tema, que son reconocidas por otros como expertos calificados, y que pueden dar información, evidencia, juicios y valoraciones. (57)

Para dar respuesta al problema definido en la investigación se construyó una aplicación utilizando la herramienta Android Studio. Mediante esta se muestran los resultados atendiendo a un conjunto de indicadores, que posteriormente se someten al criterio del panel de expertos.

Panel de expertos

Al frente del panel de expertos se encuentra la especialista B en ciencias informáticas Yamilka Rios La Hoz, la cual es Jefa y parte del equipo de XADI. Otros miembros del panel de expertos son: Yelienny Barroso Mainegra, miembro del consejo de dirección; Director de calidad del CEGEL Yordanis Garcia Leyva, las especialistas de calidad Elizabeth Enriquez Guisado y Isis Bertami Barrios, además de 4 especialistas del proyecto XADI.

Criterios de los expertos

Para la evaluación del panel de expertos se utilizaron los indicadores antes definidos y sus resultados se basan en el porcentaje del cumplimento a los objetivos definidos en la investigación. Donde se

considera que si el criterio de los expertos está por encima del 80% se cumple el objetivo del indicador. Para lo cual el panel de expertos realiza la evaluación de los resultados de la siguiente forma:

- 5: Total convergencia de criterios, si el experto considera que se cumple entre un 95%-100%.
- 4: Si el experto considera que se cumple entre un 80%-94%.
- 3: Si el experto considera que se cumple entre un 60%-79%.
- 2: Si el experto considera que se cumple entre un 40%-69%.
- 1: Si el experto considera que se cumple entre un 20%-39%.
- 0: Si el experto considera que no se cumple.

Evaluación final de los expertos

Tabla 9: Resultados de la evaluación.

Indicador	Evaluación de 5	Evaluación de 4	Evaluación de 3
1	6(67%)	3 (33%)	0
2	8 (89%)	1 (11%)	0
3	5 (56%)	3(33%)	1(11%)
4	7 (78%)	2 (22%)	0
5	9(100%)	0	0

Luego de un consenso, realizado entre todos los expertos, se concluye que el sistema cumple con el objetivo propuesto, ya que los indicadores analizados tienen los siguientes resultados (atendiendo a lo referido en la tabla 9):

- Del indicador número 1 el 67% de los expertos coinciden que se logra una mejor consulta del acuerdo entre un 95%-100% y el 33% lo considera cumplido entre un 80%-94%.
- Del indicador número 2 el 89% de los expertos coinciden que se logra una mejora en cuanto a la notificación de vencimiento de término del acuerdo entre un 95%-100% y el 11% lo considera cumplido entre un 80%-94%.

- Del indicador número 3 el 56% de los expertos coinciden que se logra una mejor visualización de contenidos y funcionalidades del acuerdo entre un 95%-100%, el 33% lo considera cumplido entre un 80%-94% y el 11% considera cumplido entre un 60%-79%.
- Del indicador número 4 el 78% de los expertos coinciden que se logra una mejor navegación por la plataforma entre un 95%-100% y el 22% lo considera cumplido entre un 80%-94%.
- Del indicador número 5 el 100% de los expertos considera que se disminuye la demora en la búsqueda de una decisión entre un 95%-100%, el 100%.

Por lo que se puede determinar que los resultados obtenidos demuestran que el desarrollo del Sistema para la Gestión de Acuerdos, Decisiones e Indicaciones, versión móvil proporciona una mejora en cuanto a visibilidad y disponibilidad de la información del sistema XADI.

3.3 Conclusiones parciales.

Del capítulo abordado se puede concluir que:

Con la aplicación de las pruebas de caja blanca a todas las clases se determinó la precisión de las instrucciones realizadas en cada sentencia presente en la solución, corroborando que todos los métodos están implementados de forma tal que no existen caminos sin resolver.

Tras aplicar las pruebas de caja negra, a cada escenario del sistema permiten asegurar que la solución responde con efectividad a cada uno de los requisitos descritos.

Para determinar la conformidad del cliente con la aplicación se definieron una serie de indicadores, los cuales fueron validados mediante juicio de expertos, quedando plasmada la satisfacción del cliente por encima del 60% en todos los casos y una mayoría del 95%-100%.

Conclusiones generales.

De manera general, durante el proceso de investigación e implementación de la aplicación para dispositivos móviles XADIM se puede llegar a las siguientes conclusiones:

Durante del estudio del arte, el análisis de las soluciones existentes evidenció la necesidad de la implementación de una nueva aplicación, utilizando metodologías, tecnologías y lenguajes orientados a dispositivos móviles, teniendo en cuenta las tendencias actuales a nivel mundial, nacional y de la universidad.

Se desarrolló un Sistema para la gestión de Acuerdos, Decisiones e Indicaciones versión móvil (XADIM), teniendo en cuenta los elementos de diseño para móviles, haciendo uso de buenas prácticas y patrones de diseño contribuyendo a la adecuada organización del proceso de desarrollo y entendimiento de este.

A partir de las pruebas de caja blanca y caja negra realizadas a la aplicación y la evaluación del panel de expertos, se pudo verificar que XADIM contribuye a elevar la visibilidad y disponibilidad de la información del XADI.

Recomendaciones.

En aras de continuar mejorando el resultado de la investigación y el producto final, se proponen las siguientes recomendaciones:

- Vincular la aplicación con los servicios web del sistema XADI.
- Agregar una nueva funcionalidad que muestre un aviso con las notificaciones aun sin estar abierta la aplicación utilizando los datos del último usuario logueado.
- Adicionar una funcionalidad que permita añadir una nueva decisión.

Referencias Bibliográficas.

1. FOMBONA CADAVIECO, Javier. *Realidad aumentada, una evolución de las aplicaciones de los dispositivos móviles.* 2012.

2. DEVELOPERS, Android. *What is android. 2011.*

3. *Sistemas Operativos Móviles.* CASTELLANOS, Luis. 12, Mérida : s.n., 03 de 10 de 2016, Revista De Tecnología y Otras Cosas, Vol. 03.

4. REYES, Melki. IphoneandoRD. *Los 5 mejores Sistemas operativos para celulares.* [En línea] [Citado el: 16 de 12 de 2016.] http://iphoneandord.com/los-5-mejores-sistemas-operativos-para-celulares/.

5. GIRONÉS, Jesús Tomás. *El gran libro de Android. Marcombo.* 2012.

6. CEPAL, N. U. *Avances en el acceso y el uso de las tecnologías de la información y la comunicación en América Latina y el Caribe. .* 2008-2010.

7. SAMPIERI, Roberto Hernández, et al. *Metodología de la investigación.* México: Mcgraw-hill : s.n., 1998.

8. FISHER, Roger, et al. *Si.. de acuerdo: como negociar sin ceder. Norma.* 1993.

9. *CONSIDERACIONES EN TORNO A SU DEFINICIÓN.* FRANCO, Simental y VÍCTOR, A. 21-22, s.l. : Revista de Derecho Privado, septiembre de 2008-abril de 2009. p. 21-22..

10. *Diccionario del español jurídico. .* AGUILAR, Elena Cianca, FRANCO, Emilio Gavilanes y GERMÁN, Montserrat Montés. no 313, s.l. : BOLETÍN DE LA REAL ACADEMIA ESPAÑOLA, 2016, Vol. vol. 96. p. 357-359..

11. WordReference.com. *Online Language Dictonaries.* [En línea] 2017. [Citado el: 10 de Enero de 2017.] http://www.wordreference.com/definicion/acuerdo.

12. CASTRO, José Luis Espíndola. *Análisis de problemas y toma de decisiones.* s.l. : Pearson Educación, 2005.

13. *Diccionario de la Lengua Española .* DE LA LENGUA, Real Academia, et al. Madrid : Real Academia Española, 1992, Vol. vol. I.

14. *Sistemas de Información.* LAUDON, F. y LAUDON, J. México : Editorial Diana , 1996.

15. SOMMERVILLE GALIPIENSO, Ian y Isabel, ALFONSO María. *Ingeniería del software. Pearson Educación.* 2005.

16. MAGRO SEGURA, Óscar. *Diseño e implementación de una aplicación Android para recordatorios.* 2011.

17. PACHECO MARTÍN, Víctor. *Diseño e implementación de una aplicación distribuida de gestión de inventario para dispositivos móviles.* 2011.

18. *ACIMED.* BONELL ROSABAL, Lic. Sheyla, y otros. no.4, Vol. vol.22.

19. BLANCO, Dr. en Ing. Sist. Telemáticos Paco. *Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone. .* 2009. p. 1-30..

20. *Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual. Revista de Tecnología.* BALAGUERA, Yohn Daniel Amaya. no 2, 2015, Vol. vol. 12. p. 111-124..

21. *Metodología para el desarrollo de aplicaciones móviles. Revista Tecnura.* MANTILLA, Maira Cecilia Gasca, ARIZA, Luis Leonardo Camargo y DELGADO, Byron Medina. no 40, 2014, Vol. vol. 18. p. 20-35.

22. GONZÁLEZ PERSONALIDAD, Rey. *Comunicación. Desarrollo. Pueblo y Educación. La Habana.* 1995. p. 1-27.

23. BRAVO, Muñoz y BOLÍVAR, Jimmy. Estudio de las fases de ciclo de desarrollo de software educativo con empleo de metodología ágil SCRUM. 2015.

24. ScrumManager. *Artefactos.* [En línea] 2017. [Citado el: 10 de Enero de 2017.] http://www.scrummanager.net/bok/index.php?title=Artefactos, Artefactos - Scrum Manager BoK..

25. VIANA, Aitor. *Linux v2. 6 en plataformas espaciales. En Libro de actas. .* 2005. p. 166-173..

26. *Google Android: An emerging software platform for mobile devices. International Journal on Computer Science and Engineering.* GANDHEWAR, Nisarg y SHEIKH, Rahila. no 1, 2010, Vol. vol. 1. p. 12-17..

27. *Open handset alliance. Retrieved August.* ALLIANCE, Open Handset. 2011, Vol. vol. 26. p. 2011.

28. Vico, Angel J. La columna 80. [En línea] 2017. [Citado el: 10 de Enero de 2017.] https://columna80.wordpress.com/.

29. DOMÍNGUEZ MEDINA, José Antonio. *Desarrollo de aplicación móvil utilizando conceptos de realidad aumentada (AR). Tesis de Licenciatura. Universitat Oberta de Catalunya.* 2011.

30. VERDECIA, Miguel Enrique y GONZÁLES, Osvaldo Manuel. *Visor de reportes para móviles con sistema operativo Android.* Habana : s.n., 2013. tesis.

31. DELGADO CORRALES, Carlos Yankiel. *Módulo de comunicación entre dispositivos móviles y el SCADA Guardián del Alba.* Cuidad de La Habana : s.n., 2013.

32. TORRES Remón, Manuel. *Fundamentos de programación, lenguajes y técnicas de programación.* 2012. ISBN: 978-607-622-621-6 216.

33. RODRÍGUEZ SÁNCHEZ, Tamara. *Programa de mejora: Metodología de desarrollo para la Actividad productiva de la UCI.* UCI. La Habana : s.n., 2015. Informe.

34. PALACIOS AGUILAR, Pedro. *Propuesta de implementación de un marco de trabajo para el desarrollo de aplicaciones Android.* 2015.

35. BLANCO, Hermes, et al. *Manual de Instalación DSpace en OpenSuse 11. x.* 2011.

36. *Balsamiq mockups. Online .* STUDIOS, Balsamiq. 2015.

37. S.A.C., Targetware Informática. software.com.ar. [En línea] 2007-2017. [Citado el: 10 de Enero de 2017.] http://www.software.com.ar/p/visual-paradigm-para-uml.

38. Standard., ECMA-404 The JSON Data Interchange. Introducción a JSON. [En línea] 2017. [Citado el: 20 de Febrero de 2017.] http://www.json.org/json-es.html.

39. GALLEGO, Manuel Trigas. *Metodología Scrum. Gestión de Proyectos Informáticos, http://openaccess. uoc. edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria. pdf.* 2012.

40. PRESSMAN, R. S. *Ingeniería del Software UIn enfoque práctico, Septima edición ed.* 2010.

41. MÉNDEZ, Gonzalo. *Especificación de Requisitos según el estándar de IEEE 830.* s.l. : Facultad de Informática, Universidad Complutense de Madrid, 2008.

42. LI, Feng-Lin, et al. *Non-functional requirements as qualities, with a spice of ontology. En Requirements Engineering Conference (RE).* s.l. : IEEE 22nd International, 2014. p. 293-302..

43. *Metodologías Ágiles en el Desarrollo de Software.* CANÓS, José H., LETELIER, Patricio y PENADÉS, Mª Carmen. no 10, 2003, Vol. vol. 1. p. 1-8.

44. GÓMEZ FUENTES, María del Carmen. *MATERIAL DIDÁCTICO NOTAS DEL CURSO ANÁLISIS DE REQUERIMIENTOS.* 2011.

45. COBOS, Raúl. KATADE. [En línea] 2016. [Citado el: 26 de Marzo de 2017.] http://katade.com.

46. develapps. *Desarrollo de aplicaciones móviles para las empresas.* [En línea] 2016. [Citado el: 26 de Marzo de 2017.] http://www.develapps.com/es/.

47. *Diseño de la base de datos para sistemas de digitalización y gestión de medias. Revista de Informática Educativa y Medios Audiovisuales.* RONDÓN, Y., DOMÍNGUEZ, L. y BERENGUER, A. no 15, 2011, Vol. vol. 8. p. 17-25.

48. *Aplicación de Patrones de Diseño para Garantizar Alta Flexibilidad en el Software. Tecnología & Desarrollo (Trujillo).* CÁRDENAS ESCALANTE, Lain. no 1, 2016, Vol. vol. 12. p. 77-82..

49. OLIVARES ROJAS, MC Juan Carlos. *Patrones de Diseño.*

50. LARMAN, Craig Pearson. *UML y Patrones.* 1999.

51. GENDLIN, Eugene T. *Thinking beyond patterns: Body, language, and situations. P. Lang.* 1991.

52. GAMMA, Erich. *Design patterns: elements of reusable object-oriented software. Pearson Education India.* 1995.

53. CECHICH, Alejandra y MOORE, Richard. *Una especificación precisa para patrones GoF. En III Workshop de Investigadores en Ciencias de la Computación.* 2001.

54. CATALDI, Zulma. *Una metodología para el diseño, desarrollo y evaluación de software educativo. Tesis Doctoral.* s.l. : Facultad de Informática, 2000.

55. CHANCUSI, Toapanta y MANUEL, Kleber. *Método Ágil Scrum, aplicado a la implantación de un sistema informático para el proceso de recolección masiva de información con Tecnología Móvil. 2012. Tesis Doctoral. SANGOLQUÍ/ESPE/2012.*

56. Pressman, Roger S. *Ingeniería de Software. Un enfoque práctico.* Sexta. Nueva York : McGraw-Hill, 2005. 0072853182.

57. DÍAZ, Francisco Javier, et al. *Herramientas open source para testing de aplicaciones Web.* En XV Congreso Argentino de Ciencias de la Computación : s.n., 2009.

Anexos.

Anexo 1: Instrumento utilizado para aplicar la métrica TOC.

	Categoria	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

Anexo 2: Instrumento utilizado para aplicar la métrica RC.

	Categoria	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

	Categoria	Criterio
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

	Categoria	Criterio
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.

	Categoria	Criterio
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.