UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Aplicación móvil para interacción con el Juez en Línea Caribeño

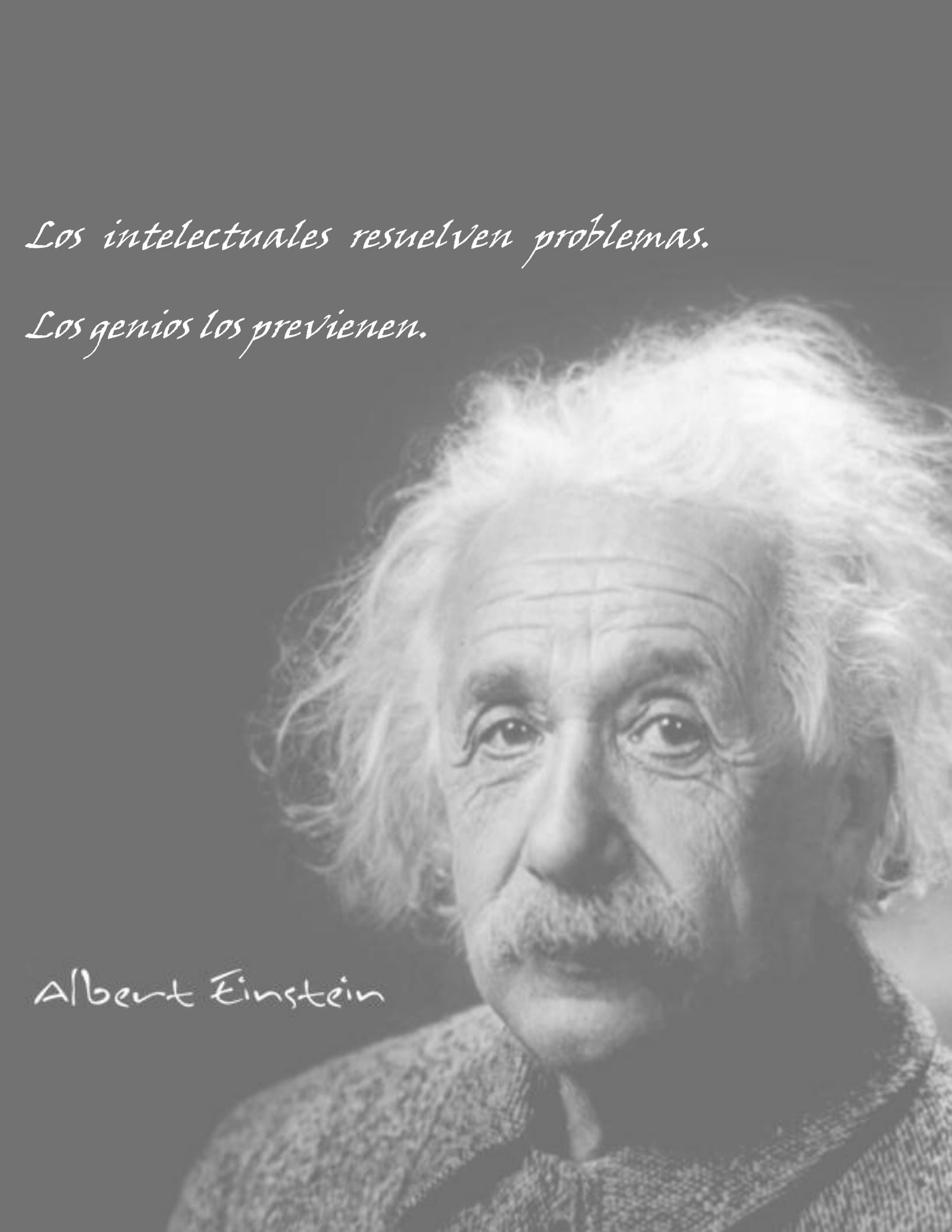
TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO
DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Osvel Alvarez Jacomino

Tutor: Ing. Yonny Mondelo Hernández

Cotutor: Ing. Ivis Leydis Rodríguez Cabrera

La Habana, junio de 2016


Los intelectuales resuelven problemas.
Los genios los previenen.
Albert Einstein

Declaración de autoría

Declaro ser el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Osvel Alvarez Jacomino

Autor

_____

Ing. Yonny Mondelo Hernández

Tutor

_____

Ing. Ivis Leydis Rodríguez Cabrera

Cotutor

Dedicatoria

A mis padres.

A mi hermano.

Agradecimientos

A esos que no son perfectos, pero son mis padres. A ellos agradezco por darme la vida y por enseñarme a luchar por las cosas que quiero. Por tener paciencia conmigo desde mis primeros pasos, incluso cuando no quería comer y con dedicación me daban un pedacito de su corazón en cada bocado. Hoy soy ingeniero gracias a ustedes, y prometo que seré en un futuro como han sido conmigo. Prometo que tendré la paciencia y dedicación para que cuando lo necesiten, darles un pedacito de mi corazón en cada bocado que tenga que brindarles.

A ese típico hermano mayor que molesta al más pequeño. Hoy es que me doy cuenta que lo hacías por amor y doy gracias a haber crecido a tu lado. Tú siempre has sido duro conmigo y me has regañado cuando has tenido que hacerlo. En otro momento nos peleábamos por eso, pero ahora sé que siempre quisiste lo mejor para mí, hoy estoy aquí gracias a ti también. Gracias además por estar siempre pendiente de este trabajo, en el cual, a pesar de no saber nada del tema, me diste muy buenos consejos y buenas ideas.

A mis sobrinitas, Alexandra y Sophia, ustedes nunca recordarán este momento, pero yo siempre las recordaré por todas esas veces que cuando pensaba que no podía lograrlo miraba a mi lado y ahí estaban como musas en mi mente.

A mis abuelos, Miriam y Alfredo, donde quiera que estén, les agradezco por estar a mi lado mientras pudieron. Ahora que no están físicamente, sé que me cuidan en cada momento de mi vida.

A Lula, que con su experiencia ha sido guía incondicional en cada paso que he dado y sé que estarás conmigo por siempre.

Papo, no te quedas atrás, aunque la muerte nos separó cuando aún era pequeño, gracias por esos recuerdos felices que inundan mi mente.

A mis amigos incondicionales: Felo, Frank, Leyna, Anel, Marla, Alexander, Adrián, Cesar, Yaritza, Yinelis, Daimel. Faltan muchos por mencionar, pero solo con nombres no se puede llenar todas las páginas de esta tesis. Los que están y los que no, siempre estarán en mi corazón como parte de la etapa más bella de mi vida.

Resumen

Los jueces en línea son aplicaciones web, que automatizan la evaluación de las respuestas a problemas de programación de competencia. Estos sistemas se asocian normalmente con entornos académicos. Su uso, en la mayoría de los casos se ve limitado a las computadoras. Debido a las grandes posibilidades de conexión y los avances tecnológicos, en la actualidad se ha generalizado la utilización de dispositivos móviles.

El presente trabajo tiene como objetivo desarrollar una aplicación móvil que permita aprovechar la conectividad y movilidad que brindan estos dispositivos para hacer uso del Juez en Línea Caribeño. Así como permitirá también el uso de algunas características sin necesidad de una conexión a internet. De esta forma se podrá llegar a un mayor número de usuarios e incentivar el estudio de la programación competitiva.

Para llevar a cabo el desarrollo del sistema, se estudiaron las metodologías, herramientas y tecnologías de desarrollo de software a emplear durante la implementación. Por último, se obtuvo un sistema que responde a los objetivos planteados, permitiendo la interacción con el usuario desde cualquier dispositivo móvil con sistema operativo Android y que cuente con una conexión a internet.

Palabras claves: aplicación móvil, dispositivo móvil, juez en línea.

Índice General

Índice de Tablas IX

Índice de Figuras X

Introducción 1

Capítulo 1: Fundamentación Teórica 6

1.1. Introducción 6

1.2. Conceptos asociados al dominio del problema 6

1.2.1. Teléfono inteligente 6

1.2.2. Aplicación Android 6

1.2.3. Tecnología móvil 7

1.3. Servicio Web 7

1.3.1. Arquitectura Orientada a Servicios 8

1.3.1. REST 8

1.3.1. Selección de servicios web 9

1.4. Juez en Línea 9

1.4.1. Juez en Línea Caribeño. 10

1.5. Sistema Operativo 10

1.6. SO Android 11

1.6.1. Arquitectura del SO Android 11

1.6.2. Android SDK 14

1.7. Ambiente de desarrollo 16

1.7.1. Metodologías de desarrollo de software 16

1.7.2. Programación Extrema 16

1.7.3. AUP 17
1.7.4. Mobile-D 18
1.7.5. Selección de Metodología de desarrollo 19
1.8. Lenguaje de modelado 19
1.9. Herramienta CASE para la modelación del sistema 20
1.9.1. Dia 20
1.9.2. Rational Rose 21
1.9.3. Visual Paradigm 21
1.9.4. Selección de la herramienta 22
1.10. Lenguaje de programación 22
1.10.1. Java 22
1.11. Gestor de base de datos 22
1.11.1. SQLite 23
1.12. Entorno de desarrollo integrado 24
1.12.1. Eclipse 24
1.12.2. Android Studio 24
1.12.3. Selección del Entorno de Desarrollo 25
1.13. Conclusiones parciales 25
Capítulo 2: Propuesta de solución 26
2.1. Introducción al capítulo 26
2.2. Definición de alcance 26
2.3. Descripción del sistema 28
2.4. Modelo de Dominio 28
2.4.1. Definición de las clases del Modelo de Dominio 29
2.5. Patrón arquitectónico 30

2.5.1. Patrón arquitectónico N-Capas 30
2.6. Diagrama de clases del diseño 32
2.7. Patrones de diseño 34
2.7.1. Patrones GRASP 35
2.7.2. Patrones GoF 36
2.8. Lista de verificación del plan de proyecto 37
2.9. Puesta en marcha 39
2.10. Planificación inicial 39
2.10.1. Interfaz de usuario 39
2.10.2. Tarjeta de historia 40
2.11. Modelo de datos 42
2.12. Modelo de despliegue 43
2.12.1. Diagrama de despliegue 43
2.13. Conclusiones parciales 43
Capítulo 3: Implementación y pruebas 44
3.1. Introducción al capítulo 44
3.2. Estabilización 44
3.3. Implementación 45
3.3.1. Día de planificación 45
3.3.2. Día de trabajo 45
3.3.1. Día de liberación 45
3.4. Tareas de ingeniería 46
3.5. Estándares de nomenclatura y codificación utilizados 48
3.5.1. Nomenclatura en general 48
3.5.2. Estilos de codificación 49

3.6. Pruebas de software aplicadas 50

3.6.1. Pruebas unitarias 50

3.6.2. Resultado de las pruebas 52

3.7. Conclusiones parciales 54

Conclusiones 55

Recomendaciones 56

Referencias Bibliográficas 57

Índice de Tablas

Tabla 1: Uso de versiones de la plataforma Android 15
Tabla 2: Requisitos funcionales 26
Tabla 3: Lista de verificación del plan del proyecto 37
Tabla 4: Funcionalidad seleccionada para desarrollar en el día de trabajo 41
Tabla 5: Historia seleccionada para implementación 41
Tabla 6: Tarea "Solicitar listado de problemas" 46
Tabla 7: Tarea "Crear la interfaz gráfica del listado de problemas" 46
Tabla 8: Tarea "Crear comportamiento del botón flotante" 47

Índice de Figuras

Fig. 1: Arquitectura de Android (Google, 2015) 14
Fig. 2: Distribución de versiones de Android (Google, 2015) 15
Fig. 3: Diagrama de clases del Modelo de Dominio 28
Fig. 4: Diagrama de la arquitectura del proyecto 31
Fig. 5: DCD Mostrar problema 33
Fig. 6: Interfaz de la historia 17 40
Fig. 7: Interfaz de la historia 18 40
Fig. 8: Interfaz de la historia 19 40
Fig. 9: Interfaz de la historia 20 40
Fig. 10: Diagrama Entidad-Relación 42
Fig. 11: Diagrama de despliegue 43
Fig. 12: Resultado de las pruebas unitarias 52
Fig. 13: Resultados de las pruebas funcionales 53

Introducción

Desde hace unos años la sociedad actual se ha visto inmersa en un rápido proceso de desarrollo y avance tecnológico. Las nuevas capacidades de almacenamiento de la información junto a las altas velocidades de procesamiento y la gran facilidad de comunicación entre los dispositivos que brinda el nuevo mundo del internet, han permitido ubicar las nuevas alternativas móviles en una posición cimera en la vida humana. De esta forma se convierten en punto de mira para diferentes sectores de la sociedad.

Según (Cubadebate, 2016) “En 2020 habrá 5 500 millones de usuarios de dispositivos móviles en el mundo, el 70% de la población mundial, con una media de 1,5 conexiones per cápita”. La situación en América Latina no es muy diferente, el mercado de teléfonos móviles en la región es uno de los mayores del mundo y actualmente se estima existen más de 200 millones de dispositivos inteligentes (Granados, 2015). Cuba ha comenzado la etapa de pruebas de un nuevo sistema de comunicación para llevar internet a los hogares, así como el desarrollo de las zonas WiFi brindadas por la Empresa de Telecomunicaciones de Cuba (ETECSA). Por tales motivos el uso de dispositivos móviles en el país ha crecido considerablemente llegando a alcanzar a finales de 2015 los tres millones de líneas móviles en uso (González, 2016). Entre los sistemas operativos instalados en dispositivos móviles predomina Android, con más del ochenta por ciento de las instalaciones en todo el mundo (Infobae, 2016).

Con el fin incorporar en el proceso de enseñanza/aprendizaje estas nuevas Tecnologías de la Información y las Comunicaciones (TICs), así como incentivar el estudio de diferentes materias, se han creado disimiles plataformas entre las que se encuentran los jueces en línea. Estos son sistemas informáticos que permiten evaluar posibles soluciones de los usuarios a miles de problemas/ejercicios de naturaleza algorítmica, matemática, física, etc. que están disponibles en sus archivos, de manera automática en tiempo real; y de esta forma incentivar el estudio de la programación. Algunos de los jueces en línea más usados en Cuba son: Sphere Online Judge (SPOJ), el de la Universidad de Valladolid (UVA), TopCoder, CodeChef y el Juez en Línea del Caribe (COJ), desarrollado en la Universidad de las Ciencias Informáticas, entre otros.

El COJ se encuentra disponible desde Junio de 2010 en la red nacional del Ministerio de Educación Superior (MES) y en Internet, con fines de entrenamiento de todos los concursantes de la Región Caribeña. Juez en Línea del Caribe es una solución elegante que surge sobre todo de la necesidad ante los problemas de conectividad que limitaban en Cuba el entrenamiento de los concursantes ACM-ICPC (Vazquez, 2012).

Por otra parte, el uso de disimiles tecnologías de redes ha hecho técnicamente posible acceder a Internet desde los móviles durante varios años, sin embargo, la navegación móvil a menudo ha sido complicada. La conversión de páginas Web diseñadas para computadoras hacia estos dispositivos es un reto, y el COJ no se encuentra exento de esto debido a que: los dispositivos móviles poseen un teclado muy pequeño para la entrada de la información y una pantalla de reducidas dimensiones por lo que el despliegue de la información es limitado. Además, el contenido de este sitio es demasiado grande para la pequeña pantalla del móvil.

Debido a esto se hace necesaria la búsqueda de alternativas que permitan adaptar el contenido del COJ a los disímiles tamaños y resoluciones de pantallas presentes en los dispositivos móviles. El diseño web adaptativo, definido por primera vez por (Marcotte, 2010) puede ser una solución factible para los problemas planteados anteriormente. Esta solución logra responder a la mayoría de las necesidades de los usuarios y de sus dispositivos. El diseño será capaz de cambiar en dependencia del tamaño y la capacidad del dispositivo. Por ejemplo, en una pantalla pequeña el usuario podrá visualizar el contenido en una columna, pero en dispositivos más grandes será visualizada la información en dos columnas.

A pesar de sus ventajas, el diseño web adaptativo presenta algunos inconvenientes como: dependencia total de una conexión a internet; falta de una navegación natural o pérdida de notificaciones y otras funcionalidades, solo posibles en aplicaciones nativas para dispositivos móviles.

Por otra parte las aplicaciones nativas pueden brindar las mismas características que presenta el diseño web adaptativo. Además, también permiten su uso independientemente de contar o no con una conexión a internet. La navegación e integración de la aplicación con el Sistema Operativo (SO) del dispositivo móvil se hace mucho más amena al permitir todo tipo de notificaciones, ya sea mediante componentes visuales, sonido, o vibración.

Este tipo de soluciones exige definir un método de comunicación entre la aplicación y el sistema web. En la mayoría de los casos este intercambio de datos se realiza mediante el uso de servicios web que facilitan la interoperabilidad del sistema con todo tipo de aplicaciones. El COJ actualmente ya posee una capa de servicios web, lo que facilitará el trabajo al realizar una aplicación nativa.

Existen algunas aplicaciones para dispositivos móviles que permiten seguir las estadísticas de un usuario, ver las soluciones enviadas al juez en línea, informar de próximos eventos a desarrollar o ver los nuevos problemas a resolver, así como otras funcionalidades más. Ejemplo de este tipo de aplicaciones es Coding

Calendar que es capaz de integrarse a jueces en línea como Codechef, Codeforces, Topcoder, HackerRank, HackerEarth; otra aplicación similar existe para el juez en línea UVA llamada UVA Mobile.

Siendo analizada la principal problemática se plantea como **problema a resolver:** ¿cómo lograr la correcta visualización del COJ en dispositivos móviles con sistema operativo Android?

El **objeto de estudio** de la presente investigación es el desarrollo de aplicaciones móviles para el SO Android, enfocando el **campo de acción** a aplicaciones Android que utilizan servicios web para jueces en línea.

Para dar solución al problema planteado se define como **objetivo general** de la investigación: desarrollar una aplicación nativa para Android que permita la correcta visualización del COJ en dispositivos móviles.

La aplicación a desarrollar deberá interactuar e intercambiar datos con COJ mediante el uso de un módulo de servicios webs que brinda estandarización en la comunicación de los datos entre la actual plataforma y las futuras aplicaciones a desarrollar.

Para dar cumplimiento al objetivo general se han derivado los siguientes **objetivos específicos**:

1. Definir el marco teórico de la investigación mediante el estudio y análisis de los principales estándares de desarrollo de aplicaciones para Android y el trabajo con jueces en línea.
2. Describir la propuesta de la aplicación y el proceso realizado para su desarrollo.
3. Desarrollar la aplicación teniendo en cuenta el diseño realizado y los requisitos definidos.
4. Validar mediante pruebas de software el funcionamiento de la aplicación.

Para dar respuesta a los objetivos específicos trazados se plantea el cumplimiento de las siguientes **tareas de la investigación**:

1. Definición del marco teórico-metodológico sobre el desarrollo de aplicaciones para Android.
2. Fundamentación de las tendencias actuales, tecnologías y conceptos relacionados desarrollo de aplicaciones Android y jueces en línea.

3. Descripción de las buenas prácticas, metodologías y estándares para el desarrollo de aplicaciones Android vinculadas jueces en línea mediante servicios web.

4. Análisis de las herramientas que se utilizan en el desarrollo de aplicaciones Android y el uso de servicios web.

5. Análisis de las tecnologías, herramientas y lenguajes de programación a utilizar en el desarrollo de las funcionalidades de la aplicación.

6. Análisis del proceso de desarrollo de software a utilizar en la aplicación.

7. Definición de la propuesta de solución teniendo en cuenta los elementos del diseño de una aplicación Android.

8. Especificación de los requisitos funcionales y no funcionales que la aplicación debe cumplir.

9. Implementación de los requisitos descritos.

10. Elaboración de los casos de prueba de funcionalidad.

11. Ejecución de las pruebas de funcionalidades del sistema.

Defendiendo como **hipótesis científica:** si se desarrolla una aplicación nativa para el COJ sobre SO Android permitirá la correcta visualización del mismo en los dispositivos móviles.

Desde el punto de vista de la investigación se emplearon los métodos científicos:

Teóricos:

- Histórico – Lógico, a través del cual se profundizó en los antecedentes y tendencias actuales de los jueces en línea, así como los argumentos que antecedieron al problema.

- Analítico – Sintético, que permitió descomponer el tema para a partir de la revisión de conceptos, teorías, técnicas y herramientas, procesar la información y llegar a conclusiones.

- Modelación, el cual contribuyó a representar de manera funcional y gráfica la aplicación para la interacción con COJ.

Empíricos:

- Observación, que se emplea para conocer la situación actual de sistemas similares a la solución propuesta.

El documento consta de tres capítulos en los que se abordan diferentes temáticas. En el primero se exponen los conceptos y fundamentos generales que sirven de soporte teórico a la propuesta de solución que se plantea. Se realiza un análisis de los sistemas similares que existen actualmente, donde se explica su funcionamiento y características. Se analizan las herramientas y lenguajes de programación idóneas para el desarrollo de la aplicación Android para el COJ, así como la metodología a emplear en el desarrollo de la misma.

En el segundo capítulo se especifican los requisitos que debe cumplir la aplicación, así como la descripción de las funcionalidades. También se muestra la explicación de la arquitectura de la aplicación y los patrones de diseño utilizados para el desarrollo de la misma.

En el tercer y último capítulo se abordan los aspectos relacionados con la implementación del sistema y el proceso de pruebas utilizado. Se implementan todas las funcionalidades identificadas, para lograr un sistema que permita cumplir el objetivo general planteado. Se detallan las pruebas que se realizaron al sistema, con el objetivo de verificar la integridad del mismo y el cumplimiento de los requisitos definidos por el cliente.

Capítulo 1: Fundamentación Teórica

1.1. Introducción

En este capítulo se abordan los conceptos relacionados con la investigación, que constituyen el conocimiento fundamentalmente teórico sobre los diferentes procesos y tecnologías asociadas al desarrollo de la solución requerida. Finalmente, se seleccionan y caracterizan las herramientas, metodología de desarrollo de software y los lenguajes de modelado y de programación a utilizar en la aplicación.

1.2. Conceptos asociados al dominio del problema

Resulta necesario conocer los conceptos relacionados a la tecnología móvil y el sistema operativo Android, así como los jueces en línea ya que facilitan la comprensión del objeto de estudio de la investigación.

1.2.1. Teléfono inteligente

Un teléfono inteligente es capaz de hacer cualquiera de las tareas que hace una computadora personal, y además brinda la ventaja de una mayor movilidad. A su vez, el tamaño de la pantalla es la principal desventaja de este tipo de teléfonos.

Un teléfono inteligente combina un teléfono celular estándar con correo electrónico, navegación web, música, video, navegación GPS y otras funcionalidades importantes. Se puede decir que esta nueva generación de teléfonos es mucho más personal que una computadora personal, y normalmente el usuario lo tiene cerca sin importar donde esté (The Computer Language Company Inc., 2015).

1.2.2. Aplicación Android

Una aplicación Android es un software que se ejecuta en una plataforma de SO Android. Una aplicación de este tipo permite realizar acciones como enviar un mensaje, realizar una llamada o chequear el correo electrónico. Como la plataforma Android actualmente está pensada principalmente para dispositivos móviles, entonces la aplicación típica es diseñada para teléfonos inteligentes o tabletas (Techopedia Inc., 2016).

1.2.3. Tecnología móvil

La tecnología móvil no es más que una de las ramas existentes dentro de los sistemas de comunicación. Esta se basa exclusivamente en sus sistemas de conexión inalámbrica. Posibilita la conversión a señales electromagnéticas de los sonidos que viajan a través del aire hasta ser captadas. Posteriormente son transformadas por antenas repetidoras o vía satélite en mensajes hasta llegar a su receptor.

Los teléfonos celulares, dispositivos de frecuencia dual para el habla y la escucha han redefinido todo nuestro punto de vista de lo que era y ahora se nace en las comunicaciones, como un elemento que la hace brillar dentro de las relaciones humanas modernas.

Innumerables son los cambios que en esta nueva tecnología han surgido, incluyendo accesibilidad, facilidad, servicios y estructura física de los teléfonos celulares, aunque continúen siendo transmisores personales, independientemente de sus características (Daichendt, 2016).

1.3. Servicio Web

Según (Machuca, 2010) un servicio web es un sistema diseñado para soportar interacción máquina a máquina sobre una red. Este tiene una interface descrita en un formato procesable por una máquina. Otros sistemas interactúan con el servicio web en una manera prescrita por su descripción usando mensajes SOAP, típicamente enviados mediante HTTP con una serialización XML en relación con otros estándares relacionados con la web.

Se puede definir de manera más sencilla como un conjunto de tecnologías estándares de software para el intercambio de datos entre aplicaciones. Estos pueden ser desarrollados en una gran variedad de lenguajes para ser implementados sobre muchos tipos de redes de computadores.

Al conjunto de servicios y protocolos para los servicios web es conocido comúnmente como “Web Services Protocol Stack” y básicamente son utilizados para definir, localizar, implementar y hacer que un servicio web interactúe con otro. Este conjunto está conformado esencialmente de cuatro subconjuntos:

- Servicio de transporte
- Mensajería XML
- Descripción del servicio
- Descubrimiento de Servicios

1.3.1. Arquitectura Orientada a Servicios

Las Arquitectura Orientada a Servicios (SOA, por sus siglas en inglés) ha emergido como un paradigma de desarrollo de software. OASIS, en su Modelo de Referencia para SOA (OASIS, 2006) la define como un paradigma para organizar y utilizar capacidades distribuidas que pueden estar bajo el control de diferentes dominios o entidades (personas u organizaciones).

El funcionamiento de SOAP según IBM[1] (IBM, 2011) consiste en un mensaje creado en XML que posee tres partes:

- Una etiqueta conocida como <Envelope> la cual define un framework para describir el contenido del mensaje y sus instrucciones de proceso, esto mediante los header que son los que contienen control de información como los atributos de calidad de servicio y el body que contiene la identificación del mensaje y sus parámetros.

- Un conjunto de reglas de codificación para expresar instancias de los tipos de datos definidos en la aplicación

- Una convención que sirve para representar los llamados y respuestas a procedimientos remotos.

Los mensajes SOAP son fundamentalmente de una sola vía de transmisión entre el que envía y el que recibe, pero también pueden ser utilizados en patrones como solicitud/respuesta.

1.3.1. REST[2]

Arquitectura que se centra en la solicitud de recursos y utiliza los principios básicos de la aplicación web:

- Transporte de datos mediante HTTP por medio de las operaciones básicas :
    - Petición GET, el recurso se solicita a través de la URL al servidor web.

[1] International Business Machines Corp.
[2] Transferencia de Estado Representacional por siglas en inglés

- Petición POST, el recurso se solicita mediante un conjunto de datos
- Petición PUT, envía el recurso identificado en la URL desde el cliente hacia el servidor
- Petición DELETE, solicita al servidor que borre el recurso identificado con el URL.

- Los diferentes servicios son invocados mediante URI (identifica un recurso en internet) unificado.
- La codificación de datos es identificada mediante tipos MIME (text/html, image/gif).

REST es una arquitectura simple, que tiene buenos tiempos de respuesta entre el cliente y el servidor, presenta mayor estabilidad frente a los cambios, además de sencillez en su desarrollo para clientes; pero su inconveniente es que no se mantiene el estado por lo tanto cuando el servidor trata una solicitud lo hace de forma independiente sin recordar solicitudes anteriores.

1.3.1. Selección de servicios web

Conforme a los servicios web presentados anteriormente, para realizar una elección se debe tener en cuenta que aunque el uso de SOAP ha sido bastante difundido, no es adecuada su utilización en el sistema operativo Android, puesto que su complejidad hace que tenga un rendimiento menor en comparación con REST, además Android no posee librerías nativas para trabajar con SOAP mientras que si las tiene para REST, conjuntamente el servidor web que aloja la plataforma web del COJ no se encuentra basado en SOAP, lo que descarta esta opción y deja a REST como elección para realizar el envío de la información almacenada.

1.4. Juez en Línea

Un juez en línea es en general, un servidor que contiene descripciones de problemas de naturaleza algorítmica, matemática, física, etc. Así como conjuntos de datos para evaluar cada una de las diferentes soluciones que pueden tener dichos problemas. Un usuario de cualquier lugar del mundo puede registrarse e intentar resolver tantos problemas como desee. El principal rasgo distintivo de un juez en línea es que permite a los usuarios estudiar la programación de manera autodidacta (Vazquez, 2012).

1.4.1. Juez en Línea Caribeño.

El desarrollo del sistema base (Xtreme Online Judge) comenzó en el año 2006 bajo la "Iniciativa Xtreme", compuesta principalmente por estudiantes y profesores de la facultad 8 de la Universidad de las Ciencias Informáticas (UCI).

Luego de que la UCI se unió al movimiento ACM-ICPC y lideró la creación de la Comunidad Caribeña del ACM-ICPC, el Xtreme Online Judge fue seleccionado para publicarse en Internet como COJ v1.0, y otras personas se unieron al pequeño pero inspirado equipo de desarrollo. El COJ está disponible en Internet desde el 5 de junio de 2010.

En octubre de 2011 el sistema base fue sustituido por otro que fue reprogramado desde cero (durante casi un año) por dos jóvenes estudiantes de la UCI. Desde entonces, los equipos y métodos de desarrollo han sido mejorados para conseguir un sistema mejor.

El COJ es sistema que compila y ejecuta automáticamente el código enviado por el usuario (solución a un problema). El código fuente enviado será probado con algunas restricciones, incluyendo el tiempo de ejecución, el uso de memoria, el tamaño del código fuente, la seguridad y otras. La salida del código será capturada por el sistema y comparada con la salida correcta proporcionada por el autor del problema. El usuario obtendrá Aceptado (como sentencia o veredicto) si todas las pruebas aplicadas a su código fuente parecen estar bien. En cualquier otro caso, obtendrá una respuesta de rechazo (Mondelo Hernández, y otros, 2016).

1.5. Sistema Operativo

Un Sistema Operativo (SO) es el programa más importante que se ejecuta en un sistema de cómputo. Un SO es un programa que administra el hardware de un sistema informático. Proporciona los mecanismos apropiados para asegurar el correcto funcionamiento del sistema informático e impedir que los programas de usuario interfieran con el apropiado funcionamiento del sistema. Cada computadora debe tener un SO capaz de ejecutar otros programas o aplicaciones (Quinstreet Enterprise, 2016).

Existe una enorme y variada gama de sistemas informáticos para los que se diseñan sistemas operativos. Tal es el caso de los teléfonos móviles que tratan de llevar más lejos el concepto de telefonía, añadiendo a

los terminales móviles funciones propias de los ordenadores. Los dispositivos móviles de la actualidad proporcionan una serie de servicios basados en el tráfico de datos a través de la red, por lo que necesitan un sistema operativo propio a este tipo de escenario.

Existen varios sistemas operativos para dispositivos móviles, de los cuales se puede citar Windows Phone, desarrollado por Microsoft. Symbian OS, que es oficialmente propiedad de Nokia. Además se encuentra el iOS un producto de la empresa Apple Inc., lanzado el 29 de enero del 2007 y solo es usado en dispositivos de este fabricante. BlackBerry OS lanzado en 1999 por la empresa Research In Motion para sus dispositivos (Agrawal, 2015).

1.6. SO Android

El SO Android está pensado para instalarlo en dispositivos móviles. Se incluye un SO basado en Linux, liberado bajo la licencia de código abierto, provee además de una capa de librerías escritas en C y C++, y un marco de trabajo para el desarrollo de aplicaciones. Originalmente este marco de trabajo estaba solo basado en Java y posteriormente se liberó para aplicaciones nativas en C, aunque no es recomendable a menos que se necesite utilizar de manera excesiva al microprocesador. Incluye además una suite de aplicaciones iniciales, por ejemplo, la aplicación que manejan los contactos, posibilita también la instalación de otras aplicaciones que pueden ser descargadas desde internet (TechTarget, 2015).

El principal impulsor de esta plataforma es Google, pues es el que lo mantiene, pero en realidad para hacerlo más abierto y con el fin de lograr estandarización en el desarrollo de dispositivos móviles, a finales del 2007 se crea una organización sin fines de lucro denominada la Alianza Abierta de Dispositivos (Open Handset Alliance) (TechTarget, 2010). Incluye a más de 80 empresas del sector de fabricación de dispositivos móviles, entre los que se encuentran a Google, fabricantes de equipos como Motorola, HTC, LG, Samsung, Alcatel; operadores móviles como Telefónica, Movistar, T-Mobile, Sprint, China Mobile y los fabricantes de microprocesadores y placas de video Intel, nVidia, Qualcomm, Texas Intruments, Huawei, entre otros. Estas empresas son unas de las pocas que forman la alianza, responsables de mantener a Android como sistema operativo (Open Handset Alliance, 2009).

1.6.1. Arquitectura del SO Android

A continuación se detallan los principales componentes de la arquitectura Android:

- **Núcleo de Linux:** Android depende de un núcleo de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red, y modelo de drivers. Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del hardware.

- **Librerías nativas:** Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

    - **System C library**: una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.

    - **Media Framework**: librería basada en PacketVideo's OpenCORE; soporta codecs de reproducción y grabación de multitud de formatos de audio vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.

    - **Surface Manager**: maneja el acceso al subsistema de representación gráfica en 2D y 3D.

    - **WebKit**: soporta un moderno navegador Web utilizado en el navegador Android y en la vista Webview: Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.

    - **SGL**: motor de gráficos 2D.

    - **Librerías 3D**: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.

    - **FreeType**: fuentes en bitmap y renderizado vectorial.

    - **SQLite**: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.

    - **SSL**: proporciona servicios de encriptación Secure Socket Layer (capa de conexión segura).

- **Runtime de Android:** Posee un núcleo de librerías para la Máquina Virtual Dalvik (VDK por sus

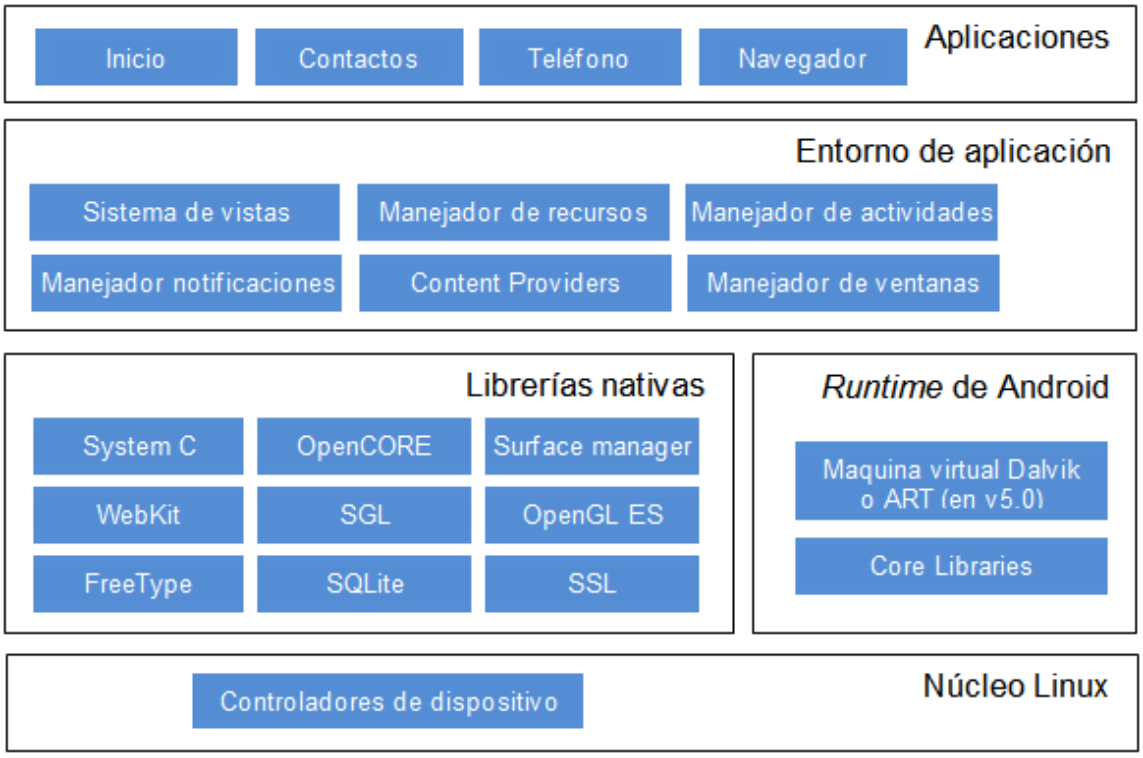
siglas en inglés) que se encarga de ejecutar las aplicaciones. Dalvik es optimizado para dispositivos móviles, por lo que consume menos memoria y provee rapidez de ejecución. A partir de Android 5.0 se reemplaza Dalvik por ART. Esta nueva máquina virtual consigue reducir el tiempo de ejecución del código Java hasta en un 33%.

- **Marco de trabajo o entorno de aplicación:** los desarrolladores tienen acceso completo a las APIs del marco de trabajo usado por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes.

  Los servicios más importantes que incluye son:

  - **Views**: extenso conjunto de vistas (parte visual de los componentes).
  - **Resource Manager**: proporciona acceso a recursos que no son en código.
  - **Activity Manager**: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
  - **Notification Manager**: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
  - **Content Providers**: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).
- **Aplicaciones:** Haciendo uso del marco de trabajo se encuentran las aplicaciones. Todas las aplicaciones, como la de contactos, los juegos o el correo usan el marco de trabajo de Android que a su vez usa el runtime y las librerías (Google, 2015).

Fig. 1: Arquitectura de Android (Google, 2015)



1.6.2. Android SDK

El Paquete de Desarrollo de Software de Android (SDK por sus siglas en inglés) es un software de desarrollo que permite a los desarrolladores crear aplicaciones para la plataforma Android haciendo uso del lenguaje de programación Java. Android SDK incluye ejemplos de proyectos con código fuente, herramientas de desarrollo y un emulador. Proporcionan bibliotecas de interfaz de programación de aplicaciones creadas con el fin de permitir un mejor uso de las técnicas a desarrollar en este sistema (Bathelot, 2012).

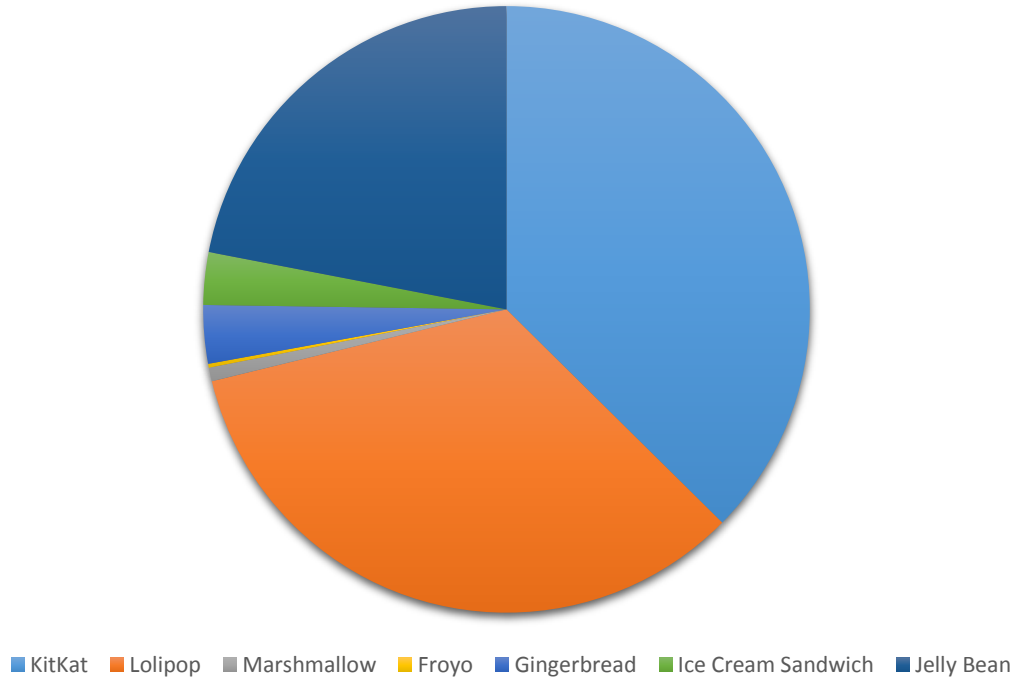
Cada vez que Google libera una nueva versión de Android también es liberada la SDK correspondiente. Para permitir el desarrollo de aplicaciones con las últimas características, los desarrolladores deben obtener e instalar cada versión de la SDK para cada dispositivo en particular.

La plataforma de desarrollo se encuentra para los sistemas operativos Windows (a partir de XP), Linux y Mac OS X (a partir de la versión 10.4.9). Cada componente puede descargarse de forma individual y añadirle librerías para facilitar su uso (Google, 2015).

Tabla 1: Uso de versiones de la plataforma Android

Versión	Nombre de Versión	API	Distribución
2.2	Froyo	8	0.2%
2.3.3 – 2.3.7	Gingerbread	10	3.0%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lolipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%

Fig. 2: Distribución de versiones de Android (Google, 2015)



Teniendo en cuenta las facilidades que brinda Android al desarrollar aplicaciones, se determina el uso de este sistema operativo abierto como base en la línea de investigación de este trabajo. Se selecciona su versión 4.4 KitKat por su impacto y uso en el mercado, así como la existencia de numerosos dispositivos con este sistema. La aplicación también será funcional para versiones de Android a partir de 4.0.3 Ice Cream. Se incluye además como marco de trabajo el SDK correspondiente para el desarrollo de la herramienta.

1.7. Ambiente de desarrollo

El desarrollo de un software es un proceso complejo, en el cual se tienen muchos elementos que permitirán su construcción con calidad. Para la implementación de una aplicación, es necesario tener en cuenta diversos aspectos de gran importancia tales como: la metodología y las tecnologías, así como los lenguajes de programación y modelado, pues de esto dependerá totalmente el éxito en el desarrollo de la misma.

1.7.1. Metodologías de desarrollo de software

Según la Real Academia Española una metodología de desarrollo de software no es más que un *"conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal"* (RAE, 2012). Una Metodología de desarrollo de software es un enfoque estructurado que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos. Existen metodologías tradicionales o robustas, centradas específicamente en el control del proceso y ágiles, que cambian significativamente algunos de los énfasis de los métodos ingenieriles para lograr un mayor enfoque solamente en el desarrollo del proyecto (Somerville, 2005). Para el desarrollo del sistema propuesto se realiza un estudio de las metodologías ágiles, puesto que según (Valdéz, 2014) las mismas proponen una estrecha relación entre el cliente y el equipo de desarrollo, no existe un contrato tradicional, se aplican a proyectos con requerimientos cambiantes o imprecisos, y están dirigidas fundamentalmente para equipos pequeños; se debe hacer entregas funcionales continuamente y el cliente es parte del equipo de desarrollo.

1.7.2. Programación Extrema

La Programación Extrema, o Extreme Programming (XP)(por sus siglas en inglés), es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo,

comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Esta metodología tiene como base la simplicidad y como objetivo principal la satisfacción del cliente; para lograrlo se deben tomar en cuenta cuatro valores fundamentales: Comunicación, Simplicidad, Retroalimentación y Coraje. XP contiene 4 fases fundamentales para llevar a cabo el proceso de desarrollo (Letelier, y otros, 2006):

- Exploración.
- Planificación.
- Iteraciones.
- Producción.

La principal **desventaja** de la metodología de desarrollo de Programación Extrema se basa en que no se tiene la definición del costo y el tiempo de desarrollo desde inicio del proceso. El sistema va creciendo después de cada entrega al cliente y nadie puede decir que el cliente no querrá una función más; se necesita de la presencia constante del usuario, lo cual en la realidad es muy difícil de lograr.

1.7.3. AUP

El AUP es una versión simplificada del Proceso Racional Unificado (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones del software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas, Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de la Base de Datos para mejorar la productividad. AUP se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto conflicto obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas identificando las debilidades desde etapas iniciales del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos ejecutables durante la base de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos. El proceso AUP establece un modelo más simple que el que aparece en RUP por lo que reúne las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño en una sola. El resto de disciplinas (Implementación, Pruebas, Despliegue,

Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP. Al igual que RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva: Concepción, Elaboración, Construcción y Transición (Inflectra, 2014).

1.7.4. Mobile-D

El objetivo de este método es conseguir ciclos de desarrollo muy rápidos en equipos muy pequeños. Fue creado en un proyecto finlandés en 2005, pero sigue estando vigente. Basado en metodologías conocidas pero aplicadas de forma estricta como: Extreme Programming, Crystal Methodologies y Rational Unified Process.

Se compone de distintas fases: exploración, inicialización, fase de producto, fase de estabilización y la fase de pruebas. Cada una tiene un día de planificación y otro de entrega.

En la fase de **exploración** se centra la atención en la planificación y a los conceptos básicos del proyecto. Aquí es donde hacemos una definición del alcance del proyecto y su establecimiento con las funcionalidades donde queremos llegar.

En esta primera fase, el equipo de desarrollo debe generar un plan y establecer las características del proyecto. Esto se realiza en tres etapas: **establecimiento actores**, **definición del alcance** y el **establecimiento de proyectos**. Las tareas asociadas a esta fase incluyen el establecimiento del cliente (los clientes que toman parte activa en el proceso de desarrollo), la planificación inicial del proyecto y los requisitos de recogida, y el establecimiento de procesos.

En la **iniciación** se configura el proyecto, identificando y preparando todos los recursos necesarios. En esta fase los desarrolladores preparan e identifican todos los recursos necesarios así como los planes para las siguientes fases. Se establece el entorno técnico como los recursos físicos, tecnológicos y de comunicaciones (incluyendo el entrenamiento del equipo de desarrollo). Esta fase se divide en cuatro etapas: la **puesta en marcha** del proyecto, la **planificación inicial**, el **día de prueba** y **día de salida**.

En la fase de **producción** se repiten las sub-fases. Se usa el desarrollo dirigido por pruebas (TDD por siglas en inglés), antes de iniciar el desarrollo de una funcionalidad debe existir una prueba que verifique su funcionamiento. En esta fase se puede decir que se lleva a cabo toda la implementación.

Después de la fase de producto llega la fase de **estabilización** en la que se realizan las acciones de integración para enganchar los posibles módulos separados en una única aplicación.

Fase de **pruebas**. Una vez parado totalmente el desarrollo se pasa una fase de testeo hasta llegar a una versión estable, según lo establecido en las primeras fases por el cliente. Si es necesario se reparan los errores, pero no se desarrolla nada nuevo.

Una vez acabada todas las fases deberíamos tener una aplicación publicable y entregable al cliente.

1.7.5. Selección de Metodología de desarrollo

Se utilizará la metodología ágil Mobile-D debido principalmente a que está enfocada directamente al desarrollo de aplicaciones para dispositivos móviles. Esta metodología facilita una documentación mucho más discreta y propicia mayor dinamismo para el desarrollo. Es apropiada para proyectos pequeños y entregas de ciclos rápidos, exigiendo que se establezca una comunicación más fluida con el cliente y que éste tenga mayor participación en el desarrollo del software, logrando que se involucre más en el proceso.

1.8. Lenguaje de modelado

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. 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) (Qing, y otros, 2009).
Sus principales funciones son:

- Visualizar: Permite representar cualquier sistema de forma gráfica de manera que cualquier persona lo pueda entender
- Especificar: Facilita especificar las características de un sistema
- Construir: Mediante los modelos permite construir el sistema
- Documentar: Documenta los elementos gráficos lo que se puede utilizar como documentación, permitiendo una futura revisión

Para la realización de este trabajo se requiere de un lenguaje gráfico, con el objetivo de especificar el sistema de software de forma estándar. Después de estudiar las características del lenguaje UML se decide utilizarlo para el diseño de los artefactos que se generan en la metodología Mobile-D. Este lenguaje permite que todo software de diseño, se visualice, especifique y documente con un lenguaje de forma común. Además, cuenta con notaciones estándar y semánticas esenciales para el modelado de un sistema y se ajusta a la metodología seleccionada.

1.9. Herramienta CASE para la modelación del sistema

La Ingeniería de Software Asistida por Computadoras (CASE, por sus siglas en inglés, Computer Aided Software Engineering) es un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores durante todos el paso del ciclo de vida de desarrollo de un software. Este puede ser generalmente aplicado a cualquier sistema o colección de herramientas que ayudan a automatizar el proceso de diseño y desarrollo de software (Pressman, 2002).

Las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros:

- Las plataformas que soportan.
- Las fases del ciclo de vida del desarrollo de sistemas que cubren.
- La arquitectura de las aplicaciones que producen.

Una de las herramientas CASE considerada como muy completa y fácil de usar, con un soporte multiplataforma que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones es el Visual Paradigm.

1.9.1. Dia

Más que un programa de dibujo es una herramienta de modelado UML. A través de bibliotecas de símbolos y de un conjunto de herramientas permite la realización de diagramas de flujo, diagramas de redes, entre otros. Cuenta con buen soporte, una biblioteca surtida y la ventaja de la libertad de "movimientos". Además, la posición que tienen las interconexiones de los objetos y el control de la escala del dibujo lo hace muy manejable (López, 2012).

1.9.2. Rational Rose

Es la herramienta CASE que permite cubrir el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables. Posibilita el modelado visual mediante UML de sistemas de software. Facilita especificar, analizar y diseñar el sistema antes de codificarlo. Mantiene la consistencia de los modelos del sistema de software y chequeo de la sintaxis UML, la generación de la documentación automáticamente, así como la generación de código a partir de los modelos de ingeniería inversa (IBM, 2006)

1.9.3. Visual Paradigm

El Visual Paradigm es una herramienta UML de ingeniería de software que favorece el desarrollo de los programas informáticos desde su planificación hasta el análisis y diseño. Su propósito general es llevar el ciclo de vida completo del desarrollo de software ya sea: análisis y diseño, construcción, pruebas y despliegue a través de la forma de representación de todo tipo de diagramas.

Como herramienta CASE posee numerosas ventajas como:

- Apoya todo lo básico en cuanto a artefactos generados en las etapas de definición de requerimientos y de especificación de componentes.
- Tiene apoyo adicional en la generación de artefactos automáticos.
- Generación de código e ingeniería inversa: brinda la posibilidad de generar código a partir de los diagramas, para las plataformas como .Net, Java y PHP, así como obtener los diagramas a partir del código.
- Generación de documentación: brinda la posibilidad de documentar todo el trabajo sin necesidad de utilizar herramientas externas.

Visual Paradigm permitir la generación de los diseños centrados en casos de uso y enfocados al negocio, así como la seguridad de que el modelo y el código permanezcan sincronizados en todo el ciclo de desarrollo (Visual Paradigm International, 2015).

1.9.4. Selección de la herramienta

Por sus características, utilidades, posibilidades, incluyendo las ventajas antes expuestas, se elige **Visual Paradigm** como herramienta de modelado, pues cumple con la necesidad de confiabilidad para el desarrollo de la herramienta COJ Mobile. Además, brinda grandes facilidades para la creación de los diversos diagramas, y cuenta con la posibilidad de realizar un control de versiones durante el ciclo de trabajo.

1.10. Lenguaje de programación

Un lenguaje de programación es un lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones para su procesamiento por un sistema de cómputo. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de un determinado hardware, para expresar algoritmos con precisión o como modo de comunicación humana (Computer Hope, 2016).

1.10.1. Java

La principal característica de Java es la de ser un lenguaje compilado e interpretado, a la vez de ser de código abierto. Todo programa en Java ha de compilarse y el código generado es interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma. Java es un lenguaje orientado a objetos de propósito general (Java, 2015).

Su uso y facilidades para la creación de aplicaciones Android así como compatibilidad, documentación y el soporte brindado por Google, la hacen la solución más óptima para incorporar a la solución propuesta como parte de la línea de investigación que se lleva a cabo.

1.11. Gestor de base de datos

Un Sistema Gestor de Base de Datos SGBD (DBMS por sus siglas en inglés) es una colección de programas cuyo principal objetivo es servir de interfaz entre las bases de datos, el usuario y las aplicaciones. La elección de datos necesarios para el almacenamiento y búsquedas, ya sea de forma interactiva o a través de un lenguaje de programación. Este permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad y la integridad de los mismos (Alvarez, 2007).

Las principales características de un SGBD son:

- Abstracción de la información.
- Seguridad.
- Independencia.
- Integridad.
- Redundancia mínima.
- Respaldo y recuperación.
- Consistencia.
- Control de la concurrencia.

1.11.1. SQLite

Sistema completo de bases de datos que soporta múltiples usabilidades, con la característica de ser ágil pero robusto. Es un sistema gestor de bases de datos relacional compatible con Atomicidad, Consistencia, Aislamiento y Durabilidad ACID (por sus siglas en inglés Atomicity, Consistency, Isolation and Durability), comprendida en una biblioteca relativamente pequeña. Es una librería escrita en C que implementa un motor de bases de datos para SQL92.

Este gestor no es más que un sistema que soporta tablas, índices y vistas que no necesita de un servidor para su utilización. Es capaz de escribir y leer directamente sobre ficheros que se encuentran en el disco duro. Es multiplataforma e imparcialmente se puede utilizar archivos en sistemas de 32 y 64 bits.

La base de datos se almacena en un único fichero a diferencia de otros SGBD que hacen uso de varios archivos. SQLite emplea registros de tamaño variable de forma tal que se utiliza el espacio en disco que es realmente necesario en cada momento (SQLite, 2016).

Ventajas

- Tamaño: posee una pequeña memoria y una única biblioteca necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas.
- Rendimiento de base de datos: SQLite realiza operaciones de manera eficiente y es más rápido que MySQL y PostgreSQL.
- Portabilidad: se ejecuta en muchas plataformas y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración.

- SQL: implementa un gran subconjunto de la ANSI[3]–92 SQL estándar, incluyendo sub-consultas, generación de usuarios y vistas.

1.12. Entorno de desarrollo integrado

Un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés, Integrated Development Environment) es un programa informático compuesto por herramientas de programación que pueden ser partes de aplicaciones existentes. Estos sistemas pueden manejar uno o varios lenguajes de programación ejemplo Java, C# y C++. Usualmente un Entorno de Desarrollo Integrado incluye herramientas para la edición de código, así como depuradores, compiladores o constructores de interfaces gráficas (PC.net, 2016).

1.12.1. Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto y multiplataforma. Tiene herramientas para desarrollar aplicaciones de consola, Web y Web Services. Da soporte a todo tipo de proyectos abarcando el ciclo de vida completo en el desarrollo de aplicaciones. Dicho IDE ha alcanzado un alto grado de madurez, así como más robustez y rendimiento.

Eclipse a su vez no posee soporte nativo para desarrollo de aplicaciones para Android, pero hace uso de algunas herramientas que facilitan la creación de este tipo de aplicaciones. Se debe tener en cuenta que, al no ser un entorno de desarrollo creado solamente para Android, en ocasiones puede un poco difícil de trabajar (Eclipse, 2015).

1.12.2. Android Studio

Android Studio es un entorno de desarrollo integrado multiplataforma y desarrollado por Google con el único propósito facilitar a los desarrolladores de Android una plataforma estable y robusta.

Android Studio por su parte posee algunas ventajas sobre Eclipse que se deben tener en cuenta, como pueden ser (Cogswell, 2014):

- Facilidad de diseño, permitiendo arrastrar componentes e insertarlos en la aplicación, ahorrando

[3] Instituto Nacional Estadounidense de Estándares por sus siglas en inglés

tiempo en el desarrollo.

- Completamiento de código avanzado.
- Interfaz de usuario amigable, brindando facilidades para encontrar y usar las principales herramientas necesarias para el desarrollo.
- Integración con el repositorio Maven de librerías.
- Soporte directamente por Google.

1.12.3. Selección del Entorno de Desarrollo

Se selecciona el IDE Android Studio que resulta un entorno de desarrollo recomendable para trabajar con Android debido a que es la plataforma oficial para el desarrollo de este tipo de aplicaciones y es desarrollado por Google, brindando el soporte necesario para dicha herramienta.

1.13. Conclusiones parciales

A través del análisis de las principales características de la metodología, herramientas y tecnologías adoptadas, en su mayoría libres y de código abierto, enfocadas en las ventajas y facilidades de cada una de estas, se logra crear un marco de trabajo adecuado a las necesidades y características propias de una aplicación Android.

Mobile-D ha sido la metodología seleccionada como guía durante el proceso de desarrollo de software. Se utilizará Android Studio como IDE de desarrollo y Java como lenguaje de programación, así como la base de datos es creada haciendo uso de SQLite. Para facilitar la creación de los artefactos planteados por la metodología se selecciona la herramienta Visual Paradigm con el lenguaje de modelado UML.

Capítulo 2: Propuesta de solución

2.1. Introducción al capítulo

En este capítulo se realiza una descripción de la solución propuesta con el objetivo de proporcionar un mejor entendimiento del sistema. Además se representa el diagrama del Modelo del Dominio con la correspondiente descripción de cada uno de sus elementos. Por otra parte se especifican los requisitos funcionales y no funcionales que debe cumplir el sistema. Asimismo, se obtienen como artefactos fundamentales las historias de usuarios y sus descripciones.

2.2. Definición de alcance

COJ Mobile es un sistema enfocado en mejorar la experiencia de los usuarios del COJ desde las plataformas móviles con sistema operativo Android. Para este propósito se definen los siguientes requisitos que describen las funcionalidades de la aplicación. A cada requisito se le asigna un nivel de importancia dado entre 1 (Poco importante) y 5 (Muy importante).

Tabla 2: Requisitos funcionales

Características / historias		Importancia
1	Autenticar usuario	5
2	Des-autenticar usuario	5
3	Listar entradas	4
4	Crear nueva entrada	3
5	Mostrar perfil de usuario	3
6	Editar perfil de usuario	3
7	Listar problemas en 24 horas	5
8	Mostrar problema	5
9	Mostrar mejores soluciones de un problema	3
10	Enviar solución	2
11	Listar sentencias	4
12	Filtrar listado de sentencias	3

13	Ver posiciones por usuarios	4
14	Ver posiciones por países	4
15	Ver posiciones por instituciones	4
16	Comparar usuarios	4
17	Listar concursos próximos	5
18	Listar concursos actuales	5
19	Listar concursos anteriores	5
20	Ver descripción del concurso	5
21	Visualizar preguntas frecuentes	4
22	Recuperar contraseña	3
23	Abrir correo	4
24	Borrar correo	3
25	Escribir correo	4
26	Reenviar correo	3
27	Responder correo	3
28	Ver correos recibidos	5
29	Ver correos en borradores	4

Por otra parte la aplicación debe contar con algunas características que, sin ser funcionalidades del sistema principal, facilitan el uso de la misma. De esta forma se definen como requisitos no funcionales:

Seguridad:

La contraseña del usuario autenticado debe almacenarse utilizando algún método de encriptado.

La clave para encriptar la contraseña debe ser única en cada dispositivo.

Usabilidad:

El sistema deberá contar con una interfaz de fácil entendimiento para que usuarios inexpertos puedan interactuar fácilmente con el software.

El sistema podrá ser utilizado por cualquier usuario que posea una cuenta en el Juez en Línea Caribeño (COJ) así como aquel que aún no posea una cuenta.

El sistema se distribuirá en lenguaje inglés debido a que la mayor parte del COJ se encuentra en ese idioma.

Software

La instalación de la aplicación requiere una versión de Android 4.0.3 Ice Cream o superior.

2.3. Descripción del sistema

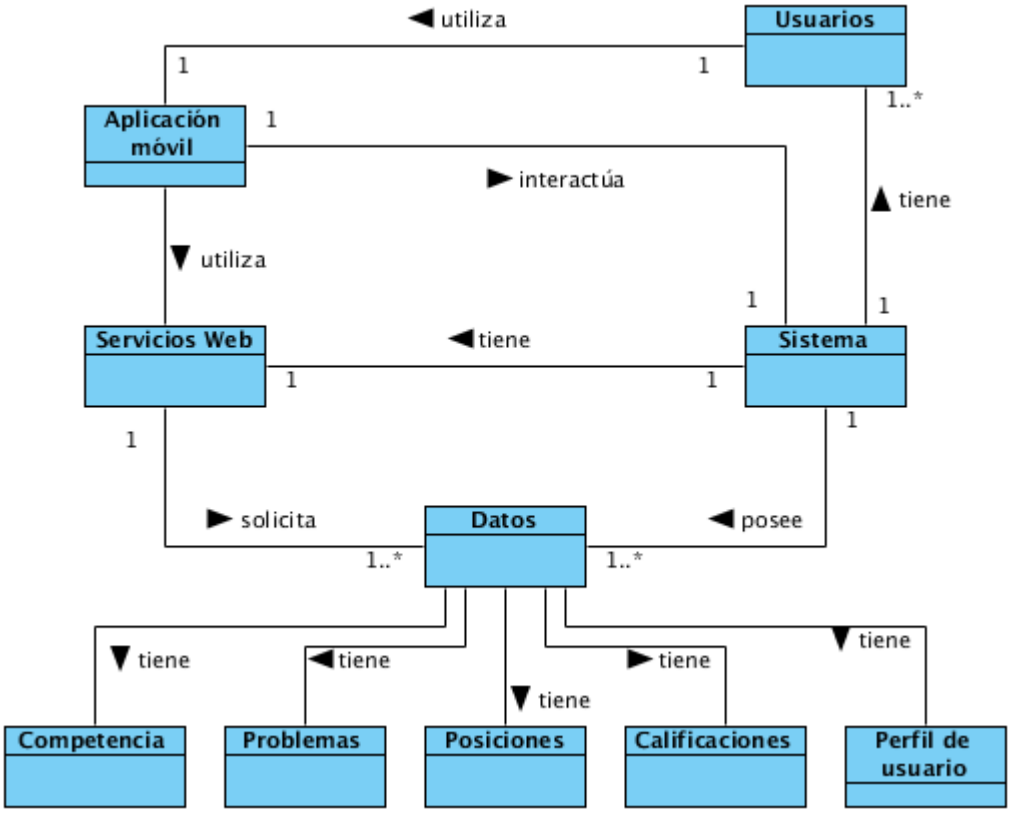
La aplicación que se propone tiene como objetivo interactuar con el COJ. La herramienta a desarrollar utiliza una capa de servicios web que garantiza la comunicación entre ambos sistemas. La misma permitirá el uso de las principales funcionalidades del COJ desde los teléfonos inteligentes que utilizan Android en la versión 4.0.3 Ice Cream o superior y cuenten con una conexión a internet ya sea WiFi o conexión de datos móviles.

COJ Mobile permite al usuario permanecer autenticado y recibiendo notificaciones constantes desde la plataforma del COJ acerca de nuevos eventos a desarrollar y correos entrantes. La herramienta es capaz de guardar en el dispositivo algunos datos seleccionados por el usuario para su posterior uso sin una conexión a internet activa.

2.4. Modelo de Dominio

El modelo de dominio es utilizado por el analista como un medio para comprender el sistema debido a que es una especie de representación esquemática de los conceptos y elementos de la vida real que serán usados en el mismo. Se crea con el fin de representar los conceptos claves del dominio del problema, las propiedades más importantes y las relaciones entre los conceptos, facilitando una mejor comunicación entre desarrolladores y clientes al establecer un lenguaje común para el entendimiento del mismo (Hans-Erik, y otros, 2000).

Fig. 3: Diagrama de clases del Modelo de Dominio



2.4.1. Definición de las clases del Modelo de Dominio

Sistema: Se refiere a la aplicación web del COJ.

Usuario: Todo aquel que tendrían acceso al sistema con el objetivo de interactuar con el mismo mediante la aplicación móvil.

Servicios Web: Es la herramienta que facilita la interoperabilidad entre el sistema y aplicaciones externas al mismo. Patrón arquitectónico

Aplicación móvil: Identifica a la aplicación para el sistema operativo Android que se encarga de solicitar los datos del sistema mediante el uso de los servicios web.

Datos: Es la información que posee el sistema. Puede estar dado por problemas, competencias, calificaciones, tabla de posiciones o información del perfil de usuario.

2.5. Patrón arquitectónico

Un patrón arquitectónico define la estructura básica de una aplicación, provee un subconjunto de subsistemas predefinidos, incluyendo reglas, lineamientos para conectarlos y pautas para su organización, además constituye una plantilla de construcción.

Entre las ventajas del uso de patrones, se pueden encontrar:

- Permiten la reutilización de soluciones arquitectónicas de calidad.
- Son de gran ayuda para controlar la complejidad de un diseño.
- Facilitan la documentación de diseños arquitectónicos.
- Proporcionan un vocabulario común que mejora la comunicación entre diseñadores.

Los patrones arquitectónicos expresan una organización estructural para un sistema, permiten estructurar los componentes y subsistemas de un sistema. La abstracción más alta en cuanto a soluciones a través de patrones se obtiene a través del uso de patrones de arquitectura.

2.5.1. Patrón arquitectónico N-Capas

Se propone el uso del patrón arquitectónico N-Capas para el desarrollo de la aplicación, ya que los entornos de ejecución de dicha aplicación estarán distribuidos en diferentes elementos o nodos, tanto lógicos como físicos. Aquí aparecen dos conceptos importantes: las capas que se encargan de la distribución lógica de los componentes, y los niveles que se refieren a la colocación física de los recursos. La característica principal de este patrón es que cada capa oculta las capas inferiores de las siguientes superiores a esta.

Algunas de las ventajas de utilizar este patrón son (Chininin, 2012):

- Mejoras en las posibilidades de mantenimiento, debido a que cada capa es independiente de la otra los cambios o actualizaciones pueden ser realizados sin afectar la aplicación como un todo.
- Escalabilidad, ya que las capas están basadas en diferentes máquinas, el escalamiento de la aplicación hacia afuera es razonablemente sencillo.
- Flexibilidad, como cada capa puede ser manejada y escalada de forma independiente, la flexibilidad se incrementa.

- Disponibilidad, las aplicaciones pueden aprovechar la arquitectura modular de los sistemas habilitados usando componentes que escalan fácilmente lo que incrementa la disponibilidad.

La descomposición en capas que se propone es:

Capa de presentación: es la única que interactúa directamente con el usuario presentándole el sistema, permitiendo el intercambio de información entre ambos. Contiene una interfaz gráfica que posibilita mostrar a los usuarios los resultados de sus peticiones y estados de la aplicación. Esta capa solo interactúa con la capa de negocio, que es su inferior inmediata.

Lógica de programación: es la que recibe las peticiones de la capa de presentación y le envía las respuestas tras el proceso. Aquí se realiza la mayor parte del procesamiento de la información del dominio de la aplicación, donde se ejecutan los algoritmos específicos del programa. Esta capa interactúa hacia arriba, con la capa de presentación, para recibir sus solicitudes y presentarle los resultados al usuario e interactúa hacia abajo haciendo solicitudes a la capa de red y a la capa de acceso de datos.

Capa de red: permite acceder a fuentes de datos externas a la aplicación y el dispositivo a través del uso de las redes de cómputo. Se realizan llamadas a servicios web en servidores de internet para obtener datos e imágenes necesarios.

Capa de acceso a datos: el código que permite acceder a las fuentes de datos. Esencialmente trata sobre 4 operaciones básicas, llamadas CRUD (por Create, Retrieve, Update y Delete), que se realizan sobre cualquier fuente de datos (normalmente alguna base de datos relacional).

Fig. 4: Diagrama de la arquitectura del proyecto

A continuación se describen los elementos de cada capa:

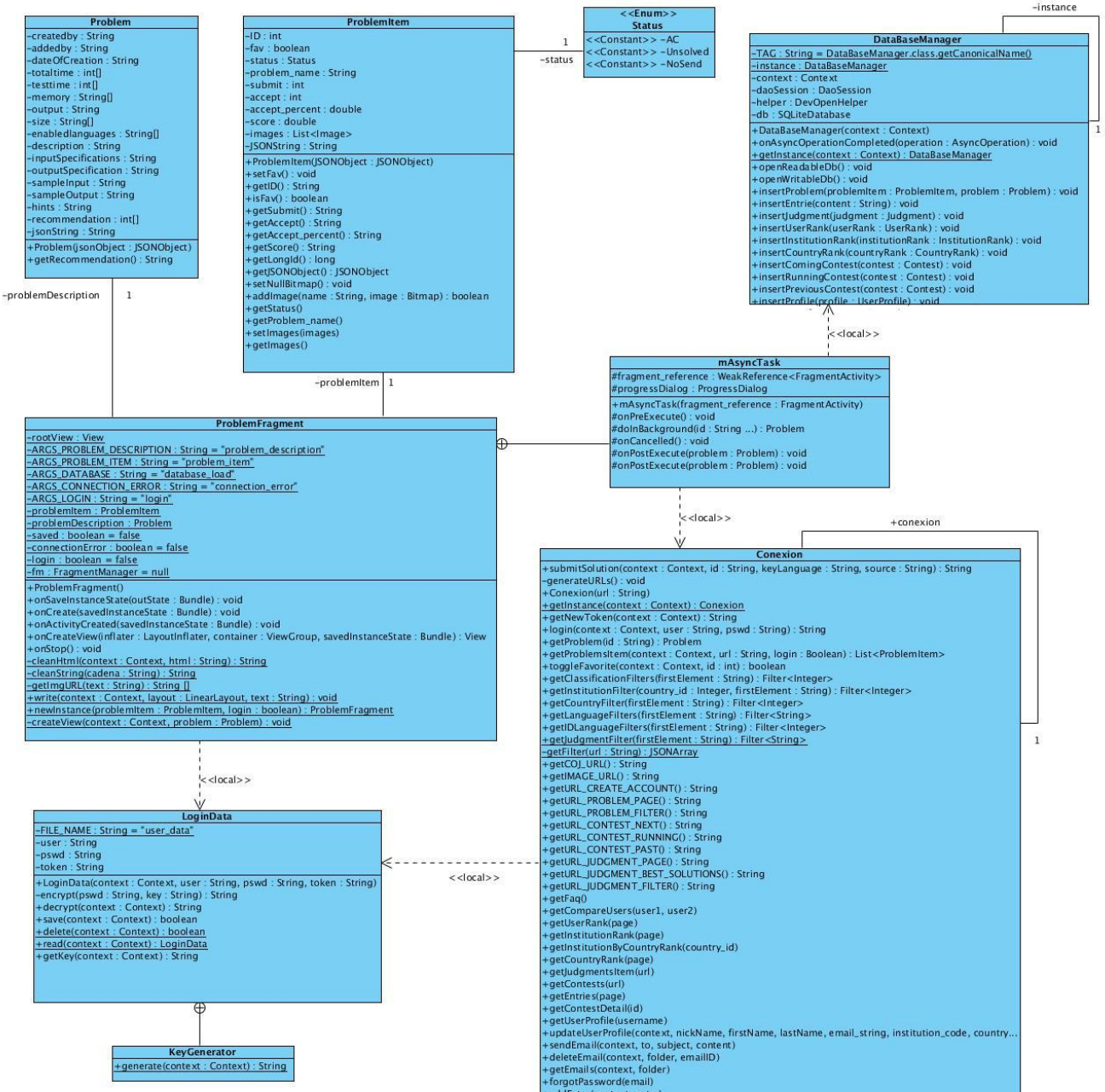
- Capa de presentación: En esta capa se encuentra la actividad Android COJ que contiene los componentes visuales que se muestran al usuario.
- Lógica de programación: En esta capa se representan todos los componentes lógicos del flujo de trabajo, o sea las clases Javas que se encargan de ejecutar las tareas propias de la aplicación.
- Capa de red: En esta capa se realizan todas las consultas necesarias al servidor del COJ mediante la utilización de servicios web.
- Capa de acceso de datos: En esta capa se crean las tablas necesarias para el almacenamiento local de los datos, así como también se realizan consultas a las mismas en caso que la capa de red no brinde una respuesta satisfactoria.

2.6. Diagrama de clases del diseño

Aquellas funcionalidades y características de cada clase en el lenguaje de desarrollo seleccionado, son representadas a través de los diagrama de clases del diseño (DCD). Estos proporcionan el objetivo y trabajo

de cada clase usada en la herramienta. El siguiente diagrama de clase del diseño contiene las principales clases con sus respectivos métodos y atributos para la historia de usuario Mostrar problema.

Fig. 5: DCD Mostrar problema



En este diagrama los elementos del modelo corresponden a las clases implicadas en la historia de usuario Mostrar problema, considerada significativa para el sistema.

Android basa su desempeño en el uso de clases Actividades o Fragmentos que poseen la responsabilidad de controlar aquellos formularios generados en estructura XML. En el caso particular de Mostrar problema se cuenta con el fragmento **ProblemFragment**, responsable de la captura de los componentes de la interfaz correspondiente en la Capa de presentación. Este fragmento posee una instancia de la clase **Problem** y otra de **ProblemItem**, las cuales contienen toda la información necesaria para mostrar la descripción de un problema. La obtención de los datos se hace mediante la clase **mAsyncTask**, que se encarga de realizar las peticiones a la capa de red o a la capa de datos, haciendo uso de las clases **Conexion** y **DataBaseManager**.

2.7. Patrones de diseño

Un patrón de diseño constituye un esquema para refinar subsistemas o componentes. Es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. Un patrón de diseño identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades, además de que ayuda a construir clases y a estructurar sistemas de clases. Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería de software, sino simplemente tratar de darle una representación a principios ya existentes que quizás ya han sido usado en los proyectos por un mero sentido de la lógica (Kioskea, 2014).

Un patrón de diseño es (Dias, 2014):

- Una solución estándar para un problema común de programación.
- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Un proyecto o estructura de implementación que logra una finalidad determinada.
- Un lenguaje de programación de alto nivel.
- Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- Conexiones entre componentes de programas.

- La forma de un diagrama de objeto o de un modelo de objeto.

Ventajas (Dias, 2014):

- Proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.
- Están basados en la recopilación del conocimiento de los expertos en desarrollo de software.
- Es una experiencia real, probada y que funciona.
- Facilitan la localización de los objetos que formarán el sistema, la determinación de la granularidad adecuada, el aprendizaje y la comunicación entre programadores.
- Especifican interfaces para las clases e implementaciones al menos parciales.

2.7.1. Patrones GRASP[4]

Los patrones GRASP son patrones generales de software para la asignación de responsabilidades a objetos. Describen los principios fundamentales del diseño de objetos para la asignación de responsabilidades y constituyen la base del cómo se diseñará el sistema. (Grosso, 2011).

Controlador

Es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Guía la asignación de responsabilidades relacionadas con la creación de objetos, permitiendo instanciar y crear las clases que le son necesarias para cumplir sus funcionalidades. En la aplicación se utiliza este patrón en la clase principal llamada MainActivity.java y en las clases que heredan de Fragment, ejemplo de estas son: ProblemFragment.java y ProfileFragment.java.

Creador

Asigna responsabilidades relacionadas con la creación de objetos o instanciación de nuevos objetos o clases. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Brinda un soporte a un bajo acoplamiento, lo que supone menos

[4] Responsabilidad de la Asignación General de Patrones de Software

dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Este patrón es utilizado en las clases de tipo adaptador, en las cuales se crean objetos de las clases ViewHolder para su manipulación. Ejemplos de clases de tipo adaptador son: ProblemList.java y JudgmenList.java.

2.7.2. Patrones GoF

El avance reciente más importante en el diseño orientado a objetos es probablemente el movimiento de los patrones de diseño, inicialmente narrado en "Design Patterns" (Patrones de Diseño), por Gamma, Helm, Johnson y Vlissides que suele llamarse el libro de la "Banda de los Cuatro" (en inglés, GoF: Gang of Four). Existen 23 patrones de diseño GoF que se clasifican en 3 categorías basadas en su propósito:

- **Creación:** Factoría Abstracta, Singleton (Creación de objetos).
- **Estructurales:** Adaptador, Constructor, Fachada (Composición de clases y objetos).
- **Comportamiento:** Observador, Estrategia (Modo en que las clases y objetos interactúan y se reparten las responsabilidades).

Singleton: asegura que una clase tiene exactamente una única instancia y que esta sea fácilmente accesible. Este es un patrón fácil de entender pero puede traer consecuencias no esperadas si es usado de forma abusiva. El patrón singleton es usado en la clase que controla la conexión a la base de datos llamada DataBaseManager.java.

Constructor: permite mantener de forma separada la creación de un objeto complejo y su representación. De esta forma es posible obtener diferentes representaciones de un mismo objeto mediante el mismo proceso de construcción. El patrón constructor se ve evidenciado en la clase ProblemsFragment.java en el momento en que se crea un nuevo dialogo para hacer uso de los filtros del COJ.

Adaptador (Adapter): convierte la interfaz de una clase en otra distinta que es la que esperan los usuarios. Permite que cooperen clases que de otra manera no podrían por tener interfaces incompatibles. Dentro de la plataforma móvil, puede emplearse para generar los elementos de componentes visuales como las listas expandible, que requieren de un adaptador para crear los grupos padres y los hijos, que han de ser mostrados al usuario. Este patrón se utiliza en la clase ProblemList.java.

Observador: define una dependencia de uno-a-muchos entre objetos. Cuando cambia el estado de un objeto entonces todas sus dependencias son notificadas y se actualizan automáticamente. Ejemplo de este patrón son las clases de comportamiento AppBarLayoutBehavior.java y FloatingActionButtonBehavior.java.

2.8. Lista de verificación del plan de proyecto

Finalmente, para lograr un control detallado acerca de las tareas desarrolladas a lo largo del proceso de desarrollo la metodología Mobile-d propone el uso de la siguiente lista de verificación del plan del proyecto.

Tabla 3: Lista de verificación del plan del proyecto

Lista de verificación del plan de proyecto			
Exploración			
Requerimientos iniciales	**Si**	**No**	**NA**
Todos los requerimientos funcionales iniciales han sido incluidos en el plan del proyecto	X		
Todos los requerimientos no funcionales iniciales han sido incluidos en el plan del proyecto	X		
Programación y ritmo	**Si**	**No**	**NA**
La programación general se ha incluido en el plan del proyecto	X		
El ritmo previsto (número y la duración de cada fase y sus iteraciones) han sido definidos en el plan del proyecto	X		
Recursos	**Si**	**No**	**NA**
El plan del proyecto ha sido actualizado con los grupos de interés identificados y sus miembros	X		
El plan del proyecto ha sido actualizado con la información acerca de las herramientas de desarrollo de software	X		
El plan del proyecto ha sido actualizado con los miembros del proyecto identificados	X		
Formación	**Si**	**No**	**NA**
Las necesidades de formación del equipo de desarrollo ha sido incluidas en el plan del proyecto			X

La programación de la formación del equipo de desarrollo ha sido incluida en el plan del proyecto			X
Documentación	**Si**	**No**	**NA**
Los documentos que se generarán en el proyecto han sido incluidos en el plan del proyecto (Ej. Arquitectura y documentación de las interfaces de usuarios)		X	
La vida útil de cada documento se ha incluido en el plan del proyecto		X	
Aseguramiento de calidad	**Si**	**No**	**NA**
Los procedimientos del aseguramiento de la calidad han sido definidos en el plan del proyecto para cada producto del trabajo, incluyendo los actores y el cronograma.	X		
Iteración 0			
Requerimientos	**Si**	**No**	**NA**
El plan del proyecto ha sido actualizado teniendo en cuenta los requerimientos seleccionados para la prueba de la iteración 0	X		
El plan del proyecto ha sido actualizado teniendo en cuenta la realización de los requerimientos de prueba seleccionados en la iteración 0	X		
La línea de arquitectura ha sido definida e incluida en el plan del proyecto	X		
Iteración(es) de producción			
Requerimientos	**Si**	**No**	**NA**
El plan del proyecto ha sido actualizado teniendo en cuenta los requerimientos seleccionados para la actual iteración	X		
El plan del proyecto ha sido actualizado teniendo en cuenta la realización de los requerimientos seleccionados para la actual iteración	X		
El plan del proyecto ha sido actualizado teniendo en cuenta los cambios en la planificación, ritmo, requerimientos y recursos	X		
El plan del proyecto ha sido actualizado teniendo en cuenta la realización de las actividades de aseguramiento de la calidad en la	X		

iteración actual			

2.9. Puesta en marcha

En la puesta en marcha del proyecto es necesario instalar, actualizar y preparar los recursos técnicos necesarios para el desarrollo. Además, se da el entrenamiento necesario al equipo de desarrollo en caso que posea poca experiencia con las herramientas a trabajar.

En el caso del desarrollo del sistema propuesto, fue necesario instalar la herramienta de modelado Visual Paradigm y el IDE Android Studio, así como actualizar el marco de trabajo de la SDK. Por otra parte, no se realizó ningún tipo de capacitación, debido a que las herramientas y tecnologías utilizadas son conocidas con anterioridad por el equipo de desarrollo.

2.10. Planificación inicial

Para un mayor entendimiento de las características de la aplicación, cada requisito funcional se presenta a través de historias de usuarios y se crean las interfaces de usuario necesarias.

2.10.1. Interfaz de usuario

Después de definir detalladamente cada una de las características que debe cumplir el sistema se procede a la creación de las interfaces de usuario. La interfaz es la vía que usa el usuario para comunicarse con la aplicación. Conocida también como UI del inglés User Interface incluye toda clase de componentes visuales que facilitan la interacción usuario/aplicación.

Fig. 6: Interfaz de la historia 17

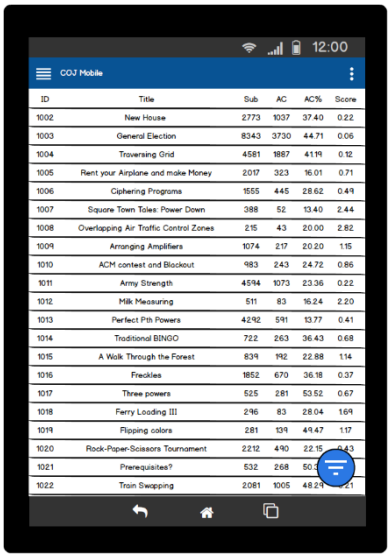


Fig. 7: Interfaz de la historia 18

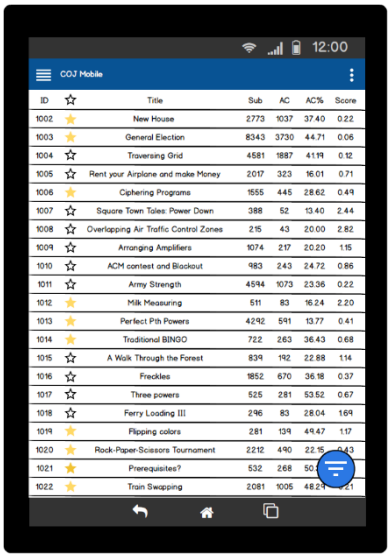


Fig. 8: Interfaz de la historia 19

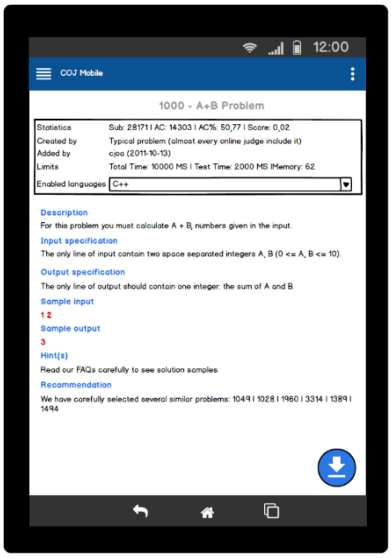
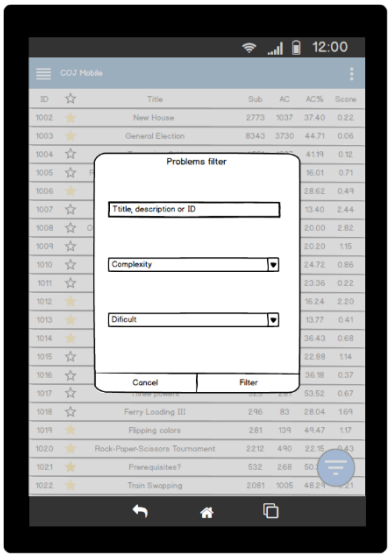


Fig. 9: Interfaz de la historia 20



2.10.2. Tarjeta de historia

Con el objetivo de probar que todas las herramientas estén funcionales y listas para el desarrollo del sistema, así como que el equipo de desarrollo tenga el conocimiento necesario, se selecciona una de las funcionalidades del sistema para implementarla.

Tabla 4: Funcionalidad seleccionada para desarrollar en el día de trabajo

Características / historias		Importancia
7	Listar problemas en 24 horas	5

Finalmente, la siguiente tabla presenta la historia definida para el desarrollo. Este documento es definido durante el día de trabajo y redefinido durante la implementación.

Tabla 5: Historia seleccionada para implementación

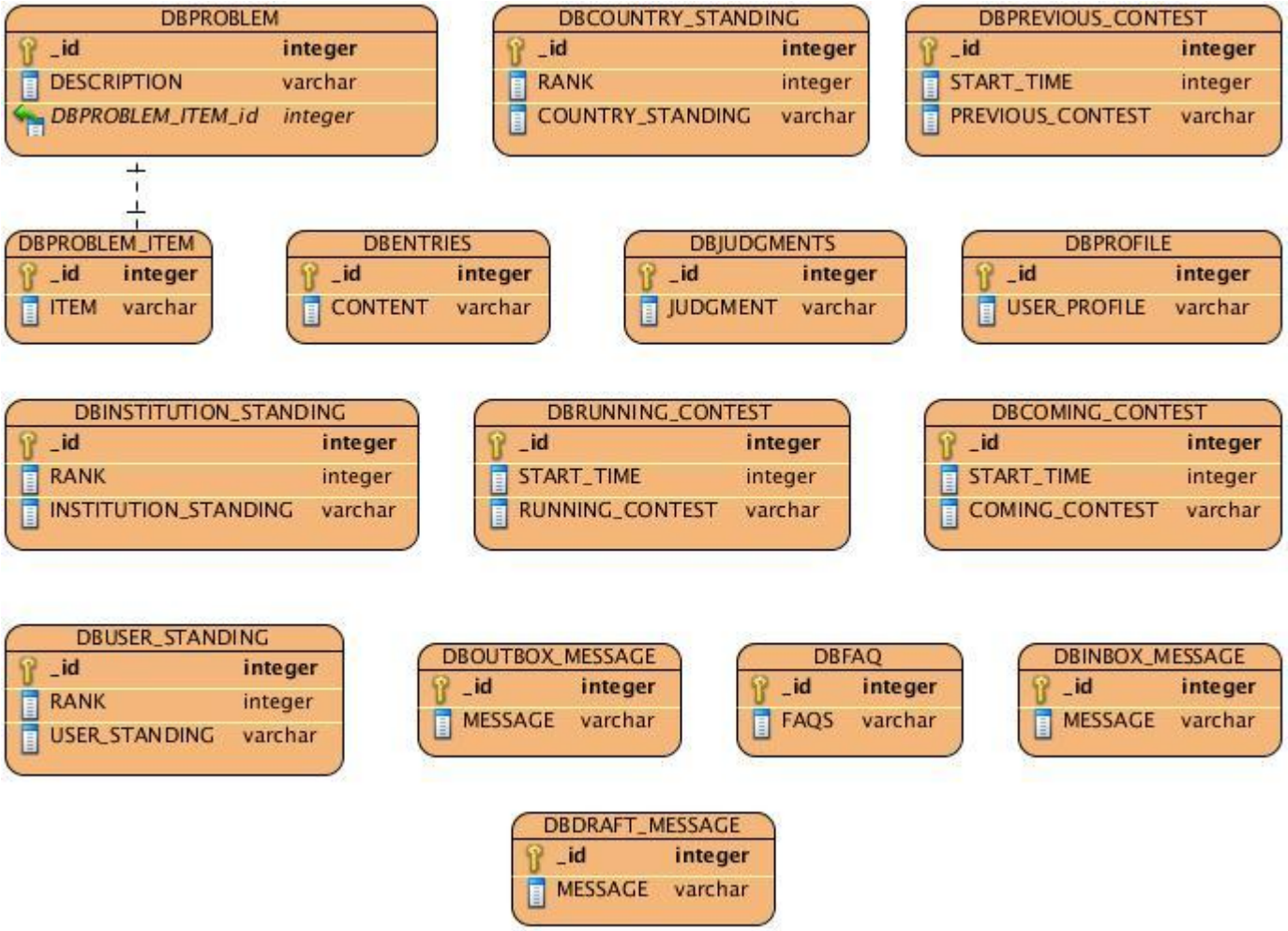
Número/ID	Tipo	Dificultad		Tiempo		Prioridad
		Antes	**Después**	**Estimado**	**Gastado**	
17	Nuevo	A	A	2	1.5	5

Notas		
Se debe ocultar/mostrar la columna favorito dependiendo del tipo de usuario		
Descripción		
Permitir visualizar todos los problemas existentes en el archivo 24 horas. Para listar los problemas el usuario debe seleccionar el vínculo problemas en el bloque Archivo 24 horas, el sistema mostrará el listados de los problemas pudiendo hacer un filtro por: título o descripción; muestra una tabla con el id, estado, título, cantidad de envíos, aceptados, porciento de aceptados y puntuación; y permite ver la descripción de un problema dando clic sobre el mismo.		
Fecha	**Estado**	**Comentario**
12.11.2015	Definido	Esta historia se ha tomado para implementarla durante el día de prueba en la tercera etapa de la fase de inicialización
13.11.2015	Implementado	Lograr ocultar y mostrar la columna de favorito fue algo complicado inicialmente
15.11.2015	Realizado	Se crearon las distintas pruebas de aceptación y las posibles respuestas del sistema
15.11.2015	Verificado	Todas las pruebas resultaron satisfactorias

2.11. Modelo de datos

Un modelo de datos es una estructura abstracta que documenta y organiza la información para la comunicación. En la informática, se centra en el planeamiento del desarrollo de aplicaciones y la decisión de cómo se almacenarán los datos y cómo se accederá a ellos. El modelo relacional se caracteriza a muy grandes rasgos por disponer que toda la información debe estar contenida en tablas, y las relaciones entre datos deben ser representadas explícitamente en esos mismos datos (Chávez, y otros, 2002).

Fig. 10: Diagrama Entidad-Relación



2.12. Modelo de despliegue

El modelo de despliegue define la arquitectura física del sistema por medio de nodos interconectados. Estos nodos son elementos de hardware sobre los cuales pueden ejecutarse elementos de software. Se utiliza como entrada principal en las actividades de diseño e implementación, debido a que la distribución del sistema tiene una influencia principal en su diseño (Rumbaught, et al., 2000).

2.12.1. Diagrama de despliegue

El diagrama de despliegue es utilizado para mostrar los nodos y conexiones del modelo de despliegue así como la asignación de los objetos a los nodos (Rumbaught, et al., 2000).

El diagrama que se presenta a continuación representa la distribución física del sistema desarrollado a través de nodos, está compuesto por una dispositivo cliente que deberá tener instalado el sistema operativo Android y la aplicación “COJ Mobile”, donde la comunicación entre ella y el servidor del COJ se llevará a cabo haciendo uso de la capa de servicios web que posee el mismo a través del Protocolo de Transferencia de Hipertexto (HTTP por sus siglas en inglés).

Fig. 11: Diagrama de despliegue

2.13. Conclusiones parciales

En el capítulo se logró un mejor entendimiento del entorno real de la herramienta a través de la realización del modelo de dominio. Se determinaron las diferentes funcionalidades y cualidades que el sistema debe cumplir mediante la descripción de las funcionalidades del sistema. Para evitar problemas en el diseño del software, se aplicaron los patrones correspondientes. Para definir la estructura general se aplicó el patrón arquitectónico N-Capas. El modelo de datos relacional especificó las clases persistentes de la base de datos, y finalmente, con la confección del diagrama de despliegue se logró modelar la distribución física de la herramienta.

Capítulo 3: Implementación y pruebas

3.1. Introducción al capítulo

En el capítulo anterior se describió la propuesta del sistema y se presentaron los artefactos generados que facilitan al desarrollador entender las funcionalidades que exige el cliente y garantizar la implementación exitosa. Igual de importante para un sistema es el proceso de pruebas, que tiene como objetivo determinar si el software se ajusta a los requisitos o las condiciones impuestas por el cliente.

La metodología Mobile-D gestiona los procesos de implementación y prueba con pocos artefactos, de forma tal que la comunicación entre los miembros del equipo y el cliente, sea la base para la obtención del producto según sus necesidades. En el presente capítulo se documentan las fases de implementación y prueba según lo propone la metodología seleccionada para el desarrollo. Se desglosan las historias en tareas con el objetivo de facilitar el trabajo de los desarrolladores y se muestran los resultados de las pruebas de aceptación realizadas a la aplicación.

3.2. Estabilización

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

Debido a que el proyecto actual no se encuentra dividido en subsistemas, entonces no es necesaria la realización de actividades de integración.

3.3. Implementación

3.3.1. Día de planificación

El propósito del día de planificación es seleccionar y planear el contenido de trabajo para la iteración en curso. Mediante la participación activa del cliente se asegura que el equipo de desarrollo entienda correctamente las funcionalidades del sistema.

En esta etapa es necesaria la elaboración de cada una de las tareas a desarrollar para el cumplimiento de la historia seleccionada para la iteración en curso. Además, es necesario que el equipo de desarrollo cuente con los prototipos de interfaces a desarrollar.

3.3.2. Día de trabajo

El propósito de esta etapa es la implementación de la funcionalidad del sistema seleccionada en el día de planificación. El equipo de desarrollo se enfoca en las funcionalidades de mayor prioridad definidas por el cliente.

De ser posible Mobile-D propone para esta etapa el desarrollo en par con el objetivo de asegurar comunicación entre el equipo de desarrollo, entendimiento del proceso y lograr mayor calidad y entendimiento del código.

Al finalizar cada día de trabajo es necesario propiciar al cliente una vista del progreso alcanzado y así dar la posibilidad al mismo de nuevas sugerencias para las funcionalidades a implementar.

3.3.1. Día de liberación

Para esta etapa del desarrollo es necesario crear una liberación totalmente funcional del sistema. Para lograr el objetivo planteado anteriormente la primera tarea a realizar es verificar que el software está listo para realizar las pruebas de aceptación.

Una vez realizadas las pruebas de aceptación se realiza una nueva iteración en caso de ser necesario corregir alguna funcionalidad y de no ser necesario, el software está listo para la liberación.

3.4. Tareas de ingeniería

Para llevar a cabo la correcta implementación de las historias, se deben definir por parte del equipo de desarrollo las TI (Tareas de Ingeniería) que se realizarán en cada una de las iteraciones. Las TI, también conocidas como tareas de implementación, permiten a los desarrolladores obtener un nivel de detalle más avanzado que el que propicia las historias. A continuación se muestran las tareas definidas para el desarrollo de la historia “Listar problemas del COJ”.

Tabla 6: Tarea "Solicitar listado de problemas"

Número/ID	Tipo	Dificultad		Historia
		Antes	**Después**	
22	Nuevo	M	M	17
Notas				
Descripción				
Teniendo la descripción del servicio web que brinda el listado de problemas es necesario realizar una conexión satisfactoria al servidor.				
Fecha	**Estado**	**Comentario**		
12.11.2015	Definido	Esta tarea se ha tomado para implementarla durante el día de prueba en la tercera etapa de la fase de inicialización		
13.11.2015	Implementado	El tipo de servicios web REST y el trabajo con JSONs facilita en gran medida la consulta		
15.11.2015	Realizado	Se crean las pruebas de aceptación		
15.11.2015	Verificado	Las pruebas resultan satisfactorias		

Tabla 7: Tarea "Crear la interfaz gráfica del listado de problemas"

Número/ID	Tipo	Dificultad		Historia
		Antes	**Después**	
23	Nuevo	A	M	17
Notas				

Descripción		
Una vez obtenido el listado de problemas es necesario crear la interfaz gráfica que se muestra al usuario con los datos correspondientes		
Fecha	**Estado**	**Comentario**
12.11.2015	Definido	Esta tarea se ha tomado para implementarla durante el día de prueba en la tercera etapa de la fase de inicialización
14.11.2015	Implementado	El trabajo con la clase ProblemList.java de tipo adaptador permite la reutilización del código y se logra fácilmente crear la interfaz gráfica
15.11.2015	Realizado	Se crean las pruebas de aceptación
15.11.2015	Verificado	Las pruebas resultan satisfactorias

Tabla 8: Tarea "Crear comportamiento del botón flotante"

Número/ID	Tipo	Dificultad		Historia
		Antes	**Después**	
24	Nuevo	B	B	17
Notas				
Descripción				
Es necesario que el botón flotante se oculte al desplazar la lista hacia abajo y se muestre al desplazar la lista hacia arriba.				
Fecha	**Estado**	**Comentario**		
12.11.2015	Definido	Esta tarea se ha tomado para implementarla durante el día de prueba en la tercera etapa de la fase de inicialización		
15.11.2015	Implementado			
15.11.2015	Realizado	Se crearon las distintas pruebas de aceptación y las posibles respuestas del sistema		
15.11.2015	Verificado	Todas las pruebas resultaron satisfactorias		

3.5. Estándares de nomenclatura y codificación utilizados

Los estándares de codificación son un conjunto de directrices, normas y reglamentos enfocados a la especificación de cómo debe escribirse el código fuente de la aplicación. Estos incluyen pautas sobre la nomenclatura de las variables, clases y paquetes; la correcta identación del código, cómo escribir estructuras de control, entre otros aspectos. La correcta elaboración y utilización de los estándares de codificación permiten que todos los desarrolladores implementen siguiendo las mismas pautas y así puedan entender el código del resto como si estuvieran mirando el de ellos, de esta manera la aplicación será más legible, uniforme y fácil de mantener (Vermeulen, y otros, 2001).

3.5.1. Nomenclatura en general

- El nombre de todas las variables y métodos comenzarán con letra minúscula y si este está compuesto por varias palabras se utilizará el estilo de escritura lowerCamelCase, que dicta que para un nombre compuesto por varias palabras comenzará con minúscula pero todas las palabras internas que lo componen comienzan con mayúscula.

Paquetes

- Por defecto todos los paquetes se escribirán en minúsculas y sin utilizar caracteres especiales. El paquete base queda definido como cu.uci.coj.

Identación

- En el contenido siempre se identará este con tabulaciones, nunca utilizando espacios en blanco.

Clases

- El nombre de las clases comenzará con mayúsculas y cada salto de palabras debe iniciar con mayúsculas.

Métodos

- Los métodos deberán ser con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas (lowerCamelCase).

- No se permiten caracteres especiales.

Nombre de variables

- El nombre de las variables debe empezar con letra minúsculas y de existir un salto de palabra comenzaría con mayúscula.

3.5.2. Estilos de codificación

Comentarios

- Como norma es obligatorio proporcionar un comentario de documentación por cada clase y método creado.
- Comentario de la clase / método:
  - Prescripción genérica de la clase o método y su responsabilidad.
- Reglas generales a la hora de escribir comentarios de documentación.
  - Siempre se escribe en tercera persona.
  - Las descripciones siempre deberían empezar por un verbo.

Sentencias

- Una sentencia por línea de código.
- Todo bloque de sentencias entre llaves, aunque sea una sola sentencia después de un if.

Ejemplo de codificación:

Clases y atributos:

```
public class Conexion {

    public static Conexion conexion = null;

    …
}
```

Métodos y comentarios:

```
    /**
```

```
 * Get valid language filters and theirs equivalents key
 *
 * @param firstElement String with a default value for spinner
 *
 * @return Filter object. A filter contain Language+Description and Integer
 * Key pairs. Pair example: <"C++ (g++ 4.8.2)":"1">
 * @throws IOException
 * @throws JSONException
 */
public Filter<Integer> getIDLanguageFilters(String firstElement) throws
                IOException, JSONException {

    Filter<Integer> languageFilter = new Filter<>(firstElement);

    JSONArray jsonArray = getFilter(URL_FILTER_LANGUAGE);

    for (int i = 0; i < jsonArray.length(); i++) {
        String name = jsonArray.getJSONObject(i).getString("description");
        int key = jsonArray.getJSONObject(i).getInt("id");
        languageFilter.addFilter(name, key);
    }

    return languageFilter;

}
```

3.6. Pruebas de software aplicadas

Al realizar la planificación de la iteración es necesario definir también el sistema de pruebas que se aplicarán, para verificar que el sistema cumpla con las funcionalidades que precisa el cliente. Las pruebas aplicadas fueron: pruebas de aceptación, unitarias y funcionales

3.6.1. Pruebas unitarias

Según (Rojas, 2012) una prueba unitaria es un método que verifica una unidad de código. Al hablar de una unidad de código se refiere a un requerimiento. Muchos desarrolladores tienen su propio concepto de lo que es una prueba unitaria; sin embargo, la gran mayoría coincide en que una prueba unitaria tiene las siguientes características:

- Prueba solamente pequeñas cantidades de código: Solamente prueba el código del requerimiento específico.

- Se aísla de otro código y de otros desarrolladores: Las pruebas unitarias verifican exclusivamente el código relacionado con el requerimiento y no interfiere con el trabajo hecho por otros desarrolladores.
- Solamente se prueban los métodos públicos: Esto principalmente porque los disparadores de los métodos privados son métodos públicos por lo tanto se abarca el código de los métodos privados dentro de las pruebas.
- Los resultados son automatizados: Cuando ejecutamos las pruebas lo podemos hacer de forma individual o de forma grupal. Estas pruebas las hace el motor de prueba y los resultados de los mismos deben de ser precisos con respecto a cada prueba unitaria desarrollada
- Repetible y predecible: No importa el orden y las veces que se repita la prueba, el resultado siempre debe de ser el mismo.
- Son rápidos de desarrollar: Contrariamente a lo que piensan los desarrolladores –que el desarrollo de pruebas unitarias quita tiempo– este tipo de pruebas, por lo general deben de ser simples y rápidos de desarrollar. Difícilmente una prueba unitaria deba de tomar más de cinco minutos en su desarrollo.

Ejemplo de prueba unitaria:

```
public void testGetProblem(){

    Conexion conexion = Conexion.getInstance(activity);

    Problem problem = null;
    try {
        //Correct test case
        problem = conexion.getProblem("1000");
    } catch (IOException | JSONException e) {
        e.printStackTrace();
    }

    assertTrue(problem != null);
    assertEquals(problem.getEnabledlanguages().length,
            problem.getMemory().length);
    assertEquals(problem.getEnabledlanguages().length,
            problem.getTesttime().length);
    assertEquals(problem.getEnabledlanguages().length,
            problem.getTotaltime().length);

    String message = "";
    try {
```

```
            //Wrong test case
            conexion.getProblem("-1");
        } catch (IOException | JSONException e) {
            e.printStackTrace();
            message = e.getMessage();
        }

        assertEquals("Unexpected error: bad pid", message);

    }
```

3.6.2. Resultado de las pruebas

Durante la etapa de desarrollo de la aplicación se realizaron un conjunto de pruebas, utilizando el método de caja negra que se concentra en los requisitos funcionales del software y de caja blanca para probar el funcionamiento del código. Tienen como objetivo evaluar el correcto desempeño de las funcionalidades propuestas.

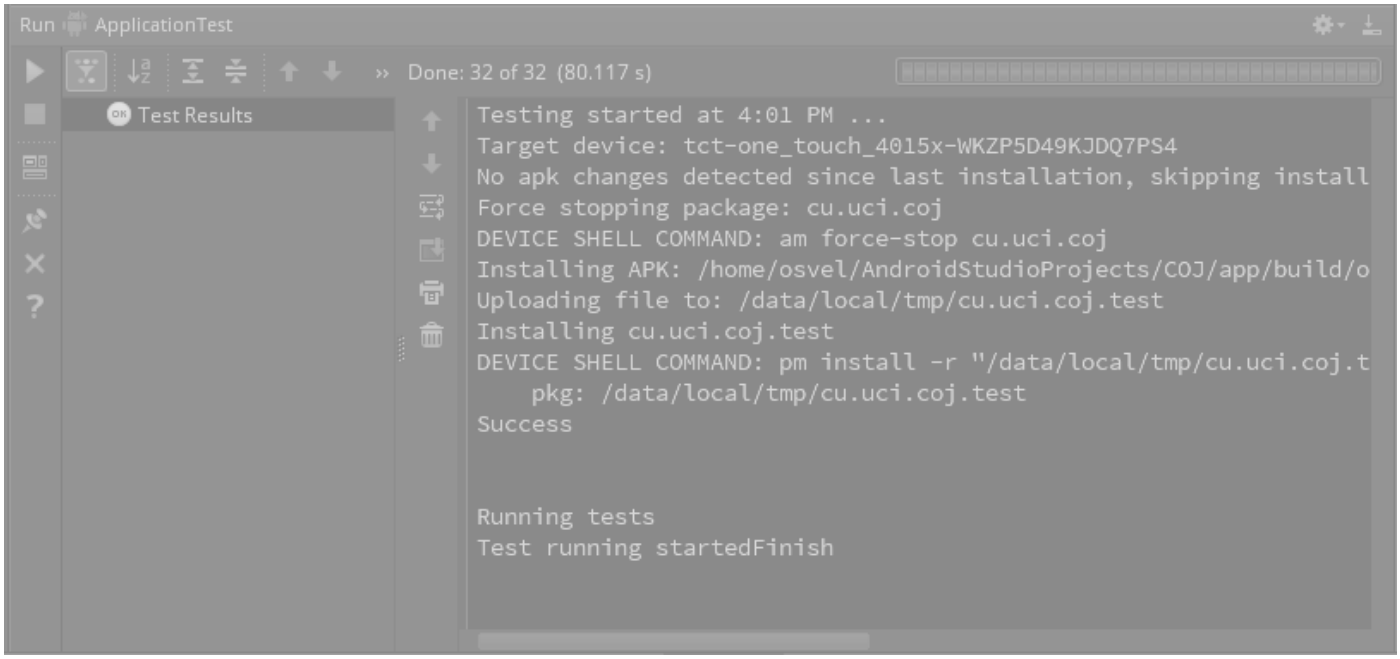
Las pruebas realizadas se muestran a continuación:

- **Pruebas unitarias:** Para probar el correcto funcionamiento del código. Conocer el tiempo y memoria consumida al realizar la prueba, aunque estos resultados pueden variar de un dispositivo a otro y de una ejecución a otra.
- **Pruebas funcionales:** Encargadas de que los resultados esperados ocurran cuando se usen datos válidos y que sean desplegados los mensajes apropiados de error y precaución en caso de datos inválidos.

A continuación se presentan los resultados arrojados por las diferentes pruebas aplicadas:

Pruebas unitarias: Las pruebas unitarias fueron desarrolladas constantemente cada vez que se terminaba de implementar alguna funcionalidad probándola directamente en el entorno real. Dichas pruebas se desarrollan utilizando la extensión Android JUnit, que es una parte integral del SDK de Android. Permite probar componentes específicos mediante clases de casos de pruebas. Estas clases proporcionan métodos auxiliares para la creación de objetos de imitación y métodos que ayudan a controlar el ciclo de vida de un componente. A continuación se muestra una imagen con los resultados obtenidos al realizar las pruebas unitarias.

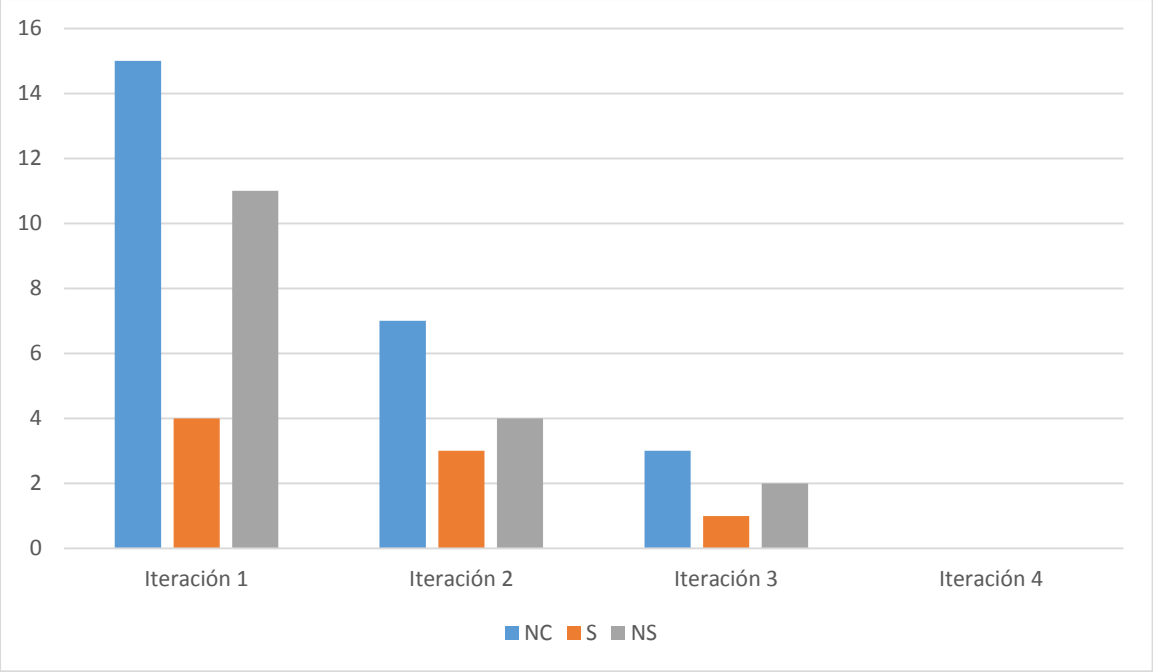
Fig. 12: Resultado de las pruebas unitarias



Pruebas funcionales: se realizaron cuatro iteraciones.

- Durante la primera iteración se encontraron 15 No Conformidades (NC), de ellas 4 Significativas (S), 11 No Significativas (NS).
- Durante la segunda iteración se encontraron 7 NC, de ellas 3 S y 4 NS.
- Durante la tercera iteración se encontraron 3 NC, de ellas 1 S y 2 NS.
- Durante la cuarta iteración no se encontraron no conformidades.

Fig. 13: Resultados de las pruebas funcionales



3.7. Conclusiones parciales

En este capítulo se llevó a cabo la fase de Implementación y Prueba planteada por la metodología Mobile-D. Se realizaron las tareas de ingeniería, logrando una mayor organización y rapidez en el desarrollo del sistema.

Se realizaron pruebas de aceptación, funcionales y unitarias al sistema. Las pruebas funcionales se realizaron en cuatro etapas. Las NC fueron resueltas al concluir las pruebas en cada iteración. Cada una de las pruebas unitarias se realizó para datos correctos e incorrectos, en ambos casos los resultados estuvieron acorde a lo esperado. Esto permite afirmar que el sistema está listo para ser utilizado en un entorno real, haciendo uso de las características del COJ desde los dispositivos móviles con SO Android.

Conclusiones

El desarrollo de este trabajo permitió arribar a las siguientes conclusiones:

- El estudio y análisis de los principales estándares de desarrollo de aplicaciones para Android y el trabajo con jueces en línea garantizó una base metodológica sobre los principales elementos que deben incluirse al interactuar con el COJ.
- Al evaluarse las características del COJ se obtuvo un modelo guía para la implementación del sistema mediante la generación de los artefactos correspondientes a los flujos de trabajo propuestos por la metodología Mobile-D.
- La implementación de las funcionalidades de la herramienta, acorde a las pautas de diseño, garantizó el cumplimiento de lo establecido en los objetivos específicos de la investigación.
- Se obtuvo una aplicación móvil desarrollada con tecnologías libres.
- Se demostró a partir de las diferentes iteraciones de pruebas practicadas al software, que este satisface los requisitos funcionales y no funcionales obtenidos durante el flujo de trabajo Modelado.
- El desarrollo de la aplicación permitirá expandir el alcance del COJ llevándolo a los usuarios de dispositivos móviles con SO Android.

Recomendaciones

Para perfeccionar y ampliar las funcionalidades de la solución propuesta, se realizan las siguientes recomendaciones:

- Actualizar la aplicación a medida que se realizan actualizaciones del COJ
- Continuar incluyendo en la aplicación las funcionalidades que brinda el COJ, que actualmente no se hayan incluido.

Referencias Bibliográficas

Agrawal, Harsh. 2015. Shoutmeloud. [En línea] 8 de 10 de 2015. [Citado el: 23 de 1 de 2016.] http://www.shoutmeloud.com/top-mobile-os-overview.html.

Alvarez, Sara. 2007. desarrolloweb. [En línea] 31 de 7 de 2007. [Citado el: 28 de 4 de 2016.] http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

Bathelot, Bertrand. 2012. The digital marketing glossary. [En línea] 15 de 10 de 2012. [Citado el: 4 de 2 de 2016.] http://digitalmarketing-glossary.com/What-is-Android-SDK-definition.

Chininin, Zorem. 2012. Guía de Arquitectura en N-Capas. [En línea] 12 de 10 de 2012. [Citado el: 26 de 4 de 2016.] http://arquitecturancapas.blogspot.com/.

Cogswell, Jeff. 2014. Dice. [En línea] 19 de 3 de 2014. [Citado el: 4 de 2 de 2016.] http://insights.dice.com/2014/03/19/googles-android-studio-vs-eclipse-fits-needs/.

Computer Hope. 2016. Computer Hope. *Free computer help and information.* [En línea] 2016. [Citado el: 4 de 2 de 2016.] http://www.computerhope.com/jargon/p/proglang.htm.

Cubadebate. 2016. Cubadebate. *En 2020, más personas con móviles que con electricidad y agua.* [En línea] 8 de 2 de 2016. [Citado el: 9 de 2 de 2016.] http://www.cubadebate.cu/noticias/2016/02/08/2020-mas-personas-con-moviles-que-con-electricidad-y-agua/.

Daichendt, Linda. 2016. Strategic Concepts. [En línea] 2016. http://www.strategicgrowthconcepts.com/growth/increase-productivity--profitability/mobile-technology-facts.html.

Dias, Carlos. 2014. Academia. [En línea] 9 de 2 de 2014. [Citado el: 26 de 4 de 2016.] http://www.academia.edu/8171370/CONCEPTOS_Y_VENTAJAS_DE_LOS_PATRONES_DE_DISE%C3%91O.

Eclipse. 2015. Eclipse. [En línea] 30 de 1 de 2015. [Citado el: 5 de 2 de 2016.] https://www.eclipse.org/eclipse/news/4.5/M5/.

González, Yaditza del Sol. 2016. Cubadebate. [En línea] 5 de 2 de 2016. [Citado el: 9 de 2 de 2016.] http://www.cubadebate.cu/noticias/2016/02/05/cuba-cerro-el-2015-con-mas-de-tres-millones-de-lineas-moviles/.

Google. 2015. Android Developers. [En línea] 12 de 2015. [Citado el: 4 de 2 de 2016.] http://developer.android.com/tools/help/sdk-manager.html.

—. 2015. Android Developers. [En línea] 12 de 2015. [Citado el: 26 de 1 de 2016.] http://developer.android.com/intl/es/about/dashboards/index.html.

Granados, Óscar. 2015. El País. [En línea] 29 de 8 de 2015. [Citado el: 9 de 2 de 2016.] http://economia.elpais.com/economia/2015/08/27/actualidad/1440698867_622525.html.

Grosso, Andrés. 2011. Prácticas de software. [En línea] 21 de 3 de 2011. [Citado el: 26 de 4 de 2016.] http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

Hans-Erik, Eriksson y Penker, Magnus. 2000. *Business Modeling with UML:Business Patterns at Work.* Canadá : John Wiley & Sons, 2000. 0471295515.

IBM. 2011. IBM Bluemix. [En línea] 5 de 8 de 2011. [Citado el: 28 de 4 de 2016.] https://www.ibm.com/developerworks/ssa/webservices/tutorials/ws-understand-web-services1/.

Inflectra. 2014. inflectra. [En línea] 2014. [Citado el: 3 de 2 de 2016.] https://www.inflectra.com/Methodologies/Agile-Development.aspx.

Infobae. 2016. Infobae America. [En línea] 23 de 1 de 2016. [Citado el: 9 de 2 de 2016.] http://www.infobae.com/2016/01/23/1785042-android-supera-ios-y-domina-el-mercado-global-sistemas-operativos-smartphones.

Java. 2015. Java. [En línea] 2015. [Citado el: 4 de 2 de 2016.] http://www.Java.com/.

Kioskea. 2014. CCM. [En línea] 6 de 2014. [Citado el: 26 de 4 de 2016.] http://es.ccm.net/contents/224-patrones-de-diseno.

Letelier, Patricio y Penadés, Carmen . 2006. [En línea] 6 de 2006. [Citado el: 3 de 2 de 2016.] http://www.cyta.com.ar/ta0502/v5n2a1.htm.

Machuca, Carlos Andrés Morales. 2010. *Estado del Arte: Servicios Web.* 2010.

Mondelo Hernández, Yonny, y otros. 2016. COJ. *Caribean Online Judge.* [En línea] 2016. [Citado el: 2 de 2 de 2016.] http://coj.uci.cu/general/about.xhtml?lang=es.

OASIS. 2006. Reference Model for Service Oriented Architecture. [En línea] 8 de 2006. [Citado el: 6 de 2 de 2016.] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

Open Handset Alliance. 2009. Open Handset Alliance. [En línea] 11 de 2009. [Citado el: 1 de 23 de 2016.] http://www.openhandsetalliance.com/oha_faq.html.

PC.net. 2016. pc. [En línea] 2016. [Citado el: 4 de 2 de 2016.] http://pc.net/glossary/definition/ide.

Pressman, Roger S. 2002. *Ingeniería del Software, un enfoque Práctico. quinta edición.* s.l. : McGraw-Hill Companies, 2002. 8448132149.

Qing, Li y Chen, Yu-Liu. 2009. *Unified Modeling Language.* Beijing : Springer Berlin Heidelberg, 2009. págs. 209-224. 978-3-540-89555-8.

Quinstreet Enterprise. 2016. Webopedia. [En línea] 2016. [Citado el: 23 de 1 de 2016.] http://www.webopedia.com/TERM/O/operating_system.html.

RAE. 2012. Real Academia Española. [En línea] 2012. [Citado el: 8 de 2 de 2016.] http://buscon.rae.es/draeI/SrvltGUIBusUsual?TIPO_HTML=2&BUS=3&LEMA=metodolog%C.

Somerville, Ian. 2005. *Ingeniería de software.* s.l. : Pearson Education, 2005.

SQLite. 2016. SQLite. [En línea] 18 de 4 de 2016. [Citado el: 28 de 4 de 2016.] https://www.sqlite.org/.

Techopedia Inc. 2016. Techopedia. [En línea] 2016. https://www.techopedia.com/definition/25099/android-app.

TechTarget. 2015. Search Enterprise Linux. [En línea] 2015. [Citado el: 1 de 23 de 2016.] http://searchenterpriselinux.techtarget.com/definition/Android.

—. 2010. Search Mobile Computing. [En línea] 2010. [Citado el: 23 de 1 de 2016.] http://searchmobilecomputing.techtarget.com/definition/Open-Handset-Alliance.

The Computer Language Company Inc. 2015. PC Magazine Encyclopedia. [En línea] 2015. http://www.pcmag.com/encyclopedia/term/51537/smartphone.

Tomás, Jesús. 2015. Diploma de Especialización en desarrollo de aplicaciones para Android. *Arquitectura Android.* [En línea] 11 de 2015. [Citado el: 6 de 2 de 2016.] http://www.androidcurso.com/index.php/99.

Valdéz, José Luis Cendejas. 2014. eumed. [En línea] 2014. [Citado el: 3 de 2 de 2016.] http://www.eumed.net/tesis-doctorales/2014/jlcv/software.htm.

Vazquez, MSc. Tomás Orlando Junco. 2012. *UN JUEZ EN LÍNEA AJUSTADO A LAS NECESIDADES DE LA DOCENCIA.* La Habana : s.n., 2012.

Vermeulen, Alan y W. Amber, Scott. 2001. *The Elements of Java(TM) Style.* New York : Cambridge University Press, 2001. 0521777682.

Visual Paradigm International. 2015. Visual Paradigm-Design & Managment Tool for Business IT System Development. [En línea] 2015. http://www.visual-paradigm.com/features/.