Universidad de las Ciencias Informáticas

Facultad 3

Título: Aplicación móvil para el desarrollo de evaluaciones frecuentes

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

Autor: Lucio Lázaro Pérez Martinez

Tutor: Ing. Guillermo Manuel Negrín Ortiz

Co-tutor: MSc. Julio César Díaz Vera

Julio 2018

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos 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 _____.

Lucio Lázaro Pérez Rodríguez **Guillermo M. Negrín Ortiz**

_____ _____

Firma del Autor Firma del Tutor

Julio César Díaz Vera

_____

Firma del Tutor

Datos de Contacto

Ing. Guillermo Manuel Negrín Ortiz

Universidad de las Ciencias Informáticas, La Habana, Cuba.

Correo electrónico: gmnegrin@uci.cu

MSc. Julio Cesar Díaz Vera

Universidad Martha Abreu, Villa Clara, Cuba.

Correo electrónico: jcdiaz@uci.cu

Dedicatoria

A mis amados padres, por su consagración, su amor, su interminable confianza y por ser unos ejemplos a seguir.

A mi tía Alicia por su amor y ser la más especial de mis tías y otra madre para mí.

A mis abuelos Publio e Hilda que, aunque no están entre nosotros sus ejemplos y amor siempre estarán.

Agradecimientos

A mi madre por todo ese cariño y amor que me has brindado, por ponerme a mi antes que todo, por todo lo que las palabras no pueden expresar, por ser mi madre.

A mi padre por ser un ejemplo para mí, por su amor, su entrega, por ser su primera prioridad, por ser el mejor padre para mí.

A mi tía Alicia por su preocupación, entrega, y cariño infinito, por ser otra madre para mí.

A mis abuelos por sus ejemplo y amor infinito.

A mis tutores por su colaboración, paciencia y apoyo para la realización del presente trabajo.

A los amigos con los que compartí los mejores momentos durante este largo camino.

A todo el que colaboró en la realización de este sueño, muchas gracias.

Resumen

Los recursos tecnológicos disponibles y su gran inserción en la sociedad y en especial en los estudiantes han hecho que la educación deba proponer nuevas y diversas estrategias que apoyen los procesos de enseñanza y aprendizaje con las nuevas tecnologías. Esto ha dado lugar a la aparición de diversas disciplinas como la Tecnología Educativa, la cual ha proporcionado al educador las herramientas de planificación y desarrollo, así como la tecnología, buscando mejorar los procesos de enseñanza y de aprendizaje a través del logro de los objetivos educativos y buscando la efectividad y el significado del aprendizaje. La tendencia en el uso de ambientes virtuales como escenarios de aprendizaje para la teleformación profesional y capacitación a distancia, han demostrado ser eficientes. Debido a que los teléfonos inteligentes poseen una capacidad cercana a una computadora es posible la utilización de los mismos a la hora de la formación de los estudiantes. El objetivo de la investigación es desarrollar una aplicación que garantice a los docentes un medio para desarrollar las evaluaciones frecuentes a través del uso de los dispositivos móviles. Se identificaron y describieron las Historias de Usuario, el Plan de Iteraciones y el Plan de Entrega, proporcionando el punto de partida para las actividades de implementación. Posteriormente se implementó la aplicación y se realizaron pruebas para validar el correcto funcionamiento de la herramienta corrigiendo los errores detectados. Como resultado se obtuvo la Aplicación para la Realización de Evacuaciones Frecuentes que permite la aplicación de evaluaciones frecuentes desde un dispositivo móvil con SO Android.

Palabras claves: Aplicación, evaluación frecuente, dispositivo móvil.

Tabla de contenidos

Contenido

Introducción

El conocimiento es el factor clave de la sociedad actual, una sociedad que es el resultado de las enormes transformaciones tecnológicas sucedidas desde finales de los años setenta del siglo pasado. Esta sociedad denominada, "Sociedad del Conocimiento", se encuentra sometida a constantes cambios y demudaciones debido a la celeridad de los avances tecnológicos (TUBELLA and REQUENA 2005).

Se trata de una sociedad en constante cambio, una sociedad que se mueve a gran velocidad, y que exige a los individuos un proceso de aprendizaje continuo no solo para su desempeño profesional sino para el pleno desarrollo de su vida cotidiana. Los individuos se ven obligados a adaptarse a situaciones cambiantes en todos los ámbitos de la actuación humana y a adoptar nuevos conocimientos y competencias para hacer frente a dichos cambios. Así, en una sociedad en movimiento surgen las tecnologías móviles para dar respuesta a las necesidades constantes de acceso a la información y de comunicación. El uso de estas tecnologías comienza a generalizarse a principios del siglo XXI, cuando los teléfonos y ordenadores que dependían de cables, y que en definitiva frenaban la movilidad, comienzan a ser sustituidos por sus homólogos inalámbricos. Pero estas tecnologías no se han limitado a dar respuestas, sino que su uso extendido ha revolucionado la forma de entender la comunicación y la educación (TUBELLA and REQUENA 2005).

Detrás de los teléfonos inteligentes existen grandes avances tecnológicos que han propiciado tan grande aceptación por la sociedad, los cuales son, batería con mejor duración, microprocesador multitareas mucho más potentes, pantallas táctiles con resoluciones mayores y mayor tamaño, gran capacidad de almacenamiento, cámaras más potentes y la posibilidad de hacer todos estos componentes tan pequeños que han permitido que dicho dispositivo quepa en la palma de la mano.

A partir del año 2005, especialmente en los países desarrollados, se produjo una espectacular expansión de la telefonía móvil y se comenzaron a utilizar de manera generalizada las tecnologías móviles para acceder a Internet (AYA and ALEJANDRO 2015). El acceso a la Red a través de estas tecnologías se realizaba especialmente con fines comerciales (e-business), académicos (e-learning) o de la Administración (e-government).

Para observar la inserción de los teléfonos inteligentes en los estudiantes en Universidad de las Ciencias Informáticas se tomó como muestra el 5to año de la Facultad 3 de la Universidad de las Ciencias Informáticas.

De los 49 estudiantes encuestados, solo 3 no poseían este tipo de dispositivo, lo que representaba un 6 por ciento de la muestra, mientras que 46 estudiantes, el 94 por ciento de la muestra si poseía este tipo de dispositivo, por lo no se necesitaría invertir en recursos a la hora de implementar una solución que permita la utilización de estos dispositivos en el aula de la universidad.

Los recursos tecnológicos disponibles y su gran inserción en la sociedad y en especial en los estudiantes han hecho que la educación deba proponer nuevas y diversas estrategias que apoyen los procesos de enseñanza y aprendizaje con las nuevas tecnologías. Esto ha dado lugar a la aparición de diversas disciplinas como la Tecnología Educativa, esta ha proporcionado al educador las herramientas de planificación y desarrollo, así como la tecnología, buscando mejorar los procesos de enseñanza y de aprendizaje a través del logro de los objetivos educativos y buscando la efectividad y el significado del aprendizaje (MOREIRA 2010). Un aspecto que lo hace tangible son los diversos sistemas informáticos denominados plataformas didácticas tecnológicas, como son, por ejemplo, la Plataforma EVA (Entorno Virtual de Aprendizaje) montada sobre Moodle y BlackBoard.

Estas plataformas son desarrollos informáticos que buscan representar la acción educativa en su conjunto y a la vez optimizar la creación, desarrollo, gestión y evaluación del proceso de enseñanza y aprendizaje a través de Internet. La tendencia en el uso de ambientes virtuales como escenarios de aprendizaje para la teleformación profesional y capacitación a distancia, han demostrado ser eficientes sistemas durante los últimos 20 años (MOREIRA 2010). Debido a que los teléfonos inteligentes poseen una capacidad cercana a una computadora, es posible acceder a estas plataformas desde estos dispositivos, lo que se ha convertido en una tendencia por la movilidad propia de dichos teléfonos.

La evaluación es el componente que regula el proceso de enseñanza-aprendizaje, y juega un papel fundamental en el cambio educativo. Responde a la pregunta: "¿en qué medidas han sido cumplidos los objetivos del proceso de enseñanza-aprendizaje?" (FUENTES *et al.* 1997).

En la actualidad, la evaluación debe responder a un proceso de enseñanza-aprendizaje desarrollador, promotor del cambio educativo, por lo que debe ser: desarrolladora, procesual, holística, contextualizada, democrática, formativa, cualitativa, investigativa, sistemática, que contemple la revalorización de errores, que tenga en cuenta indicadores que garanticen su objetividad, que promueva y transite por formas como la

heteroevaluación, coevaluación y autoevaluación, que garanticen un cambio cualitativamente superior (FUENTES *et al.* 1997).

El uso de los teléfonos inteligentes en el proceso de evaluación de los estudiantes, puede constituir un elemento diferenciador respecto a las prácticas evaluativas que hasta ahora se vienen realizando. Como consecuencia del nuevo cambio metodológico, los procesos de evaluación se verán inmediatamente afectados por el mismo. La nueva tecnología puede colaborar en todos estos procesos, no tanto para introducir cambios conceptuales en el mismo, sino como herramientas que permitan utilizar los recursos de tiempo y materiales de manera más eficiente, tanto para el profesor como para el estudiante, además de la posibilidad de almacenar claramente los resultados históricos de todos los estudiantes.

Con motivo de esta situación y para guiar el desarrollo de la investigación se plantea lo siguiente:

Problema a resolver:

¿Cómo desarrollar una aplicación para el sistema operativo Android que posibilite el desarrollo de la evaluación frecuente mediante el uso de teléfonos inteligentes?

Objetivo General:

Desarrollar una aplicación para el sistema operativo Android que permita la utilización de teléfonos inteligentes en el desarrollo de la evaluación frecuente.

Idea a defender:

Si se implementa una aplicación en el sistema operativo Android se puede diversificar la manera en la que se desarrollan las evaluaciones frecuentes.

Objeto de Estudio:

Proceso de desarrollo de software.

Campo de Acción:

Proceso de desarrollo de software para dispositivos móviles.

Posibles resultados:

Aplicación para el sistema operativo Android orientada al desarrollo de las evaluaciones frecuentes en clases.

Objetivos Específicos:

1. Establecer el marco conceptual para el desarrollo de la investigación.
2. Establecer el análisis y diseño de la aplicación.
3. Realizar el diseño e implementación de la aplicación para el sistema operativo Android.
4. Validar la aplicación a partir de pruebas de concepto.

Descripción de los capítulos:

Capítulo 1 Fundamentación teórica:

En este capítulo se establecen todos los elementos teóricos de la investigación. Se define la metodología para el desarrollo del software.

Capítulo 2 Propuesta de solución:

A lo largo de este capítulo se presenta la solución propuesta con todos los aspectos definidos en la fundamentación teórica. Se generan los artefactos, asociados a la metodología seleccionada en la fundamentación teórica en cada una de sus fases, además de otros artefactos que mejoraran el entendimiento de la propuesta a solución.

Capítulo 3 Validación de los resultados:

En este capítulo se realizan las validaciones de los resultados que se obtuvieron con el sistema ya en ejecución, haciendo uso de los métodos definidos. Además, se valida que el diseño realizado cumpla con la calidad requerida y que el sistema implementado satisface las necesidades de los clientes. Además se realizan pruebas de concepto para determinar si la solución cumple con los objetivos para los que fue definida.

Capítulo 1. Fundamentación Teórica

Introducción:

La presente investigación se enmarca en el desarrollo de una aplicación móvil, particularmente para el sistema operativo Android, que satisfaga el requisito de alto nivel de posibilitar realizar evaluaciones frecuentes en una clase universitaria. Constituyen elementos determinantes en el éxito de este trabajo las definiciones de aplicación, la especialización de las mismas que las convierte en aplicaciones móviles, la conceptualización de requisito de alto nivel y fundamentalmente la conceptualización de desarrollo de aplicaciones. Los elementos anteriormente mencionados posibilitarán establecer un marco de discernimiento a partir del cual puedan ser evaluados los resultados alcanzados.

En las secciones siguientes se presentan las definiciones formales de los términos, el análisis de la bibliografía en la que han sido presentados y el posicionamiento del investigador con respecto a las distintas alternativas encontradas en la literatura relevante.

1.1. Aplicación

El termino aplicación está estrechamente relacionado con la palabra en idioma inglés software que según Roger S. Pressman un software es: 1) instrucciones (programas de cómputo) que cuando se ejecutan proporcionan las características, función y desempeño buscados; 2) estructuras de datos que permiten que los programas manipulen en forma adecuada la información, y 3) información descriptiva tanto en papel como en formas virtuales que describen la operación y uso de los programas (PRESSMAN 2005).

Actualmente, hay siete grandes categorías de software de computadora que plantean retos continuos a los ingenieros de software (PRESSMAN 2005):

Software de sistemas: conjunto de programas escritos para dar servicio a otros programas. Determinado software de sistemas (por ejemplo, compiladores, editores y herramientas para administrar archivos) procesa estructuras de información complejas pero deterministas. Otras aplicaciones de sistemas (por

ejemplo, componentes de sistemas operativos, manejadores, software de redes, procesadores de telecomunicaciones) procesan sobre todo datos indeterminados. En cualquier caso, el área de software de sistemas se caracteriza por: gran interacción con el hardware de la computadora, uso intensivo por parte de usuarios múltiples, operación concurrente que requiere la secuenciación, recursos compartidos y administración de un proceso sofisticado, estructuras complejas de datos e interfaces externas múltiples.

Software de aplicación: programas aislados que resuelven una necesidad específica de negocios. Las aplicaciones en esta área procesan datos comerciales o técnicos en una forma que facilita las operaciones de negocios o la toma de decisiones administrativas o técnicas. Además de las aplicaciones convencionales de procesamiento de datos, el software de aplicación se usa para controlar funciones de negocios en tiempo real (por ejemplo, procesamiento de transacciones en punto de venta, control de procesos de manufactura en tiempo real).

Software de ingeniería y ciencias: se ha caracterizado por algoritmos “devoradores de números”. Las aplicaciones van de la astronomía a la vulcanología, del análisis de tensiones en automóviles a la dinámica orbital del transbordador espacial, y de la biología molecular a la manufactura automatizada. Sin embargo, las aplicaciones modernas dentro del área de la ingeniería y las ciencias están abandonando los algoritmos numéricos convencionales. El diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a hacerse en tiempo real e incluso han tomado características del software de sistemas.

Software incrustado: reside dentro de un producto o sistema y se usa para implementar y controlar características y funciones para el usuario final y para el sistema en sí. El software incrustado ejecuta funciones limitadas y particulares (por ejemplo, control del tablero de un horno de microondas) o provee una capacidad significativa de funcionamiento y control (funciones digitales en un automóvil, como el control del combustible, del tablero de control y de los sistemas de frenado).

Software de línea de productos: es diseñado para proporcionar una capacidad específica para uso de muchos consumidores diferentes. El software de línea de productos se centra en algún mercado limitado y particular (por ejemplo, control del inventario de productos) o se dirige a mercados masivos de consumidores (procesamiento de textos, hojas de cálculo, gráficas por computadora, multimedios, entretenimiento, administración de base de datos y aplicaciones para finanzas personales o de negocios).

Aplicaciones web: llamadas “webapps”, esta categoría de software centrado en redes agrupa una amplia gama de aplicaciones. En su forma más sencilla, las webapps son poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas limitadas. Sin embargo, desde que surgió Web 2.0, las webapps están evolucionando hacia ambientes de cómputo sofisticados que no sólo proveen características aisladas, funciones de cómputo y contenido para el usuario final, sino que también están integradas con bases de datos corporativas y aplicaciones de negocios.

Software de inteligencia artificial: hace uso de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o con el análisis directo. Las aplicaciones en esta área incluyen robótica, sistemas expertos, reconocimiento de patrones (imagen y voz), redes neurales artificiales, demostración de teoremas y juegos.

El autor considera que la aplicación que va a tener como resultado este trabajo, se clasificaría como software de aplicación, dado que esta va a gestionar y automatizar todo el proceso de evaluación frecuente. Cuando ha de construirse una aplicación nueva o sistema incrustado, deben escucharse muchas opiniones. Y en ocasiones parece que cada una de ellas tiene una idea un poco distinta de cuáles características y funciones debiera tener el software. Se concluye que debe hacerse un esfuerzo concertado para entender el problema antes de desarrollar una aplicación de software. Esta realidad simple lleva a una conclusión: debe hacerse ingeniería con el software en todas sus formas y a través de todos sus dominios de aplicación.

De lo antes planteado se desprende entonces que una aplicación es un software de tipo software de aplicación. Con el fin de constatar que efectivamente este

trabajo se enmarca en el desarrollo de una aplicación basta con comprobar que realizar evaluaciones frecuentes en una clase universitaria es una tarea y por tanto el software que se desarrolle con vista a complementarla es una aplicación.

1.2. Requisitos de alto nivel

Un requisito según (IEEE 2002) es una condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo. Existen diferentes clasificaciones para estos según diferentes autores. (SOMMERVILLE 2010) los clasifica por niveles y categorías, los primeros se dividen en requisitos de usuario y requisitos de sistema, también conocidos como requisitos de alto nivel, mientras que los segundos se dividen requisitos funcionales y no funcionales. Los requisitos de alto nivel son un elemento fundamental para la confección de un proyecto, son este tipo de requisitos los que se utilizan para establecer el primer entendimiento entre la persona que desea el sistema y el equipo encargado de su desarrollo. Y es finalmente asociados a estos requisitos que se evalúa si el sistema cumplió el objetivo para el que fue desarrollado.

1.3. Proceso de software

En el contexto de la ingeniería de software, un proceso no es una prescripción rígida de cómo elaborar software de cómputo. Por el contrario, es un enfoque adaptable que permite que las personas que hacen el trabajo (el equipo de software) busquen y elijan el conjunto apropiado de acciones y tareas para el trabajo. Se busca siempre entregar el software en forma oportuna y con calidad suficiente para satisfacer a quienes patrocinaron su creación y a aquellos que lo usarán (PRESSMAN 2005) .

La estructura del proceso establece el fundamento para el proceso completo de la ingeniería de software por medio de la identificación de un número pequeño de actividades estructurales que sean aplicables a todos los proyectos de software, sin importar su tamaño o complejidad. Además, la estructura del proceso incluye un conjunto de actividades sombrilla que son aplicables a través de todo el proceso del software. Una estructura de proceso general para la ingeniería de software consta de cinco actividades (PRESSMAN 2005):

Comunicación. Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros

participantes). Se busca entender los objetivos de los participantes respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software.

Planeación. Cualquier viaje complicado se simplifica si existe un mapa. Un proyecto de software es un viaje difícil, y la actividad de planeación crea un "mapa" que guía al equipo mientras viaja. El mapa —llamado plan del proyecto de software— define el trabajo de ingeniería de software al describir las tareas técnicas por realizar, los riesgos probables, los recursos que se requieren, los productos del trabajo que se obtendrán y una programación de las actividades.

Modelado. Ya sea un diseñador de paisaje, constructor de puentes, ingeniero aeronáutico, carpintero o arquitecto, a diario trabaja con modelos. Crea un "bosquejo" del objeto por hacer a fin de entender el panorama general —cómo se verá arquitectónicamente, cómo ajustan entre sí las partes constituyentes y muchas características más—. Si se requiere, refina el bosquejo con más y más detalles en un esfuerzo por comprender mejor el problema y cómo resolverlo. Un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.

Construcción. Esta actividad combina la generación de código (ya sea manual o automatizada) y las pruebas que se requieren para descubrir errores en éste.

Despliegue. El software (como entidad completa o como un incremento parcialmente terminado) se entrega al consumidor que lo evalúa y que le da retroalimentación, misma que se basa en dicha evaluación.

Todos los modelos del proceso del software pueden incluir las actividades estructurales generales, pero cada uno pone un distinto énfasis en ellas y define en forma diferente el flujo de proceso que invoca cada actividad estructural (así cómo acciones y tareas de ingeniería de software). De aquí la necesidad de seleccionar un modelo según las características del software a desarrollar, entre estos modelos se encuentran los de proceso descriptivo y los de proceso ágil.

Los **modelos de proceso prescriptivo** enfatizan la definición, la identificación y la aplicación detalladas de las actividades y tareas del proceso. Su objetivo es mejorar la calidad del sistema, desarrollar proyectos más manejables, hacer más

predecibles las fechas de entrega y los costos, y guiar a los equipos de ingenieros de software cuando realizan el trabajo que se requiere para construir un sistema. Si los modelos prescriptivos se aplican en forma dogmática y sin adaptación, pueden incrementar el nivel de burocracia asociada con el desarrollo de sistemas basados en computadora y crear inadvertidamente dificultades para todos los participantes (PRESSMAN 2005).

Los **modelos de proceso ágil** ponen el énfasis en la "agilidad" del proyecto y siguen un conjunto de principios que conducen a un enfoque más informal (pero no menos efectivo) del proceso de software. Por lo general, se dice que estos modelos del proceso son "ágiles" porque acentúan la maniobrabilidad y la adaptabilidad. Son apropiados para muchos tipos de proyectos y son útiles en particular cuando se hace ingeniería sobre aplicaciones web (PRESSMAN 2005).

1.4. Proceso de desarrollo de software para dispositivos móviles.

(ABRAHAMSSON 2005) presenta una serie de características a tener en cuenta a la hora de desarrollar software para dispositivos móviles, como son, que el software es liberado en un ambiente incierto y dinámico con un alto nivel de competencia, los equipos que desarrollan aplicaciones móviles son generalmente pequeñas y medianas empresas, las aplicaciones en sí son de pequeño tamaño, se entregan en versiones rápidas con el fin de satisfacer las demandas del mercado y se dirigen a un gran número de usuarios finales. El autor sugiere que en el desarrollo de aplicaciones de software para dispositivos móviles los equipos de desarrollo deben hacer frente al desafío de un entorno dinámico, con modificaciones frecuentes en las necesidades y expectativas del cliente (ABRAHAMSSON 2007), por tanto, deben ser desarrollados con enfoques orientados a ciclos de desarrollo relativamente cortos, propios de las metodologías ágiles para el desarrollo de software.

(ABRAHAMSSON 2005) realiza una comparación directa entre las características del método ágil y características de las aplicaciones móviles, centrándose en la cantidad de documentación producida, la planificación involucrada, el tamaño del equipo de desarrollo, la identificación del cliente, y la orientación a objetos. Excepto la identificación del cliente, todas las

características de las metodologías ágiles son adecuadas para el desarrollo de aplicaciones móviles. El cliente puede ser identificado como el distribuidor de software, sin embargo, especialmente en el caso de las aplicaciones móviles, el problema de la identificación del cliente es mucho más complejo.

La aparición de las metodologías ágiles no puede ser asociada a una única causa, sino a todo un conjunto de ellas, si bien es cierto que la mayoría de autores lo relacionan con una reacción a las metodologías tradicionales, ¿cuáles fueron las causas de esta reacción?, los factores que comúnmente se mencionan son la pesadez, lentitud de reacción y exceso de documentación, en definitiva, falta de agilidad de los modelos de desarrollo formales; otro punto importante sería la explosión de la red, las aplicaciones Web y las aplicaciones móviles, así como el crecimiento notorio del movimiento open source.

A todo esto, se puede añadir un cambio bastante importante, en cuanto a la demanda del mercado del software, cada vez más orientada a la web y a dispositivos móviles, con requisitos muy volátiles y en constante cambio, que requieren tiempos de desarrollo cada vez más cortos, lo que provocó que las empresas se fijaran más en nuevos desarrolladores, con nuevos métodos “amateurs” que se combinan con técnicas de las metodologías formales. Los modelos de desarrollo de las comunidades open source pudieron ciertamente determinar la aparición de las metodologías ágiles, pero cada autor determina el surgimiento de las metodologías ágiles de diferentes maneras.

1.5. Metodología de desarrollo

Desarrollar un buen software depende de un gran número de actividades y etapas, donde el impacto de elegir la metodología para un equipo en un determinado proyecto es trascendental para el éxito del producto.

Según la filosofía de desarrollo se pueden agrupar las metodologías en dos enfoques. Las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

Metodologías ágiles:

- FDD: Desarrollo Manejado por Rasgos

- XP: Extreme Programming
- Scrum
- AUP: Proceso Unificado Ágil

Metodologías tradicionales:

- MFS: MICROSOFT SOLUTION FRAMEWORK
- RUP: RATIONAL UNIFIED PROCESS

Para el desarrollo del sistema se hace necesaria la selección de una metodología adecuada a las características del proyecto que sea capaz de brindar ventajas al equipo de desarrollo.

Para la selección del enfoque a usar para el desarrollo del sistema se realizó una comparación de las principales características ofrecidas por cada uno de ellos (Ver Tabla 1) y las características que presenta el proyecto, decidiéndose hacer uso de un enfoque ágil para el desarrollo ya que es el que más se adecúa a las características del proyecto.

Tabla 1: Tabla comparativa entre el enfoque tradicional y ágil.

Metodologías Ágiles	Metodologías Robustas
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.

Principales características del proyecto:

- Equipo de desarrollo pequeño (solo 1 persona).

- Proyecto pequeño y de corta duración (solo 6 meses).
- Existe un constante intercambio con el cliente

Se seleccionó dentro de este enfoque como metodología de desarrollo EXTREME PROGRAMMING (XP por sus siglas en inglés).

1.5.1. Programación extrema

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el 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 la 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 cuenta con un conjunto de valores que son importantes para el logro del producto, entre los que se encuentran:

Valores de XP (BECK and GAMMA 2000)

- Comunicación: Crear software requiere de sistemas comunicados.
- Simplicidad: Empezar con lo necesario y requerido y trabajar desde ahí.
- Retroalimentación: Del sistema, del cliente, y del equipo.
- Respeto: El equipo debe trabajar como uno, sin hacer decisiones repentinas.

Cuenta también con las siguientes características y artefactos, los cuales son necesarios en el desarrollo de la metodología.

Características fundamentales (BECK and GAMMA 2000)

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, rescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad, pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Artefactos esenciales de XP (BECK and GAMMA 2000)

- Historias del usuario.
- Tareas de ingeniería.
- Pruebas de aceptación.
- Pruebas unitarias y de integración.
- Plan de iteraciones.
- Código.
- Casos de prueba.

Fases de la metodología XP:

- **Planificación:** Durante esta etapa se lleva a cabo el proceso de identificación y confección de las HU, el plan de iteraciones y el plan de entrega.

- **Diseño:** Durante esta etapa se crea un diseño evolutivo que va mejorando incrementalmente y que permite hacer entregas pequeñas y frecuentes de valor para el cliente, basado principalmente en el desarrollo de las tarjetas Clase-Responsabilidad- Colaboración (CRC).

- **Desarrollo:** En esta fase se realiza la implementación de las HU que fueron seleccionadas por cada iteración. Al inicio se lleva a cabo un chequeo del plan de iteraciones por si es necesario realizar modificaciones. Como parte de este plan se crean tareas de ingeniería para ayudar a organizar la implementación exitosa de las HU.

- **Pruebas:** Esta fase permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones y refactorizaciones. XP divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñadas por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñada por el cliente final.

Descripción de los artefactos propuestos a generar

- **Historias del usuario:** Las historias de usuario (HU) son la técnica utilizadas en XP para la descripción de los requisitos del software. Representan una breve descripción del comportamiento del sistema. Emplea terminología del cliente sin lenguaje técnico para describir brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Se emplean para hacer estimaciones de tiempo y son la base para las pruebas de unidad y presiden la creación de las pruebas de aceptación.

  Las historias de usuario deben ser: (COHN 2004)

- **Independientes unas de otras:** De ser necesario, combinar las historias dependientes o buscar otra forma de dividir las historias de manera que resulten independientes.
- **Negociables:** La historia en sí misma no es lo suficientemente explícita como para considerarse un contrato, la discusión con los usuarios debe permitir esclarecer su alcance y éste debe dejarse explícito bajo la forma de pruebas de validación.
- **Valoradas por los clientes o usuarios:** Los intereses de los clientes y de los usuarios no siempre coinciden, pero en todo caso, cada historia debe ser importante para alguno de ellos más que para el desarrollador.
- **Estimables:** Un resultado de la discusión de una historia de usuario es la estimación del tiempo que tomará completarla. Esto permite estimar el tiempo total del proyecto.
- **Pequeñas:** Las historias muy largas son difíciles de estimar e imponen restricciones sobre la planificación de un desarrollo iterativo. Generalmente se recomienda la consolidación de historias muy cortas en una sola historia.
- **Verificables:** Las historias de usuario cubren requerimientos funcionales, por lo que generalmente son verificables. Cuando sea posible, la verificación debe automatizarse, de manera que pueda ser verificada en cada entrega del proyecto.

- **Plan de entrega:** es una práctica en donde el Cliente presenta las características deseadas a los programadores, y los programadores estiman la dificultad. Teniendo las estimaciones de costo, y sabiendo la importancia de las características, el Cliente establece un plan para el proyecto.
- **Tareas de ingeniería:** las tareas de ingeniería son un conjunto de acciones a desarrollar para resolver las historias de usuario. Permiten organizar el proceso de implementación además de posibilitar que sea

conocido el grado de complejidad de cada historia de usuario teniendo en cuenta la cantidad de tareas asociadas a ella.

- **Pruebas de aceptación:** constituyen pruebas basadas en la ejecución, revisión y retroalimentación de las funcionalidades que han sido diseñadas para el software. Estas pruebas se realizan a través de modelos de prueba conocidos como casos de prueba.
- **Pruebas unitarias:** constituyen una forma de probar el correcto funcionamiento de un módulo de códigos, por lo que también se le conoce como pruebas modulares. Tienen como objetivo asegurar que cada uno de los módulos de códigos funcione correctamente por separado.
- **Plan de iteraciones:** define exactamente cuáles historias de usuario serán implementadas en cada iteración. Se toma como base cada una de las historias de usuarios y el esfuerzo que se requiere para el desarrollo de estas, y se procede a fragmentar el trabajo en iteraciones.

Conclusiones del capítulo

Se estableció el marco teórico de referencia para la investigación en el cual se obtiene un mejor dimensionamiento del problema a partir del análisis de los principales conceptos asociados a la solución, tales como software, aplicación, requisito de alto nivel, y metodología de desarrollo. De este último concepto se decidió seguir un enfoque ágil para su desarrollo, seleccionando la metodología XP para guiar el proceso, presentando los artefactos y adaptaciones a la metodología que se hicieron teniendo en cuenta las condiciones del equipo de proyecto.

Capítulo 2. Propuesta de solución

En el presente capítulo se hace una descripción de la propuesta de solución mediante las tres primeras fases que define la metodología XP (Planificación, Diseño e Implementación) y los principales artefactos a generar en cada una de estas. Entre los artefactos que se generarán se encuentran las historias de usuarios, el plan de iteraciones y las tareas de ingeniería, para obtener de esta forma un producto con la calidad requerida por el usuario final.

2.1. Descripción de los procesos del negocio

Para lograr una mejor comprensión del negocio a continuación se hace una breve descripción de las características y el funcionamiento de los principales procesos que intervienen en el negocio:

Evaluación: El proceso de evaluación es mediante el cual se realiza la obtención y presentación de las evaluaciones que serán aplicadas a los estudiantes, este proceso involucra a solo a los profesores.

Revisión y otorgamiento de nota: Este proceso se lleva a cabo una vez que el estudiante haya concluido la evaluación y solicite su nota al sistema, consiste en la recopilación por parte del sistema de las respuestas emitidas por el estudiante a la pregunta, este compara las respuestas emitidas por el estudiante con las respuestas correctas anteriormente definidas y emitirá una nota basándose en el porciento de aciertos por el estudiante, posteriormente se notificará al estudiante de su nota.

2.2. Planificación

La planificación en XP está basada en un conjunto de decisiones tomadas por el cliente de conjunto con los programadores. En esta primera fase se debe hacer primero una recopilación de todos los requisitos del proyecto, también debe

haber una interacción con el usuario, y se debe planificar bien entre los desarrolladores que es lo que se quiere para así lograr los objetivos finales.

2.2.1. Requisitos del Sistema

La ingeniería de requisitos proporciona el mecanismo apropiado para entender lo que el cliente quiere, analizar las necesidades, evaluar la factibilidad, negociar una solución razonable, especificar la solución sin ambigüedades, validar la especificación, y administrar los requisitos conforme estos se transforman en un sistema operacional (PRESSMAN 2005).

Los requisitos de un sistema describen los servicios que ha de ofrecer el sistema y las restricciones asociadas a su funcionamiento (SOMMERVILLE 2010).

Técnicas de Captura

Para la captura de los requisitos tantos funcionales como no funcionales las técnicas que se aplicarán serán la entrevista (abierta y cerrada) y la tormenta de ideas (SOMMERVILLE 2010).

Luego de la aplicación de las técnicas para la obtención de los requerimientos del sistema se obtuvieron los siguientes.

Requisitos Funcionales

Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar ante entradas particulares y de cómo se debe comportar en situaciones particulares (SOMMERVILLE 2010).

Se identificaron 5 requisitos funcionales agrupados en 5 historias de usuarios. A continuación, se muestran los requisitos obtenidos con la historia de usuario a la que pertenecen y su prioridad para el negocio.

Tabla 2: Agrupación de requisitos funcionales por historias de usuario.

Historia de Usuario	Requisitos Funcionales	Prioridad
HU_1: Autenticar usuario	RF#1_ Identificar estudiante	Alta
HU_2: Obtener pregunta a evaluar	RF#2_ Obtener pregunta a evaluar	Alta
HU_3: Enviar las respuestas a evaluar	RF#3_ Enviar las respuestas a evaluar	Alta
HU_4: Identificar respuesta por estudiante	RF#4_ Identificar respuesta por estudiante	Alta
HU_5: Obtener evaluaciones emitidas	RF#5_ Obtener evaluaciones emitidas	Media

Requisitos no Funcionales

Son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares (SOMMERVILLE 2010).

Estos requerimientos especifican o restringen las propiedades emergentes del sistema, pueden enmarcar el rendimiento, la protección, la disponibilidad y otras propiedades del sistema (SOMMERVILLE 2010).

El incumplimiento de un requerimiento no funcional puede significar que el sistema entero sea inutilizable ya que estos son más críticos en ocasiones que los requerimientos funcionales debido a que los usuarios normalmente pueden encontrar formas de trabajar alrededor de una función del sistema que realmente

no cumple sus necesidades pero no ante la falla de un requerimiento no funcional que sea crítico para el funcionamiento del sistema (SOMMERVILLE 2010).

Se identificaron 8 requisitos no funcionales lo cuales para un mejor entendimiento se agruparon por las siguientes clasificaciones:

Requerimientos de Software

- Teléfonos inteligentes con Android 4.4 o superior.

Requerimientos de Hardware

- En el cliente se requiere un teléfono inteligente con 256 MB de RAM como mínimo, procesador a 1 GHz de velocidad de procesamiento o superior y un adaptador WiFi.
- Todos los teléfonos implicados en la funcionalidad de la aplicación deben estar conectados a la WiFi.

Interfaz

- Tiene que poseer una interfaz amigable y de fácil entendimiento para el estudiante.
- El sistema deberá ser independiente de la plataforma.
- Los colores serán estándares en toda la aplicación.

Usabilidad

- El sistema deberá poseer una interfaz sencilla, lo más atractiva y clara posible para el usuario, además de poder ser usado por cualquier persona con conocimientos básicos en el manejo de teléfonos inteligentes.

Disponibilidad

- Para poder hacer uso del sistema deberá existir conexión.

2.2.2. Historias de Usuario

Durante la fase de exploración se identificaron 5 Historias de Usuario (HU), representando cada una las funcionalidades del sistema, a continuación, se ilustran las HU identificadas las cuales se describen en las siguientes figuras.

Tabla 3: Descripción de la HU Autenticar usuario.

Historia de Usuario	
Número: HU_1	**Nombre del requisito:** Identificar estudiante.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Estudiante	**Iteración Asignada:** 1
Prioridad: Alta	**Puntos Estimado:** 1
Riesgo en Desarrollo: Medio	**Puntos Real:** 1
Descripción: El estudiante al acceder al sistema por primera vez insertará su nombre, apellidos y grupo, el sistema guardará los datos insertados permitiendo identificar al usuario en posteriores accesos al sistema.	
Observaciones: Todos los usuarios deberán insertar los datos requeridos en su primer acceso al sistema.	

Tabla 4: Descripción de la HU Obtener pregunta a evaluar.

Historia de Usuario	
Número: HU_2	**Nombre del requisito:** Obtener pregunta a evaluar.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Estudiante	**Iteración Asignada:** 1
Prioridad: Alta	**Puntos Estimado:** 2
Riesgo en Desarrollo: Medio	**Puntos Real:** 2
Descripción: Permite obtener la pregunta que se le va a evaluar al estudiante.	
Observaciones:	

Tabla 5: Descripción de la HU Enviar las respuestas a evaluar.

Historia de Usuario	
Número: HU_3	**Nombre del requisito:** Enviar las respuestas a evaluar.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Estudiante	**Iteración Asignada:** 1
Prioridad: Alta	**Puntos Estimado:** 2
Riesgo en Desarrollo: Medio	**Puntos Real:** 2
Descripción: Permite enviar las respuestas dadas por el estudiante para obtener una evaluación.	
Observaciones:	

Tabla 6: Descripción de la HU Identificar respuesta por estudiante.

Historia de Usuario	
Número: HU_4	**Nombre del requisito:** Identificar respuesta por estudiante.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Estudiante	**Iteración Asignada:** 1
Prioridad: Alta	**Puntos Estimado:** 2
Riesgo en Desarrollo: Medio	**Puntos Real:** 2
Descripción: Permite saber a qué estudiantes pertenecen cada respuesta.	
Observaciones:	

Tabla 7: Descripción de la HU Obtener evaluaciones emitidas.

Historia de Usuario	
Número: HU_5	**Nombre del requisito:** Obtener evaluaciones emitidas.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Estudiante	**Iteración Asignada:** 2
Prioridad: Media	**Puntos Estimado:** 2
Riesgo en Desarrollo: Medio	**Puntos Real:** 2
Descripción: Permite saber al estudiante cual fue la evaluación obtenida.	
Observaciones:	

2.2.3. Plan de Iteraciones

En este plan se definen varias iteraciones sobre el sistema antes de ser entregado. Algunos de los elementos que deben tenerse en cuenta para su elaboración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas y tareas no terminadas en la iteración.

El plan especifica exactamente que historias de usuarios serán implementadas para cada lanzamiento del sistema. Esto da al cliente un grupo de historias de usuario para incluir en el próximo plan de iteración (CAMPOS 2004).

A continuación, se muestra el plan de iteraciones definido para el desarrollo del sistema:

Tabla 8: Descripción del plan de iteraciones.

Número de Iteración	Descripción de la iteración	HU a implementar
1	Se tuvo en cuenta aquellas historias de usuario de mayor importancia en cuanto a la funcionalidad que describen cada una, que se ve reflejado en la alta prioridad para el negocio.	**HU_1:** Identificar estudiante. **HU_2:** Obtener pregunta a evaluar. **HU_3:** Enviar las respuestas a evaluar. **HU_4:** Identificar respuesta por estudiante. .
2	Se tuvo en cuenta aquellas funcionalidades del sistema con menos prioridad, es decir, aquellas historias de usuario que presentan prioridad media o baja. Por otra parte, en esta iteración se corregirán los errores encontrados en la iteración anterior, obteniéndose una nueva versión del sistema.	**HU_5:** Obtener evaluaciones emitidas.

2.2.4. Plan de Entregas

El plan de entregas es un documento que especifica exactamente que historias de usuario serán implementadas en cada entrega del sistema y sus prioridades, de modo que también permita conocer con exactitud qué historias de usuario serán implementadas en la próxima liberación. La idea principal del plan es hacer

entregas frecuentes para obtener una mayor retroalimentación. A continuación, se presenta una propuesta para el plan de entrega elaborado por el equipo de desarrollo donde se reflejan las fechas de entregas para las primeras versiones de las historias de usuario.

Tabla 9: Descripción del plan de entregas.

Número de iteración	HU a implementar	Fecha de entrega
1	**HU_1:** Identificar estudiante. **HU_2:** Obtener pregunta a evaluar. **HU_3:** Enviar las respuestas a evaluar. **HU_4:** Identificar respuesta por estudiante. .	18 de abril del 2018
2	**HU_5:** Obtener evaluaciones emitidas.	1 de mayo del 2018

2.3. Diseño

XP sugiere que hay que conseguir diseños simples y sencillos. Para procurar hacerlo todo lo menos complicado posible para el usuario o cliente y conseguir un diseño fácilmente entendible que costará menos tiempo y esfuerzo para desarrollarlo. Además de los artefactos propuestos a generar por XP.

2.3.1. Tarjetas CRC

Las tarjetas CRC, Class, Responsibilities and Collaboration (CRC por sus siglas en inglés) sirven para diseñar el sistema en conjunto entre todo el equipo. Estas tarjetas representan objetos, para los cuales se especifica la clase a la que

pertenece dicho objeto, las responsabilidades u objetivos que debe cumplir y las clases que colaboran con cada responsabilidad.

A continuación, se muestran las tarjetas CRC creadas.

Tarjeta CRC	
Clase: MainActivity	
Responsabilidades: Identificar al estudiante Obtener listado de preguntas Mostrar listado de preguntas Mostrar evaluación	**Colaboraciones:** PreguntaSeleccionMultiple PreguntaSeleccionSimple PreguntaVoF

Tarjeta CRC	
Clase: PreguntaSeleccionMultiple	
Responsabilidades: Mostrar pregunta Enviar la respuesta del estudiante	**Colaboraciones:** MainActivity

Tarjeta CRC	
Clase: PreguntaSeleccionSimple	
Responsabilidades: Mostrar pregunta Enviar la respuesta del estudiante	**Colaboraciones:** MainActivity

Tarjeta CRC	
Clase: PreguntaVoF	
Responsabilidades: Mostrar pregunta Enviar la respuesta del estudiante	**Colaboraciones:** MainActivity

2.3.2. Patrón de Arquitectura

Los patrones arquitectónicos, o patrones de arquitectura, también llamados arquetipos, ofrecen soluciones a problemas de arquitectura de software en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor (PRESSMAN 2005).

Modelo-Vista-Controlador

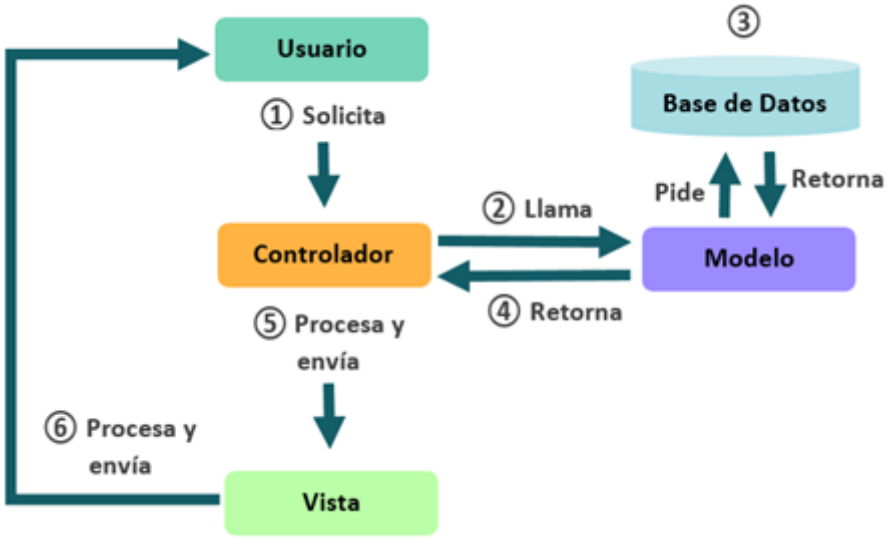
El Modelo–Vista–Controlador (MVC por sus siglas en inglés) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario (MAR-CORNELIO and CAEDENTEY-MORENO 2016).

Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

De manera genérica, los componentes de MVC se definen como sigue (LEFF and RAYFIELD 2001):

- **El Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- **El Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta de 'modelo', por tanto, se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
- **La Vista:** Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

Ilustración 1: Representación del patrón MVC



Para estructurar el desarrollo del software se usó como patrón arquitectónico el MVC apoyados en los principios y arquitectura que propone el consumo de servicios desde dispositivos móviles, además de las facilidades que ofrece tanto en términos de escalabilidad, pues divide la lógica del negocio de la lógica del diseño como en la sencillez de mantenimiento, además de mantener un bajo acoplamiento.

2.3.3. Patrones de Diseño

El patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas (LARMAN 1999).

Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería del software. Todo lo contrario: intentan codificar el conocimiento, las expresiones y los principios ya existentes (LARMAN 1999).

Existen dos clasificaciones en las que se agrupan, los patrones GRASP (General Responsibility Asignment Software Patterns) en español Patrones Generales de Software para Asignar Responsabilidades, los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones y los patrones GOF (Gang Of Four) que solucionan problemas de creación de instancias.

Para garantizar un mejor diseño del sistema propuesto se hizo uso de los siguientes patrones de diseños.

Patrones GRASP utilizados en la solución

Creador

El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento, constituyendo su principal beneficio, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización (LARMAN 1999).

Beneficios ofrecidos por el patrón (LARMAN 1999):

- Se brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

Este patrón se evidencia en las clases controladoras que son las encargadas de crear las instancias de los objetos que manejan, favoreciendo así la reutilización y el bajo acoplamiento.

Ilustración 2: Uso del patrón creador.

```
final String url = "http://192.168.137.1:3000/api/estudiante";
StringRequest postRequest = new StringRequest(Request.Method.POST, url,
        new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                // response
                Log.d("Response", response);
            }
        },
        new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // error
                Log.d("Error.Response", error.toString());
            }
        }
) {
    @Override
    protected Map<String, String> getParams() {
        Map<String, String> params = new HashMap<~>();
        params.put("nombre", nombre.getText().toString());
        params.put("apellidos", apellidos.getText().toString());
        params.put("grupo", grupo.getText().toString());
        params.put("imei", imei);


        return params;
    }
};
queue.add(postRequest);
```

Bajo acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. El Bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como experto o alta cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades (LARMAN 1999).

Ventajas del uso del patrón (LARMAN 1999):

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar.

Estas ventajas permitieron potenciar la reutilización y disminuir la dependencia entre las clases. La arquitectura del sistema es la máxima evidencia de la utilización de este patrón

Alta cohesión

En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme (LARMAN 1999).

Grady Booch señala que se da una alta cohesión funcional cuando los elementos de un componente (clase, por ejemplo) colaboran para producir algún comportamiento bien definido.

Entre los beneficios ofrecidos por el patrón se encuentran (LARMAN 1999):

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Su utilización puede verse, de conjunto con el bajo acoplamiento, en la arquitectura del sistema.

Controlador

Propone asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase (LARMAN 1999). En la arquitectura definida este patrón se evidencia en las clases controladoras, responsables de implementar todas las funcionalidades pertenecientes a una interfaz determinada.

Patrones de Comportamiento (Patrones GOF):

Decorador

Añade responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades. Provee una alternativa muy flexible para agregar funcionalidad a una clase (GAMMA 1995).

Ilustración 3: Uso patrón decorador

```
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:fitsSystemWindows="true"
tools:context="com.example.user.evaluacion.MainActivity">

<android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        app:popupTheme="@style/AppTheme.PopupOverlay" />

</android.support.design.widget.AppBarLayout>

<include layout="@layout/content_main" />

<android.support.design.widget.FloatingActionButton
    android:id="@+id/obtenerPreguntas"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="16dp"
    app:srcCompat="@android:drawable/ic_dialog_email" />
```

La utilización de estos patrones resultó en un diseño sencillo, pensado para la creación de un sistema robusto, fácil de entender, mantener y ampliar; aumentando la capacidad de reutilización de sus componentes y conservando el encapsulamiento de la información.

2.4. Implementación

2.4.1. Herramientas y tecnologías

2.4.1.1. Entorno de Desarrollo Integrado

Un IDE (Entorno de desarrollo integrado, por sus siglas en inglés) es un programa informático compuesto por un conjunto de herramientas (un editor de código, un compilador, un depurador y un constructor de interfaz gráfica) para

programar en un lenguaje de programación o en varios, que van a facilitar las tareas de desarrollo y mantenimiento de las aplicaciones (SALAVERT and PÉREZ 2000).

Android Studio

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

- Un sistema de compilación basado en Gradle flexible
- Un emulador rápido con varias funciones
- Un entorno unificado en el que puedes realizar desarrollos para todos los dispositivos Android
- Instant Run para aplicar cambios mientras tu aplicación se ejecuta sin la necesidad de compilar un nuevo APK
- Integración de plantillas de código y GitHub para ayudarte a compilar funciones comunes de las aplicaciones e importar ejemplos de código
- Gran cantidad de herramientas y frameworks de prueba
- Herramientas Lint (realizan un análisis estático del código fuente) para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, etc.
- Compatibilidad con C++ y NDK
- Soporte incorporado para Google Cloud Platform, lo que facilita la integración de Google Cloud Messaging y App Engine

2.4.1.2. Lenguajes de Programación

Java

Java es un lenguaje de programación con el que se puede realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras (ALVAREZ 2001).

Una de las principales características por las que Java se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Eso quiere decir que si se hace un programa en Java podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo, Windows, Linux. Esto lo consigue porque se ha creado una Máquina Virtual de Java para cada sistema que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente (ALVAREZ 2001).

2.4.2. Estándares de codificación

XP resalta que la comunicación de los programadores es a través del código, por lo que es necesario que sigan ciertos estándares de programación para lograr un entendimiento entre los programadores, de manera que cualquier persona del equipo de desarrollo pueda modificar el código. Además, se hace preciso que el código sea entendible para que posteriormente otros programadores puedan apoyarse en ese trabajo y desarrollen otras soluciones.

En el caso de la herramienta que se desarrolla, el estándar que se utiliza es:

Máxima longitud de las líneas

- Todas las líneas se limitan a un máximo de 79 caracteres.

Importaciones

- Las importaciones se encuentran en líneas separadas.

Comentarios

- Se utilizan comentarios de una línea para hacer más entendible el código.

Comentarios de una línea: comentario pequeño que solo abarca una línea y describe el código que le sigue.

```
// Esto es un comentario de una línea
```

Estilo de los nombres

- **Clases:** los nombres de las clases presentan la primera letra en mayúscula, en caso de ser un nombre compuesto, la inicial de cada palabra se representa en mayúscula. Se utilizan nombres simples y de alguna manera que describan el contenido, se usan palabras completas, a no ser que la abreviatura sea muy conocida.
- **Métodos y variables:** los nombres de los métodos se representan en minúscula, en caso de ser un nombre compuesto, la inicial de la primera palabra se simboliza en minúscula, y la de las otras palabras que lo componen en mayúscula. Los nombres de las variables son cortos, pero con significados lógicos, capaces de permitir a un observador identificar su función.

2.4.3. Tareas de Ingeniería

Luego de la descripción de las historias de usuarios del sistema, se hace necesario la definición un grupo de tareas ingenieriles orientadas a dar cumplimiento a la implementación de las funcionalidades definidas en ellas, lo cual sienta las bases para el posterior desarrollo de la solución.

Las tareas de ingeniería son un conjunto de acciones a desarrollar para resolver las historias de usuario. Permiten organizar el proceso de implementación además de posibilitar que sea conocido el grado de complejidad de cada historia de usuario teniendo en cuenta la cantidad de tareas asociadas a ella.

Tabla 10: Descripción de la tarea de ingeniería de la HU_1

Tareas de la Ingeniería	
Número de Tarea: T_1	Número de Historia de Usuario: HU_1
Nombre Tarea: Diseñar las interfaces requeridas para la funcionalidad: Identificar estudiante	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 0.5
Fecha Inicio:2-3-2018	Fecha Fin:4-3-2018
Programador Responsable: Lucio	
Descripción: Se crean las interfaces necesarias	

Tabla 11: Descripción de la tarea de ingeniería de la HU_1

Tareas de la Ingeniería	
Número de Tarea: T_2	Número de Historia de Usuario: HU_1
Nombre Tarea: Implementar funcionalidad: Identificar estudiante	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 0.5
Fecha Inicio:5-3-2018	Fecha Fin:7-3-2018
Programador Responsable: Lucio	
Descripción: Se crean las clases y métodos necesarios para la implementación de la funcionalidad	

Tabla 12: Descripción de la tarea de ingeniería de la HU_2

Tareas de la Ingeniería	
Número de Tarea: T_3	Número de Historia de Usuario: HU_2
Nombre Tarea: Diseñar las interfaces requeridas para la funcionalidad: Obtener pregunta a evaluar	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:8-3-2018	Fecha Fin:14-3-2018
Programador Responsable: Lucio	
Descripción: Se crean las interfaces necesarias	

Tabla 13: Descripción de la tarea de ingeniería de la HU_2

Tareas de la Ingeniería	
Número de Tarea: T_4	Número de Historia de Usuario: HU_2
Nombre Tarea: Implementar funcionalidad: Obtener pregunta a evaluar	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:15-3-2018	Fecha Fin:21-3-2018
Programador Responsable: Lucio	
Descripción: Se crean las clases y métodos necesarios para la implementación de la funcionalidad	

Tabla 14: Descripción de la tarea de ingeniería de la HU_3

Tareas de la Ingeniería	
Número de Tarea: T_5	Número de Historia de Usuario: HU_3
Nombre Tarea: Diseñar las interfaces requeridas para la funcionalidad: Enviar las respuestas a evaluar	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:22-3-2018	Fecha Fin:28-3-2018
Programador Responsable: Lucio	
Descripción: Se crean las interfaces necesarias	

Tabla 15: Descripción de la tarea de ingeniería de la HU_3

Tareas de la Ingeniería	
Número de Tarea: T_6	Número de Historia de Usuario: HU_3
Nombre Tarea: Implementar funcionalidad: Enviar las respuestas a evaluar	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:29-3-2018	Fecha Fin:4-4-2018
Programador Responsable: Lucio	
Descripción: Se crean las clases y métodos necesarios para la implementación de la funcionalidad	

Tabla 16: Descripción de la tarea de ingeniería de la HU_4

Tareas de la Ingeniería	
Número de Tarea: T_7	Número de Historia de Usuario: HU_4
Nombre Tarea: Diseñar las interfaces requeridas para la funcionalidad: Identificar respuesta por estudiante	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:5-2-2018	Fecha Fin:11-4-2018
Programador Responsable: Lucio	
Descripción: Se crean las interfaces necesarias	

Tabla 17: Descripción de la tarea de ingeniería de la HU_4

Tareas de la Ingeniería	
Número de Tarea: T_8	Número de Historia de Usuario: HU_4
Nombre Tarea: Implementar funcionalidad: Identificar respuesta por estudiante	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:12-4-2018	Fecha Fin:18-4-2018
Programador Responsable: Lucio	
Descripción: Se crean las clases y métodos necesarios para la implementación de la funcionalidad	

Tabla 18: Descripción de la tarea de ingeniería de la HU_5

Tareas de la Ingeniería	
Número de Tarea: T_9	Número de Historia de Usuario: HU_5
Nombre Tarea: Diseñar las interfaces requeridas para la funcionalidad: Obtener evaluaciones emitidas	
Tipo Tarea: Desarrollo	Puntos Estimados (semanas): 1
Fecha Inicio:19-4-2018	Fecha Fin:25-4-2018
Programador Responsable: Lucio	
Descripción: Se crean las interfaces necesarias	

Tabla 19: Descripción de la tarea de ingeniería de la HU_5

Tareas de la Ingeniería	
Número de Tarea: T_10	**Número de Historia de Usuario: HU_5**
Nombre Tarea: Implementar funcionalidad: Obtener evaluaciones emitidas	
Tipo Tarea: Desarrollo	**Puntos Estimados (semanas): 1**
Fecha Inicio:26-4-2018	**Fecha Fin:1-5-2018**
Programador Responsable: Lucio	
Descripción: Se crean las clases y métodos necesarios para la implementación de la funcionalidad	

Conclusiones del capítulo

Se realiza el análisis, diseño e implementación de la aplicación mediante la elaboración de los artefactos correspondientes a los flujos de trabajo indicados por la metodología de desarrollo XP, creando así la documentación necesaria de la investigación que sirvió de guía a los desarrolladores para la implementación de la solución. Se trataron los aspectos más significativos del proceso de implementación, donde se mostró el Plan de Iteraciones y el estándar de codificación empleado en la solución lo que permitió realizar las diferentes entregas al cliente y estandarizar el código del sistema para la realización de futuros mantenimientos. Se obtuvo la implementación de la aplicación, dando solución a los requisitos funcionales identificados.

Capítulo 3. Validación de los resultados

El presente capítulo precisa cada uno de los pasos a la hora de la realización de las pruebas para la obtención de una aplicación sin deficiencias. Se ofrece un análisis de cada uno de los niveles de pruebas ejecutados, los términos en que se realizaron las pruebas y los resultados alcanzados.

3.1. Pruebas de aceptación

Las pruebas de aceptación en la metodología XP son especificadas por el cliente, y se centran en las características y funcionalidades generales del sistema, que son visibles y revisables por parte del usuario. Estas pruebas derivan de las HU que se han implementado como parte de la liberación del software (JOSKOWICZ 2008).

Los clientes son responsables de verificar que los resultados de estas pruebas sean correctos. Así mismo, en caso de que fallen varias pruebas, deben indicar el orden de prioridad de resolución. Una HU no se puede considerar terminada hasta tanto pase correctamente todas las pruebas de aceptación. Dado que la responsabilidad es grupal, es recomendable publicar los resultados de las pruebas de aceptación, de manera que todo el equipo esté al tanto de esta información (JOSKOWICZ 2008).

3.2. Pruebas de Calidad

Las pruebas son un conjunto de actividades que se planean con anticipación y se realizan de manera sistemática con el objetivo de detectar errores en el software (PRESSMAN 2005).

Hay dos maneras de probar cualquier producto construido: uno, si se conoce la función específica para la que se diseñó el producto, se aplican pruebas que demuestren que cada función es plenamente operacional, mientras que se buscan los errores de cada función; dos, si se conoce el funcionamiento interno, se aplican pruebas para asegurarse que “todas las piezas encajan”, es decir que las operaciones internas se realizan de acuerdo con las especificaciones, y se han probado todos los componentes internos de manera adecuada. Al primer enfoque de prueba se le denomina prueba de caja negra, y al segundo prueba de caja blanca (PRESSMAN 2005).

XP propone como pruebas a realizar las pruebas unitarias y las pruebas de aceptación las cuales serán llevadas cabos bajo casos de prueba previamente diseñados como lo propone la metodología.

3.2.1. Pruebas Unitarias (caja blanca)

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa cerciorándose que se devuelvan los valores de salida adecuados (PRESSMAN 2005).

- Las pruebas de caja blanca intentan garantizar que:
- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

La técnica utilizada dentro de las pruebas de caja blanca fue camino básico. En la ilustración 4 se enumeran las sentencias de código del método obtenerEvaluacion().

Ilustración 4: Método del código obtenerEvaluacion()

```
public int obtenerEvaluacion(int acertadas, int total) {
    int evaluacion;
    int acertadasPorciento = acertadas * 100 / total;

    if (acertadasPorciento >= 90) {
        evaluacion = 5;
        return evaluacion;
    } else if (acertadasPorciento < 90 && acertadasPorciento >= 80) {
        evaluacion = 4;
        return evaluacion;
    } else if (acertadasPorciento < 80 && acertadasPorciento >= 60) {
        evaluacion = 3;
        return evaluacion;
    } else {
        evaluacion = 2;
        return evaluacion;
    }
}
```

Luego de haberse construido el grafo se realiza el cálculo de la complejidad ciclomática mediante las tres fórmulas descritas a continuación, las cuales deben arrojar el mismo resultado para asegurar que el cálculo de la complejidad sea correcto.

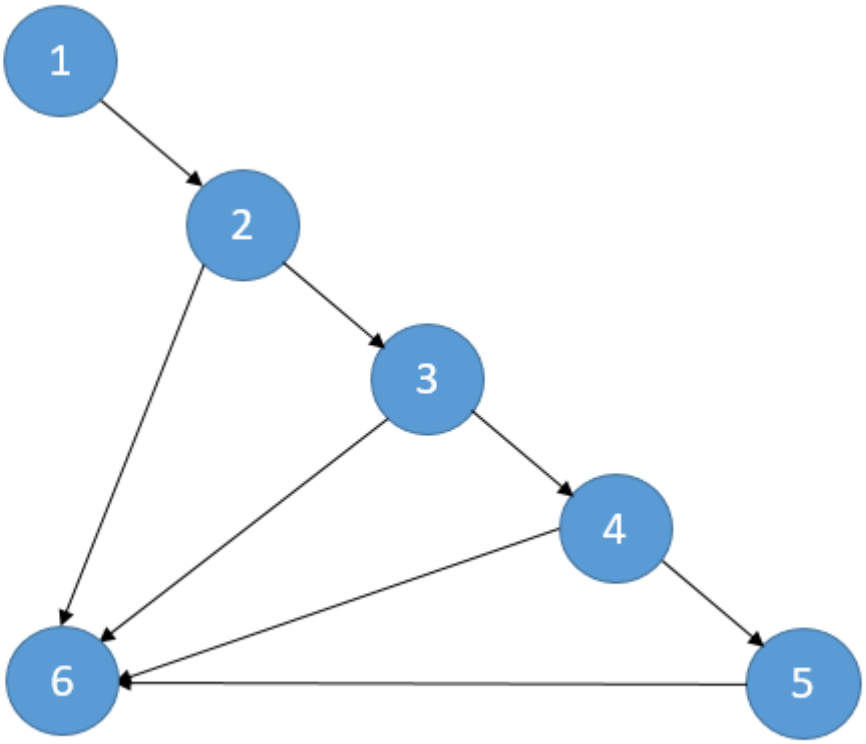
1. La complejidad ciclomática coincide con el número de regiones del grafo de flujo.
2. La complejidad ciclomática, V(G), de un grafo de flujo G, se define como:

   V(G) = Aristas - Nodos + 2.
3. La complejidad ciclomática, V(G), de un grafo de flujo G, también se define como: V(G) = Nodos de predicado + 1.

A partir del grafo de flujo del método obtenerEvaluacion() que se presenta en la ilustración 5, la complejidad ciclomática sería:

- Como el grafo tiene cuatro regiones, V(G) = 4.
- Como el grafo tiene 8 aristas y 6 nodos, V(G) = 8 – 6 + 2 = 4.
- Como el grafo tiene 3 nodos predicados, V(G) = 3 + 1 = 4.

Ilustración 5: Grafo de flujo del método obtenerEvaluacion().



Dado a que el cálculo de las tres fórmulas anteriormente mencionadas arrojó el mismo resultado se puede plantear que la complejidad ciclomática del método es 4. Esto significa que existen 4 posibles caminos por donde el flujo puede circular. Este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

Caminos básicos identificados:

- Camino 1: 1-2-6
- Camino 2: 1-2-3-6
- Camino 3: 1-2-3-4-6
- Camino 4: 1-2-3-4-5-6

Para cada camino básico determinado se realiza un diseño de caso de prueba.

CASO DE PRUEBA CAMINO 1	
Condición de ejecución	• acertadasPorciento > 90
Entrada	• evaluación = 5
Resultados esperados	Se retorna el número 5.

CASO DE PRUEBA CAMINO 2	
Condición de ejecución	• acertadasPorciento < 90 • acertadasPorciento >= 80
Entrada	• evaluación = 4
Resultados esperados	Se retorna el número 4.

CASO DE PRUEBA CAMINO 3	
Condición de ejecución	• acertadasPorciento < 80 • acertadasPorciento >= 60
Entrada	• evaluación = 3
Resultados esperados	Se retorna el número 3.

CASO DE PRUEBA CAMINO 4	
Condición de ejecución	• acertadasPorciento < 60
Entrada	• evaluación = 2
Resultados esperados	Se retorna el número 2.

Una vez ejecutados los casos de prueba, se llegó a la conclusión que todos los caminos básicos identificados fueron probados satisfactoriamente, demostrando que no existe código innecesario.

3.2.2. Pruebas Funcionales (caja negra)

Las pruebas de caja negra, también denominadas, pruebas de comportamiento, se concentran en los requisitos funcionales del software. Permiten derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales del programa (PRESSMAN 2005).

La prueba de caja negra intenta encontrar errores de las siguientes categorías (PRESSMAN 2005):

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Técnica de partición de equivalencia.

La partición equivalente es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba (PRESSMAN 2005).

Los casos de pruebas que se diseñan para este tipo de técnica se basan en una evaluación de las clases de equivalencia para una condición de entrada.

Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada (PRESSMAN 2005).

Las clases de equivalencia se definen de acuerdo a las siguientes directrices (PRESSMAN 2005):

- Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.

- Si una condición de entrada requiere un valor especifico, se define una clase de equivalencia válida y una no válida.
- Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.
- Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida
- Al aplicar estas directrices para la derivación de clases de equivalencia, se desarrollan y se ejecutan los casos de prueba para cada objeto de los datos del dominio de entrada.

Para realizar esta prueba se le entregó la descripción de casos de prueba y el software al cliente. Con la revisión fueron detectados un total de 5 no conformidades en la 1era iteración, 3 de funcionalidad, 1 de validación y 1 en el diseño y 0 en la segunda iteración.

A continuación, se muestra un ejemplo de caso de prueba aplicado a el requisito Identificar Estudiante.

Tabla 20: Descripción del caso de prueba del requisito Identificar estudiante

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de Texto	No	Se escribe el nombre del estudiante que se quiere identificar en el sistema. Solo se aceptarán letras.
2	Apellidos	Campo de Texto	No	Se escriben los apellidos del estudiante que se quiere identificar en el sistema. Solo se aceptarán letras.

3	Grupo	Campo de texto	No	Se escribe el grupo del estudiante que se quiere identificar en el sistema. Solo aceptará números

Escenario	**Descripción**	**Variable 1**	**Variable 2**	**Variable 3**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Inserta nombre, apellidos y grupo en el sistema	Identifica satisfactoriamente el estudiante en el sistema	V lucio	V pérez martinez	V 3502	Se inserta satisfactoriamente el nombre, apellidos y grupo	Selecciona la opción aceptar
EC 1.2 Los datos son incorrectos	Introduce datos incorrectos	I 1123##asdd	I rrerwe42343	I asdsd	Identifica que los datos son incorrectos, no inserta los datos y notifica	Selecciona la opción aceptar
EC 1.3 Existen campos obligatorios vacíos	Deja campos en blanco	V lucio	I	I	Notifica que existen campos obligatorios vacíos	Selecciona la opción aceptar
EC 1.4 La operació	La operación es cancelada	N/A	N/A		Cierra la interfaz	Selecciona la opción

n es cancelada						cancelar

3.3. Pruebas de concepto

Una prueba de concepto en el contexto del desarrollo de software viene a ser la validación de un aspecto funcional o no funcional de un sistema de información o de parte de él, ya sea por el área técnica o por el área usuaria (BERMUDEZ TALAVERA 2013).

En este trabajo es necesario aplicar pruebas de conceptos para comprobar la funcionalidad del resultado de la investigación, debido a que no existe ninguna aplicación que pueda proveer los servicios necesarios para ello. Por lo que se implementarán varios servicios webs para el consumo del resultado de la investigación.

3.3.1. Tecnologías de desarrollo

Servidor web

Node.js 10.1.0

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente (NODE.ORG).

Lenguaje de desarrollo

Express 4.16.3

Express es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles. Con miles de métodos de programa de utilidad HTTP y middleware (lógica de intercambio de información entre aplicaciones) a su disposición, la creación de una API sólida es rápida y sencilla. Express proporciona una delgada

capa de características de aplicación web básicas, que no ocultan las características de Node.js que tanto ama y conoce (EXPRESSJS.COM).

PostgreSQL 9.4.1

Es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD (Berkeley Software Distribution, en español Distribución de Software Berkeley es una licencia de software libre permisiva) y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más utilizado del mercado. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando (POSTGRESQL.ORG).

Dentro de las principales características que identifican a PosgreSQL están (POSTGRESQL.ORG):

- Es una base de datos 100% ACID (Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad).
- Integridad referencial.
- Copias de seguridad en caliente (Online/hot backups).
- Unicode.
- Juegos de caracteres internacionales.
- Regionalización por columna.
- Múltiples métodos de autentificación.
- Acceso encriptado vía SSL.
- Completa documentación.
- Multiplataforma.

3.3.2. Requisitos del sistema

Luego de la aplicación de las técnicas para la obtención de los requerimientos del sistema se obtuvieron los siguientes.

Tabla 21: Requisitos de los servicios web

Requisitos Funcionales
RF#1_ Gestionar pregunta
RF#2_ Gestionar estudiante
RF#3_ Obtener respuestas por estudiante
RF#4_ Enviar cantidad de aciertos por estudiante

Conclusiones del capítulo

Se realizó la validación de la investigación primeramente mediante las pruebas de aceptación por parte del cliente comprobando la satisfacción del mismo con la aplicación desarrollada. Se realizaron pruebas de caja negra y caja blanca que evaluaron y comprobaron la operatividad de las diferentes funcionalidades. Se realizaron las pruebas de conceptos permitiendo validar la funcionalidad de la aplicación Android en el entorno en que debe ser utilizada.

Conclusiones generales

La realización de este trabajo responde a la necesidad de buscar una solución al problema planteado. Terminada la investigación se llegó a las siguientes conclusiones:

- Se estableció el marco teórico de referencia para la investigación en el cual se obtiene un mejor dimensionamiento del problema a partir del análisis de los principales conceptos asociados a la solución, tales como software, aplicación, requisito de alto nivel, y metodología de desarrollo. De este último concepto se decidió seguir un enfoque ágil para su desarrollo, seleccionando la metodología XP para guiar el proceso, presentando los artefactos y adaptaciones a la metodología que se hicieron teniendo en cuenta las condiciones del equipo de proyecto.
- Se realiza el análisis, diseño e implementación de la aplicación mediante la elaboración de los artefactos correspondientes a los flujos de trabajo indicados por la metodología de desarrollo XP, creando así la documentación necesaria de la investigación que sirvió de guía a los desarrolladores para la implementación de la solución. Se trataron los aspectos más significativos del proceso de implementación, donde se mostró el Plan de Iteraciones y el estándar de codificación empleado en la solución lo que permitió realizar las diferentes entregas al cliente y estandarizar el código del sistema para la realización de futuros mantenimientos. Se obtuvo la implementación de la aplicación, dando solución a los requisitos funcionales identificados.
- Se realizó la validación de la investigación primeramente mediante las pruebas de aceptación por parte del cliente comprobando la satisfacción del mismo con la aplicación desarrollada. Se realizaron pruebas de caja negra y caja blanca que evaluaron y comprobaron la operatividad de las diferentes funcionalidades. Se realizaron las pruebas de conceptos permitiendo validar la funcionalidad de la aplicación Android en el entorno en que debe ser utilizada.

4. Referencias

1. ABRAHAMSSON, P. *Agile software development of mobile information systems*. International Conference on Advanced Information Systems Engineering, Springer, 2007. 1-4 p.
2. ---. *Keynote: Mobile software development–the business opportunity of today*. proceedings of the International Conference on Software Development, Citeseer, 2005. 20-23 p.
3. ALVAREZ, M. A., 2001. [Disponible en: http://www.desarrolloweb.com/articulos/497.php
4. AYA, R. and Á. ALEJANDRO Teléfonos móviles inteligentes en el aula de clase, una perspectiva de las ciberculturas juveniles, 2015.
5. BECK, K. and E. GAMMA. *Extreme programming explained: embrace change*. addison-wesley professional, 2000. p. 0201616416
6. BERMUDEZ TALAVERA, P., 2013. [Disponible en: https://es.slideshare.net/pbermudez10/prueba-de-concepto
7. CAMPOS, M. *Extreme Programming*, 2004. [Disponible en: http://apit.wdfiles.com/local--files/start/02_apit_xp_conceptos.pdf
8. COHN, M. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004. p. 0321205685
9. DEVELOPERS, G. *Conoce Android Studio*, 2018. [Disponible en: https://developer.android.com/studio/intro/
10. EXPRESSJS.COM. 2016. [Disponible en: http://expressjs.com/es/
11. FUENTES, H.; U. MESTRE*, et al.* Fundamentos didácticos para un proceso de enseñanza-aprendizaje participativo *Monografía. CeeS «MF Gran», Santiago de Cuba*, 1997.
12. GAMMA, E. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995. p. 8131700070
13. IEEE. *IEEE Standard 610.12-1990 Glossary of Software Engineering Terminology (Reaffirmed 2002)*, 2002. p.
14. JOSKOWICZ, J. Reglas y prácticas en eXtreme Programming *Universidad de Vigo*, 2008, 22.
15. LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos. Ed*, Pearson, 1999.

16. LEFF, A. and J. T. RAYFIELD. *Web-application development using the model/view/controller design pattern*. Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International, IEEE, 2001. 118-127 p. 076951345X
17. MAR-CORNELIO, O. and N. CAEDENTEY-MORENO Monitoreo energético en los laboratorios de la Universidad de las Ciencias Informáticas *Ingeniería Industrial*, 2016, 37(2): 190-199.
18. MOREIRA, M. A. Introducción a la tecnología educativa *DIM: Didáctica, Innovación y Multimedia*, 2010, (19): 1-78.
19. NODE.ORG. 2018. [Disponible en: https://nodejs.org/es/
20. POSTGRESQL.ORG. 2018. [Disponible en: https://www.postgresql.org/about/
21. PRESSMAN, R. S. Ingeniería de Software, 2005.
22. SALAVERT, I. R. and M. D. L. PÉREZ. *Ingeniería del software y bases de datos: tendencias actuales*. Univ de Castilla La Mancha, 2000. p. 8484270777
23. SOMMERVILLE, I. *Software engineering*. New York: Addison-Wesley, 2010. p.
24. TUBELLA, I. and J. V. REQUENA. *Sociedad del conocimiento*. Editorial UOC, 2005. p. 8497883144