Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

DISEÑO DE INVESTIGACIÓN PARA EL DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO

Francisco Rogelio Anzueto Marroquín
Asesorado por el MA. Ing. Héctor Alberto Heber Mendía Arriola

Guatemala, julio de 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO DE INVESTIGACIÓN PARA EL DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

FRANCISCO ROGELIO ANZUETO MARROQUIN
ASESORADO POR EL MA. ING. HECTOR ALBERTO HEBER MENDIA ARRIOLA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, JULIO DE 2020

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Christian Moisés de la Cruz Leal
VOCAL V	Br. Kevin Armando Cruz Lorente
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
EXAMINADOR	Ing. José Ricardo Morales Prado
EXAMINADOR	Ing. Oscar Alejandro Paz Campos
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DE INVESTIGACIÓN PARA EL DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO

Tema que me fuera asignado por la Dirección de Escuela de Estudios de Postgrado, con fecha 9 de marzo de 2020.

Francisco Rogelio Anzueto Marroquín

Teléfono: 2418-9142 / 24188000 ext. 1282
WhatsApp: 5746-9223
Email: información.eep@ing.usac.edu.gt
https://postgrado.ingenieria.usac.edu.gt

Ref. EEPFI-418-2020
Guatemala, 09 de marzo de 2020

Director
Carlos Gustavo Alonzo
Escuela de Ingeniería en Ciencias y Sistemas
Presente.

Estimado Ing. Alonzo:

Reciba un cordial saludo de la Escuela de Estudios de Postgrado. El propósito de la presente es para informarle que se ha revisado y aprobado el **DISEÑO DE INVESTIGACIÓN: DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO**, presentado por el estudiante **Francisco Rogelio Anzueto Marroquín** carné número **200611559**, quien optó por la modalidad del "PROCESO DE GRADUACIÓN DE LOS ESTUDIANTES DE LA FACULTAD DE INGENIERÍA OPCIÓN ESTUDIOS DE POSTGRADO". Previo a culminar sus estudios en la **Maestría en Artes en Tecnologías de la Información y la Comunicación**.

Y habiendo cumplido y aprobado con los requisitos establecidos en el normativo de este Proceso de Graduación en el Punto 6.2, aprobado por la Junta Directiva de la Facultad de Ingeniería en el Punto Décimo, Inciso 10.2 del Acta 28-2011 de fecha 19 de septiembre de 2011, firmo y sello la presente para el trámite correspondiente de graduación de Pregrado.

Sin otro particular,

Atentamente,

"Id y Enseñad a Todos"

Mtro. Héctor Alberto Heber Mendía Arriola
Asesor

Mtro. Marlon Antonio Pérez Türk
Coordinador de Área
Transferencia Tecnológica

Ing. Héctor Alberto H. Mendía Arriola
Ciencias y Sistemas
Col 10,057

Mtro. Edgar Darío Álvarez Coti
Director
Escuela de Estudios de Postgrado
Facultad de Ingeniería

DE GUATEMALA

FACULTAD DE INGENIERÍA

EEP-EICS-003-2020

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador y Director de la Escuela de Estudios de Postgrado, del Diseño de Investigación en la modalidad Estudios de Pregrado y Postgrado titulado: **DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO**, presentado por el estudiante universitaria **Francisco Rogelio Anzueto Marroquín**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería en esta modalidad.

ID Y ENSEÑAD A TODOS

Ing. Carlos Gustavo Alonzo
Director
Escuela de Ingeniería en Ciencias y Sistemas

DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Guatemala, abril de 2020

USAC
TRICENTENARIA

Decanato
Facultad de Ingeniería
24189101- 24189102

DTG. 153.2020.

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **DISEÑO DE INVESTIGACIÓN PARA EL DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO,** presentado por el estudiante universitario: **Francisco Rogelio Anzueto Marroquín,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Inga. Anabela Cordova Estrada
Decana

Guatemala, Julio de 2020

AACE/asga

ACTO QUE DEDICO A:

Dios Por darme una vida llena de bendiciones, fortaleza y consuelo en Jesús y el Espíritu Santo.

Mis padres Dina Marroquín, por siempre cuidarnos con amor, educarnos con paciencia y tener la fuerza y la valentía de sacarnos adelante sin importar las circunstancias. Rogelio Anzueto, por su amor, comprensión, sabios consejos y guiarme de la mejor manera en esta vida. No existen palabras para agradecerles por todo lo que han hecho por mí.

Mi abuelo Francisco Javier Marroquín Guzmán por ser un ejemplo de vida, de amor, de esfuerzo, de carácter, de trabajo y, sobre todo, por ser más que un padre para nuestra familia.

Mis hermanos Sussan y Daniel Anzueto Marroquín, por su apoyo y compañía durante mi vida. Por el apoyo incondicional en los buenos y malos momentos.

Mi familia	Porque todos son el mayor motivo de alegría que tengo en esta vida. En especial agradezco a Luis (q. e. p. d.), Sergio y Lorena Anzueto, por apoyarme en todos los momentos que lo necesité durante mis estudios.
Mis amigos y sus familias	Por todas las experiencias vividas dentro y fuera de la universidad. Agradezco todo su apoyo incondicional recibido y amistad sincera que han demostrado durante tantos años.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser la *alma mater* que me permitió nutrirme de conocimientos, valores y enseñanzas.
Facultad de Ingeniería	Por proporcionarme los conocimientos que me han permitido alcanzar esta meta profesional.
Mi asesor	MA. Ing. Héctor Alberto Heber Mendía Arriola, por haberme guiado durante el trabajo de graduación.
Familia y amigos en general	

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$	Dólar
=	Igual a
%	Porcentaje
Q	Quetzales
√	Raíz cuadrada
Σ	Sumatoria

GLOSARIO

Acoplamiento Forma y nivel de dependencia entre módulos de software.

Android Sistema operativo móvil desarrollado por Google, basado en Kernel de Linux.

Clean Arquitectura de software propuesta por Robert C. Martin.

Cohesión Grado que mide la fuerza de relación entre los módulos de software.

Dagger *Framework* para inyección de dependencias en Java y Android.

Firebase Plataforma para el desarrollo de aplicaciones *web* y móviles.

Fragmento Representa un comportamiento o una parte de la interfaz de usuario en Android.

Framework Estructura conceptual y tecnológica de soporte definido con módulos de software concretos que sirven de base para la organización y desarrollo de software.

FTP	Protocolo de red para la transferencia de archivos entre sistemas conectados a una red.
Git	Software de control de versiones.
Github	Herramienta *web* para alojar proyectos utilizando el sistema de control de versiones Git.
Google	Compañía principal subsidiaria de la multinacional estadounidense Alphabet Inc., cuya especialización son los productos y servicios relacionados con Internet, software, dispositivos electrónicos y otras tecnologías.
GPS	Sistema de posicionamiento global.
HTTP	El protocolo de transferencia de hipertexto es el protocolo de comunicación que permite las transferencias de información en la *web*.
Interfaces de software	Son programas o parte de ellos, que permiten expresar las órdenes a la computadora o visualizar su respuesta.
iOS	Es un sistema operativo móvil de la multinacional Apple Inc.
Java	Lenguaje de programación y plataforma informática.

JSON	Formato de texto sencillo para el intercambio de datos.
Kotlin	Lenguaje de programación de tipos estáticos que corre sobre la máquina virtual de Java y que también puede ser compilado a código fuente de JavaScript.
MVC	Modelo Vista Controlador
MVP	Modelo Vista Presentador
MVVM	Modelo Vista Modelo de Vista.
NDK	*Native Development Kit*.
Prueba unitaria	Es una forma de comprobar el correcto funcionamiento de una unidad de código.
Refactorización	Técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.
REST	Es un estilo de arquitectura software para sistemas hipermedia distribuidos como la *World Wide Web*.
SDK	Un kit de desarrollo de software es un conjunto de herramientas de desarrollo de software que permite a un desarrollador de software crear una aplicación informática para un sistema concreto.

SMTP	Protocolo para transferencia simple de correo electrónico.
Software	Comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas.
SQLite	Sistema de gestión de bases de datos relacional, contenida en una biblioteca relativamente pequeña biblioteca escrita en lenguaje C.
Teléfonos Inteligentes	Es un tipo de ordenador o computadora de bolsillo con las capacidades de un teléfono móvil/celular
VIPER	*View, Interactor, Presenter, Entity y Router*. Arquitectura basada en principios de responsabilidad única.
Web	Conjunto de información que se encuentra en una dirección determinada de internet.
WSDL	Es un formato del XML que se utiliza para describir servicios *web*.
XML	Siglas en inglés de *eXtensible Markup Language*. Es un metalenguaje que permite definir lenguajes de marcas.

RESUMEN

Las tecnologías móviles son parte de la vida diaria de las personas en diferentes países alrededor del mundo. Este tipo de aplicaciones brinda diversos beneficios a los usuarios y se han convertido en un pilar de desarrollo en el ámbito de la tecnología afectando tanto a la información y la comunicación.

Debido a la naturaleza del software, en casi todos los sistemas es necesario tener una interfaz para interactuar con el mundo real. Construir una interfaz de usuario dentro de una aplicación resulta problemático pues disminuye la flexibilidad que esta tiene de migrar y también causa que la interfaz no pueda ser reutilizada por otras aplicaciones.

En general, el objetivo principal de una arquitectura de software es la separación de componentes y entre la mayoría de las arquitecturas que se proponen la diferencia son algunos detalles en las capas que dividen el software. Suelen tener una capa para la lógica de negocio y otra para interfaces.

El acoplamiento de código se define como una medida que expresa el nivel de interdependencia entre los módulos de un software. Un buen software tendrá un nivel bajo de acoplamiento y por el otro lado, un mal software tendrá un alto nivel de acoplamiento. Cuando los módulos del sistema se encuentran altamente acoplados provocan que un software sea difícil de mantener y especialmente difícil de modificar.

El presente diseño de investigación busca desarrollar una arquitectura móvil para Android que permita reducir el nivel de acoplamiento de código en una aplicación y permita la reutilización de sus componentes en otro proyecto alterno.

1. INTRODUCCIÓN

A pesar del gran avance que ha tenido la tecnología alrededor del mundo y de la evolución que han sufrido los lenguajes de programación, desarrollar un software aún puede ser una tarea compleja, principalmente al desarrollarlo en equipos grandes o distribuidos. Pero, gracias a la experiencia de muchos desarrolladores se han definido mejores prácticas y guías que ayudan a que este proceso sea menos complejo. Por esto es importante que los equipos conozcan y dominen estos temas.

Dentro de estas prácticas y guías se encuentra principalmente el tema de arquitectura de software. En una arquitectura se encuentran definidas las partes, la estructura y las interacciones de un programa. Esto permite trabajar de una manera ordenada y brinda muchas ventajas adicionales, como por ejemplo tener la habilidad de reutilizar el código que se desarrolla en módulos de otros programas.

El mercado de aplicaciones móviles ha mantenido un crecimiento constante en los últimos años principalmente por la cantidad de teléfonos inteligentes disponibles. Se estima que este crecimiento no se detendrá ni reducirá en los siguientes años, por lo que este tema es muy relevante en la industria tecnológica y el presente trabajo bajo ese contexto busca proponer una arquitectura para desarrollar aplicaciones móviles Android y asegurar una buena calidad tanto en el producto final como en el código desarrollado.

En el primer capítulo de antecedentes se hace una recopilación bibliográfica de libros, artículos científicos y estudios previos a nivel de maestría en los que se fundamenta la base para desarrollar la investigación.

En el segundo capítulo se describe la justificación de la investigación para mostrar la importancia y las razones que han motivado a desarrollar este trabajo. Desarrollar estudios investigativos para mejorar procesos en el área de desarrollo de software continúa siendo un aporte importante principalmente para las empresas y personas que se dedican a esta área profesionalmente.

En el tercer capítulo se describen los alcances de la investigación, tomando en cuenta las perspectivas: investigativa, técnica y de resultados. Aquí se define puntualmente cada alcance esperado del proyecto con relación a las tareas específicas que se desarrollarán para alcanzar el objetivo principal.

En el cuarto capítulo se encuentra el marco teórico donde se incluyen los conceptos relevantes y necesarios para implementar las aplicaciones que son incluidas dentro del estudio. Para asegurar la calidad en el código de una aplicación es necesario conocer las características del código desarrollado y se incluyen las técnicas necesarias para analizar estas características. La característica principal que la arquitectura propuesta busca mejorar es el acoplamiento de código entre cada módulo de una arquitectura. Al reducir este nivel, se incrementa la facilidad de reutilizar módulos en otros proyectos porque estos se vuelven independientes y agnósticos de otras capas en el modelo definido por una arquitectura.

En el quinto capítulo se desarrollará la presentación de resultados definiendo y delimitando todas las funcionalidades que incluirán las aplicaciones, así como las tecnologías que se utilizarán. Se desarrollarán dos versiones de una

aplicación de catálogo de comercios que servirán para tener un punto de comparación y realizar las mediciones sobre las características del código. También se implementará una aplicación de catálogo de museos de Guatemala en la cual se reutilizarán los módulos posibles de la aplicación de catálogo de comercios para demostrar que la arquitectura planteada lo permite. Luego se desarrollará un análisis sobre las características de cada aplicación midiendo las interacciones que ocurren entre sus módulos de acuerdo con cada arquitectura utilizada para finalmente comparar el nivel de acoplamiento que existe y así demostrar la efectividad obtenida.

Finalmente, en el sexto capítulo se presentará una discusión de resultados de acuerdo con el análisis realizado y los datos obtenidos. Se desarrollarán puntos para describir el impacto tecnológico y económico del estudio, terminando al presentar acciones que se pueden desarrollar en futuras investigaciones.

2. ANTECEDENTES

Gracias a los avances tecnológicos que se han alcanzado, es muy común para las personas estar rodeadas de diferentes aparatos y dispositivos electrónicos. En los últimos años se ha logrado percibir un alto crecimiento en la demanda para desarrollar software que funcione en estos dispositivos, entre lo que se destaca las aplicaciones para teléfonos inteligentes.

En el año 2018 se descargaron 205.4 billones de aplicaciones móviles alrededor del mundo (Clement, 2019). Se estima que para el año 2022 se descargarán 258.2 billones, pues ha sido una estadística que no ha detenido su crecimiento en los recientes años. Aunque se muestran datos que indican un crecimiento en la cantidad de aplicaciones disponibles y descargadas, también se muestra que el indicador de retención ha disminuido de 38 % a 32 %, entre 2018 y 2019 respectivamente. Esto indica que la cuarta parte de las aplicaciones móviles son utilizadas únicamente una vez luego de haber sido descargadas. Google Play es la tienda de aplicaciones para Android y es la que obtiene la mayor cantidad de descargas. En la segunda posición de descargas, se encuentra la tienda App Store para dispositivos de Apple, pero esta domina en primer lugar con un 64 % el mercado en términos de usuarios que hacen algún gasto monetario en las aplicaciones.

Para lograr que un usuario utilice una aplicación constantemente es necesario que esta cuente con un nivel de calidad alto y presente la menor cantidad de fallos posibles. Debido a esto las aplicaciones necesitan ser actualizadas constantemente para brindar una mejor experiencia. Existen aplicaciones móviles Android que han sido desarrolladas utilizando patrones de

arquitectura que permiten que el código esté altamente acoplado a la capa de presentación. Para realizar cambios en estas aplicaciones o reutilizar sus módulos es necesario realizar un gran esfuerzo porque cualquier cambio es capaz de afectar distintas capas de su arquitectura y puede volverse un proceso complejo debido a que las clases del sistema operativo Android pueden estar a su vez muy relacionadas a un ciclo de vida del hilo principal de ejecución.

Tener altos niveles de acoplamiento en una aplicación está altamente relacionado con una baja productividad en los equipos, mayor necesidad de volver a elaborar partes y realizar un mayor esfuerzo para rediseñar componentes. Es necesario contar con herramientas que ayuden a medir el nivel de acoplamiento en una aplicación porque se obtienen métricas que colaboran a la toma de decisiones de diseño y ayudan en el proceso de refactorización de un programa pues evidencian en qué capas es necesario realizar este esfuerzo (Alghamdi, 2008).

Los desarrolladores de aplicaciones móviles que trabajan utilizando un ambiente de desarrollo Java, presentan dificultades para adoptar buenas prácticas de desarrollo. Esto aplica en proyectos de aplicaciones Android al ser Java uno de sus lenguajes principales soportados. Ejemplo de esto son las pruebas unitarias, pues al comparar con proyectos desarrollados utilizando otros lenguajes se puede observar que el nivel de código cubierto por pruebas suele ser menor. Los desarrolladores han expresado que existe un nivel mayor de dificultad para desarrollar pruebas unitarias en aplicaciones móviles muy relacionadas a interfaces de usuario, y también concluyen que afectan factores como la inexperiencia y dominio de aplicación de los equipos (Abrahamsson, Hanhineva y Jäälinoja, 2005).

Distintas técnicas han sido propuestas para medir el nivel de acoplamiento de código de un software. Una muy común es medir el nivel de acoplamiento a través de propiedades estructurales y análisis estático de código. Esto es complicado para lenguajes que incluyen polimorfismo, como lo es Java en Android, o partes de código que no está siendo utilizado. Existen técnicas de análisis dinámico para software basado en lenguajes orientados a objetos y demuestran que existe una alta relación entre las medidas del nivel de acoplamiento de un software con la propensión al cambio de cada clase implementada (Arisholm, Briand y Foyen, 2004).

Existen diversos estudios que se relacionan con la calidad en el proceso de desarrollo de software y se enfocan tanto al proceso de desarrollo, como al producto final. Al hablar de calidad en un software se pueden distinguir tres entidades: procesos, productos y recursos, y generalmente, no es posible mejorar los recursos existentes, principalmente porque dependen de las personas que integran los equipos de desarrollo (Calero, Moraga y Piattini, 2010). Esto ayuda a evidenciar que es necesario invertir tiempo para mejorar el producto final invirtiendo en el diseño de la arquitectura de una aplicación. Esto debe fundar una base para que los desarrolladores puedan guiarse durante la implementación.

Existen diferentes diseños de arquitectura aplicables en Android que tienen como propósito principal realizar una total separación de las responsabilidades que posee el código en distintas capas. Con esto buscan asegurar que se haga una distinción totalmente clara entre los roles de cada una de estas capas (Cheng y Olivares, 2018). Por ejemplo, en el patrón MVVM, se hace un énfasis muy fuerte en que la capa Modelo de Vista no debe contener ninguna referencia a la Vista. Esta capa debe estar encargada únicamente de proveer información y no tener ningún conocimiento en quien la consume. Se plantean diferentes patrones de

arquitectura que pueden ser utilizados para desacoplar código en aplicaciones Android.

Con base en los antecedentes presentados se evidencia que es necesario analizar, definir e implementar una arquitectura que se adecue correctamente a cada aplicación. Para poder reutilizar el código de una aplicación es necesario que esta se encuentre estructurada de manera correcta y que sus módulos sean altamente cohesivos y posean un bajo nivel de acoplamiento. Para asegurarlo se debe ser capaz de medir el nivel de acoplamiento del software e implementar una refactorización en la aplicación a través de una arquitectura que ayude a desacoplar las capas principales, como lo es la capa de presentación en las aplicaciones Android.

3. PLANTEAMIENTO DEL PROBLEMA

Desarrollar una aplicación Android es algo a lo que muchas empresas se sienten atraídas hoy en día por el alto índice de uso que tienen estos dispositivos. En general las aplicaciones Android suelen ser desarrolladas por equipos pequeños y en ocasiones con poca experiencia. Esto causa que algunos proyectos de desarrollo no sean completados satisfactoriamente y que los clientes no queden completamente satisfechos.

El problema principal que enfrentan los desarrolladores de este tipo de aplicaciones cuando poseen poca experiencia, es que escriben el código sin haber diseñado una arquitectura a la cual regirse para asegurar una buena calidad del software. Conforme los proyectos avanzan, el código suele estar demasiado acoplado a la capa de presentación y esto eleva el nivel de complejidad en cualquier modificación que quiera realizarse, provocando que existan atrasos en las entregas y que prácticas importantes para el aseguramiento de la calidad no sean incluidas en los procesos de desarrollo.

La documentación, el desarrollo de pruebas unitarias, implementar módulos independientes, entre otros, son prácticas que se utiliza para mejorar la calidad en el código y en el producto final en un desarrollo de software. Los desarrolladores de aplicaciones móviles Android suelen dejar estas prácticas a un lado ya que el código suele estar muy acoplado a la capa de presentación y esto complica su implementación. Es importante mejorar el diseño de la arquitectura de una aplicación móvil para facilitar su desarrollo, mantenimiento y actualización, permitiendo adoptar las mejores prácticas a los desarrolladores.

Una causa importante de este problema es que Google, empresa que desarrolla el SDK de Android, no tenía definida ni sugerida formalmente una arquitectura robusta que ayudara a los desarrolladores a mejorar este aspecto en las aplicaciones, sino hasta el año 2017, cuando incluyó como parte de su SDK nuevos componentes de arquitectura y definió una arquitectura como sugerencia para los desarrolladores Android (Documentación oficial Android 2019).

Por esta razón existen aplicaciones desarrolladas actualmente que no cuentan con una arquitectura robusta que ayude a desacoplar el código. Anteriormente los desarrolladores simplemente seguían un patrón MVC, debido a su simpleza y popularidad. El problema de seguir este patrón es que en las aplicaciones Android tanto la capa del modelo como la capa del controlador se comunican de alguna manera con la vista (presentación) y conforme las aplicaciones evolucionan, esto eleva su nivel de acoplamiento. Por esta razón es necesario diseñar una arquitectura que aísle correctamente la capa de presentación de las demás para desacoplar el código y mejorar su calidad.

Google sugiere utilizar una arquitectura Modelo-Vista-Modelo de vista (MVVM) para desarrollar una aplicación Android. (Guía oficial Android, 2019). Es un diseño que se puede aplicar muy bien a la mayoría de las aplicaciones, pero es necesario mencionar que esta no es la única arquitectura que se puede utilizar para resolver los problemas de acoplamiento en una aplicación móvil Android. Existen otros modelos definidos que se han vuelto muy populares en el desarrollo de aplicaciones móviles como Modelo-Vista-Presentador (MVP), arquitectura Clean, arquitectura VIPER, entre otros. Es necesario evaluar cada aplicación y el equipo de desarrollo para definir correctamente la arquitectura que se debe seguir.

Un claro ejemplo de una aplicación que se encuentra fuertemente acoplada es la aplicación móvil Android que funciona como un catálogo de comercios en Guatemala. Fue desarrollada en el año 2014 utilizando un patrón Modelo-Vista-Controlador (MVC) y actualmente resulta demasiado complejo reutilizar la vista para construir una nueva aplicación de catálogo orientada a museos de Guatemala.

Para lograrlo se deben utilizar diferentes servicios remotos y nuevas estructuras de datos para almacenamiento local en el dispositivo, lo que genera un alto nivel de complejidad debido al alto nivel de acoplamiento que existe actualmente. Es necesario implementar una arquitectura robusta que permita desacoplar la capa de presentación y desarrollar nuevos módulos independientes y cohesivos que puedan ser reutilizados por la nueva aplicación.

Con base en la descripción del problema planteada anteriormente se define la pregunta principal:

- ¿Cómo reducir el nivel de acoplamiento en la aplicación móvil Android de catálogo de comercios y facilitar la reutilización de código al implementar la aplicación de catálogo de museos?

Adicionalmente se definen las preguntas orientadoras:

- ¿Qué arquitectura debe ser implementada para poder reutilizar el código de la aplicación de catálogo de comercios?

- ¿Cómo medir el nivel de acoplamiento del código de la aplicación de catálogo de comercios para garantizar su eficiencia?

- ¿Qué herramientas permiten la reutilización de módulos en diferentes aplicaciones móviles Android?

4. JUSTIFICACIÓN

La elaboración de este trabajo seguirá la línea de investigación de sistemas para impulsar la adopción de tecnología móvil en los negocios.

Este trabajo contribuye a solucionar de manera efectiva el problema presentado sobre el alto nivel de acoplamiento en el código que suelen tener las aplicaciones móviles Android. La arquitectura propuesta reducirá el nivel de acoplamiento entre las distintas capas sugeridas y brindará la habilidad de desarrollar módulos independientes. Mediante la implementación de técnicas para medir el nivel de acoplamiento se asegurará que cada módulo desarrollado sea altamente cohesivo y pueda ser reutilizado en diferentes proyectos.

Debido al alto nivel de acoplamiento que suelen tener aplicaciones móviles que implementan un patrón MVC tradicional han sido propuestas arquitecturas más robustas como MVVM, MVP, Clean o VIPER, entre otras, para solucionar este problema. La utilización de estas arquitecturas ayuda a dividir una aplicación en múltiples capas que pueden ser desarrolladas incluso como módulos independientes. Se busca que cada capa diseñada logre tener una función específica por sí misma al no tener conocimiento alguno de las demás capas de la arquitectura.

Es relevante mencionar que la industria de desarrollo de software es un sector que se encuentra en constante cambio. Los lenguajes de programación son redefinidos en ocasiones en sus diferentes versiones y surgen lenguajes nuevos como alternativas conforme las tecnologías avanzan. Cada año Google, como también lo hacen otras empresas, presenta sus nuevas tecnologías

desarrolladas y busca brindar mejoras a las herramientas de desarrollo de aplicaciones móviles Android. Por esto es importante diseñar una arquitectura que facilite el mantenimiento de las aplicaciones, ya que se encuentran en un ecosistema en el que eventualmente se ven forzadas a necesitar cambios o actualizaciones. Además, es común en el desarrollo de software que los requerimientos cambien y sea necesario realizar una actualización.

Mejorar la calidad de un software beneficia, tanto a los usuarios finales, como también a las personas o empresas propietarias de cada aplicación. Implementar una arquitectura en una aplicación ya desarrollada mejorará la posibilidad de reutilizar sus módulos en diferentes aplicaciones, como también facilitará su proceso de actualización.

La descripción de técnicas para la medición de acoplamiento en un software y la utilización de herramientas para la reutilización de módulos son beneficios para el lector del presente trabajo porque evidenciarán los beneficios de utilizarlos dentro de un proyecto real.

En Guatemala existen compañías de software que desarrollan aplicaciones móviles Android. Esta investigación se presenta como un aporte a estas empresas describiendo prácticas que pueden ser imitadas para mejorar sus procesos de desarrollo. Se brindará un soporte teórico para comprobar la mejora de las aplicaciones mediante la implementación de la arquitectura desarrollada.

Mediante la implementación de la aplicación de catálogo de los museos de Guatemala, las personas que deseen buscar información de estos lugares se verán beneficiados al tener información al alcance en los dispositivos móviles Android. Una aplicación móvil puede colaborar a atraer una mayor cantidad de

personas a los museos, por lo que estos se verán beneficiados al tener un canal más de difusión de información.

5. OBJETIVOS

5.1. General

Desarrollar una arquitectura móvil para Android que permita reducir el acoplamiento de código en la aplicación de catálogo de comercios y validar su eficiencia mediante la reutilización de código al implementar la aplicación de catálogo de museos de Guatemala.

5.2. Específicos

- Implementar una arquitectura móvil para Android que permita el desarrollo de módulos independientes en la aplicación de catálogo de comercios.

- Describir e implementar técnicas para medir el nivel de acoplamiento en el código de la aplicación de catálogo de comercios y validar la eficiencia en la implementación de la arquitectura propuesta.

- Emplear las herramientas que permitan la reutilización de módulos de la aplicación de catálogo de comercios en la aplicación de catálogo de museos.

6. NECESIDADES A CUBRIR Y ESQUEMA DE SOLUCIÓN

El desarrollo de software es un área inherentemente relacionada al tema de tecnologías de información y comunicación. Una persona que se dedica a estudiar o ejercer en esta área debe tener amplios conocimientos en el tema de arquitectura de software para sobresalir en este ámbito. En Guatemala, gracias al crecimiento que ha tenido la utilización de dispositivos móviles inteligentes, existen muchas oportunidades profesionales para desempeñarse como desarrollador, diseñador o arquitecto de aplicaciones móviles. El presente trabajo pretende cubrir esta necesidad mediante el desarrollo de una aplicación móvil Android que demuestre la importancia de utilizar una arquitectura adecuada y de asegurar un bajo nivel de acoplamiento entre las capas diseñadas del software.

Actualmente existen proyectos de emprendimiento, tanto en la ciudad capital como en el interior de la República de Guatemala, que desarrollan aplicaciones móviles Android y son implementados por una persona o por un equipo pequeño. Estos proyectos pueden estar siendo desarrollados siguiendo un patrón de arquitectura MVC por ser este un patrón muy popular y que fue previamente sugerido por Google. Esto puede causar, a medida que los proyectos crecen, que se enfrenten a muchas dificultades para para ser actualizados. La presente investigación busca apoyar a estos desarrolladores brindando una solución de arquitectura aplicable desde el inicio de una implementación que permita escalar y reutilizar el código desarrollado.

El presente estudio pone en práctica los conocimientos de arquitectura de software discutidos como parte del programa de maestría en tecnologías de

información. Desarrollar mejoras en una arquitectura de software nueva o existente es una habilidad que cada participante del programa adquiere.

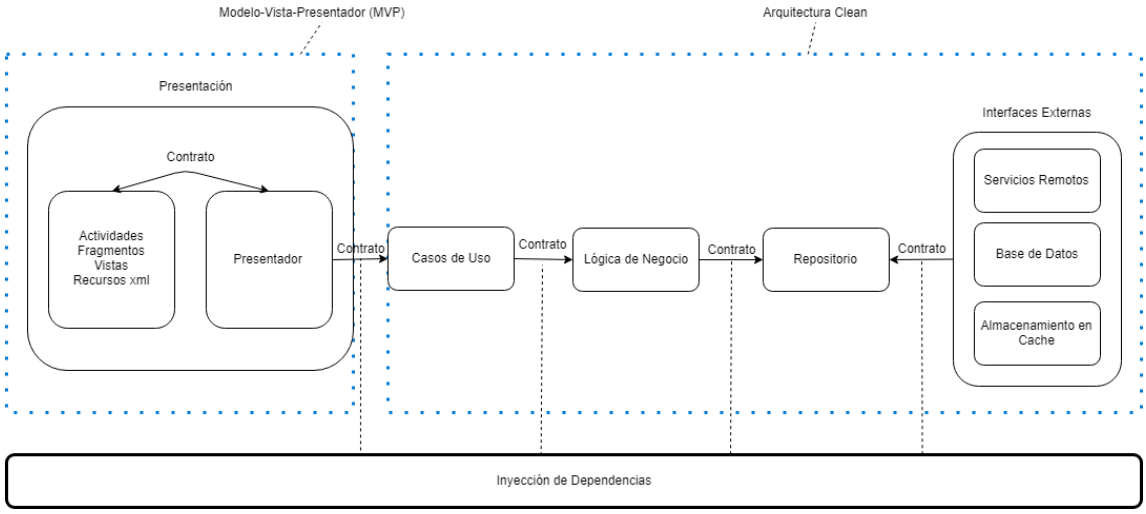
Como parte de la solución al problema de acoplamiento, se desarrollará una arquitectura que asegure reducir el nivel de acoplamiento en el código. Para lograrlo, la arquitectura propuesta estará basada en dos modelos actuales de arquitectura: arquitectura Modelo-Vista-Presentador (MVP) y arquitectura *Clean*. Ambas han sido diseñadas para lograr estructurar el código de una mejor manera y brindan fortalezas sobre las cuales se puede fundamentar una nueva arquitectura innovadora para crear componentes cohesivos que puedan ser reutilizados.

Por un lado, MVP ayuda a tener una capa de Presentación que separa las clases propias del SDK de Android mediante la creación de un Presentador que se encarga de coordinar la comunicación con los componentes de vista mediante la utilización de contratos. Esto ayuda a mantener la capa de vista aislada y mantener la lógica de la vista en una subcapa independiente.

La arquitectura Clean propone una mayor segregación en las capas relacionadas a la lógica de negocios, comúnmente conocida como modelo en otras arquitecturas. Se enfoca en separar aún más las responsabilidades que puede tener cada capa y la manera en que se intercomunican. Esto colabora a que se tengan módulos altamente cohesivos debido a que se encuentran agnósticos de otras capas y pueden ser reutilizados o modificados independientemente.

Esto se comprobará mediante la reutilización de componentes en la aplicación de catálogo para museos. A continuación, se presenta el esquema de la arquitectura de la solución planteada:

Figura 1. **Esquema de solución**



Fuente: elaboración propia.

Del esquema anterior se describen a continuación módulos que serán abstraídos y desarrollados independientemente y que permitirán su reutilización en diferentes proyectos:

- Presentación: este módulo contendrá en una capa las vistas desarrolladas mediante el SDK de Android y en otra el presentador. Ambas capas se intercomunican mediante la definición de un contrato entre ellas. La capa de presentación contendrá la lógica para cada vista y es la única que se comunica con la capa casos de uso.

- Casos de uso: en este módulo se encapsulan las acciones que son desencadenadas por los usuarios en la capa de presentación y funciona como un controlador que es capaz de comunicarse con la capa que contiene la lógica de negocio.

- Lógica de negocio: este módulo contendrá la lógica de negocio y los modelos específicos de la aplicación. Se comunica únicamente con la capa de datos mediante contratos definidos.

- Repositorio: es un módulo encargado de realizar la definición abstracta de las acciones permitidas para obtener datos de cualquier fuente. Maneja el acceso a los *frameworks* externos sin tener conocimiento directo de ellos mediante la definición de contratos y haciendo uso de los conceptos de inversión de control e inversión de dependencias.

- Interfaces externas: se encapsulan los *frameworks* utilizados mediante definiciones de contratos abstractos para poder ser llamados desde el repositorio de datos sin que esta capa conozca explícitamente cuales son. Aquí se incluye cualquier biblioteca externa o marco de trabajo de Android que pueda ser reemplazado en un futuro.

En la capa de repositorio se mencionan los conceptos de inversión de control e inversión de dependencias. Estos conceptos son los que hacen posible que cada capa sea cohesiva y agnóstica a la implementación de las demás mediante la definición de contratos. Esto permite reducir altamente el acoplamiento entre el código de cada módulo desarrollado. Para facilitar la implementación de estos conceptos se utilizará un *framework* de inyección de dependencias.

7. MARCO TEÓRICO

7.1. Aplicaciones móviles

Las tecnologías móviles son parte de la vida diaria de las personas en diferentes países alrededor del mundo. Este tipo de aplicaciones brinda diversos beneficios a los usuarios y se han convertido en un pilar de desarrollo en el ámbito de la tecnología afectando tanto a la información y la comunicación. (Diaz, 2015).

Una aplicación móvil es un software que se puede ejecutar directamente en el dispositivo, como teléfonos, tabletas y otros dispositivos móviles, lo cual aporta diversas ventajas entre las cuales cabe destacar el acceso a muchos servicios que se conectan a través de internet. Otra ventaja importante es que las aplicaciones tienen la posibilidad de acceder a características del hardware y herramientas que brindan los sistemas operativos.

Se pueden descargar e instalar directamente en el dispositivo. Esto facilita su ejecución y mejora la experiencia del usuario al no necesitar acceder a la aplicación desde un navegador *web* y, en algunos casos, poder ser ejecutadas sin necesidad de acceder a internet.

7.2. Tipos de aplicación móvil

De acuerdo con la forma en que una aplicación móvil es desarrollada, estas pueden ser categorizadas en tres tipos.

7.2.1. Aplicaciones nativas

Los dispositivos móviles, comúnmente, vienen acompañados de un sistema operativo principal que permite instalar aplicaciones directamente en el dispositivo. Estas aplicaciones son diseñadas y desarrolladas utilizando el software que ofrece cada sistema operativo, al cual se le conoce como *Software Development Kit* (SDK) y permite escribir el código utilizando un lenguaje de programación específico.

Estas aplicaciones pueden ser publicadas en las tiendas de distribución de aplicaciones respectivas para cada sistema operativo y brindan algunas ventajas como la posibilidad de ser ejecutadas sin una conexión a internet y la facilidad de acceso a múltiples recursos del sistema.

7.2.2. Aplicaciones web

Son aplicaciones desarrolladas para ser ejecutadas desde un navegador *web*. Debido a la posibilidad que tienen los dispositivos móviles de ejecutar estos navegadores, se puede aprovechar este recurso para adaptar aplicaciones *web* existentes y brindar una experiencia distinta al ser accedidas desde un dispositivo móvil.

Su principal ventaja es que su desarrollo es independiente al sistema operativo en el que se ejecutará la aplicación pues no necesitan ser instaladas, pero dependen fundamentalmente de una conexión a internet. Debido a que dependen de un navegador para ser ejecutadas suelen tener menos facilidad de acceso a recursos del sistema y puede llegar a ser complejo el proceso de adaptar una *web* al formato móvil.

7.2.3. Aplicaciones híbridas

Se les denomina así a las aplicaciones que son una mezcla de aplicaciones nativas y aplicaciones *web*. Son desarrolladas utilizando patrones de diseño y lenguajes para desarrollar aplicaciones *web* como HTML, Javascript y CSS, pero se diferencian en que se compilan para poder ser ejecutadas de manera nativa.

Las herramientas para desarrollar aplicaciones híbridas brindan la posibilidad de compilar la misma aplicación hacia distintos sistemas operativos, siendo esta su principal ventaja por permitir desarrollar un solo proyecto que luego puede ser distribuido en distintas plataformas.

7.3. Sistema operativo Android

Android es un sistema operativo de código abierto, basado en Linux y desarrollado principalmente para correr en teléfonos móviles, aunque también se utiliza en tabletas, notebooks, reproductores de música e incluso en computadores personales (Báez, Borrego y Cordero, 2013).

Este sistema operativo se destaca junto con el sistema operativo iOS de la empresa Apple como los más utilizados en teléfonos inteligentes y otros dispositivos móviles alrededor del mundo. Gracias a la gran popularidad que han ganado ambos sistemas, la mayoría de las aplicaciones móviles que se desarrollan están relacionadas a estos dos sistemas principalmente.

7.4. Desarrollo de aplicaciones nativas Android

El desarrollo de aplicaciones móviles abarca todos los procesos involucrados en la escritura de un software cuyo objetivo es ser ejecutado en un dispositivo móvil. Este tipo de desarrollo se diferencia del desarrollo tradicional de software, que generalmente aprovecha los recursos del dispositivo móvil.

Android brinda la opción de desarrollar las aplicaciones en dos lenguajes principalmente:

- Java: es un lenguaje con muchos años de desarrollo el cual es orientado a objetos y se caracteriza por correr sobre una máquina virtual. Android soporta la versión 7 y 8 de este lenguaje (Google Inc., 2019).

- Kotlin: es un lenguaje de programación moderno multiplataforma que se caracteriza por ser de tipos estáticos y que corre sobre la máquina virtual de Java. Android soporta la versión 1.3, la cual es la última versión publicada de este lenguaje (Google Inc., 2019).

Android también brinda la opción de desarrollar utilizando bibliotecas de bajo nivel escritas en lenguajes C y C++ a través del *Native Development Kit* (Android NDK). Este tipo de desarrollo requiere otro tipo de conocimientos y procesos que no están relacionados con el propósito de la investigación.

Existen distintas herramientas y componentes que pueden utilizarse al desarrollar una aplicación para Android. A continuación, se describen los conceptos relacionados con la investigación y que son necesarios para su desarrollo.

7.4.1. Android Software Development Kit (Android SDK)

Contiene las herramientas necesarias que se utilizan para desarrollar aplicaciones Android. Incluye librerías requeridas, documentación de APIs, código de ejemplo, emuladores, entre otras herramientas y documentos útiles.

7.4.2. SQLite como almacenamiento de datos

Android brinda una API que permite manejar una base de datos SQLite la cuál únicamente puede ser accedida por la aplicación asociada. SQLite funciona como un motor de base de datos que posee características simples como el manejo de archivos de poco tamaño y que se ejecuta en la aplicación cliente sin necesidad de un servidor (Báez, Borrego y Cordero, 2013).

Las bases de datos SQLite pueden utilizarse para distintos propósitos como lo es el almacenamiento de información que suele ser accedida concurrentemente y con esto evitar la necesidad de realizar conexiones al servidor o para realizar una configuración inicial de datos, entre otros.

7.4.3. Uso del GPS

La posibilidad de utilizar el sistema de posicionamiento global (GPS) es una de las ventajas de un dispositivo móvil. Esto brinda a los usuarios beneficios de movilidad y se ha visto beneficiado por el crecimiento de las conexiones a internet en los dispositivos.

El desempeño del GPS en un teléfono móvil inteligente es ya comparado con el desempeño que brinda un dispositivo regular diseñado únicamente para esta función (Zandbergen, 2011).

7.5. Arquitectura de software

Debido a la naturaleza del software, en casi todos los sistemas es necesario tener una interfaz para interactuar con el mundo real, es decir, con un usuario. Suele requerir un esfuerzo constante mantener actualizadas estas interfaces para brindar una buena experiencia al usuario. Construir una interfaz de usuario dentro de una aplicación resulta problemático pues disminuye la flexibilidad que esta tiene de migrar y también causa que la interfaz no pueda ser reutilizada por otras aplicaciones. Esto es solo uno de los muchos ejemplos que se encuentran como las razones para diseñar una arquitectura de software.

En general, el objetivo principal de una arquitectura de software es la separación de componentes y entre la mayoría de las arquitecturas que se proponen la diferencia son algunos detalles en las capas que dividen el software. Suelen tener una capa para la lógica de negocio y otra para interfaces (Martin, 2017).

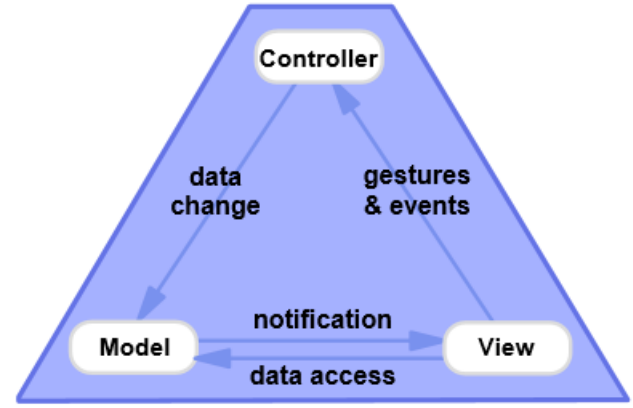
Los sistemas que se producen al seguir una arquitectura correctamente diseñada obtienen diferentes ventajas como:

- Independencia a marcos de trabajo.
- Facilidad de pruebas.
- Independencia de interfaz de usuario, base de datos y cualquier agente externo.

7.5.1. Arquitectura MVC

La arquitectura MVC aparece como una de las primeras respuestas para solucionar el problema de programar lógica del programa y lógica de presentación de manera conjunta. Divide el sistema de software en tres capas principales, como su nombre lo indica: Modelo, Vista y Controlador.

Figura 2. **Arquitectura MVC**



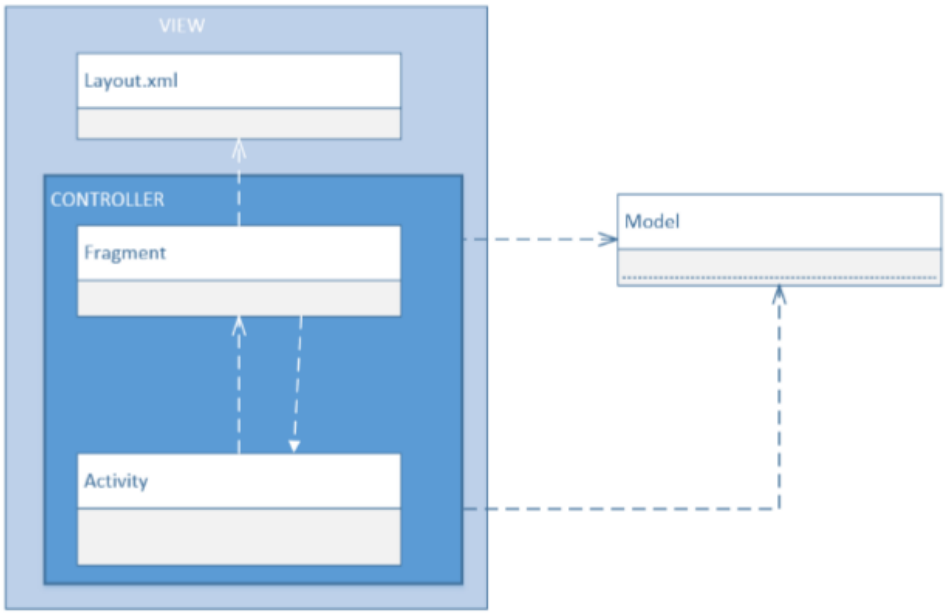
Fuente: Potel (1996). *MVP: Model-View-Presenter the Taligent programming model for C++ and Java.*

Al desarrollar un diseño perfecto de esta arquitectura se obtiene un producto en el que tanto la vista como el controlador se encuentran siempre en pares y están altamente acoplados. Es decir, una vista tiene un controlador específico y un controlador tiene la responsabilidad de una sola vista. (Lou, 2016)

Esta es la arquitectura de una aplicación Android que por defecto se genera al crear un proyecto en Android Studio. El modelo suele ser un modelo plano de Java o una clase separada responsable de manejar la lógica de negocio que incluye tareas como obtener datos remotos o de servicios locales, como un archivo o una fotografía.

La vista es representada por un archivo XML y su actividad o fragmento relacionado. El controlador es generado dentro de cada actividad o fragmento. Debido a esto no se obtiene una separación claramente definida entre estas capas. En la Figura 3 puede observarse el acoplamiento que existe entre la vista y el controlador.

Figura 3. **Arquitectura MVC en Android**

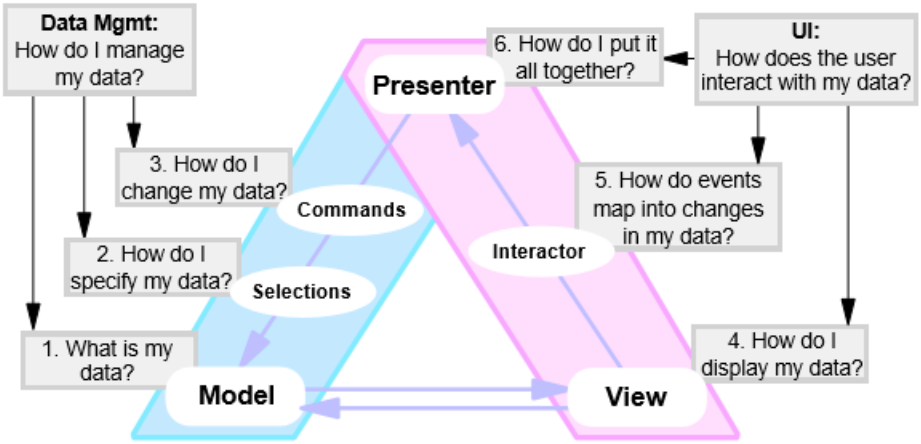


Fuente: Lou (2016). *A Comparison of Android Native App Architecture–MVC, MVP and MVVM.*

7.5.2. Arquitectura MVP

Los principales componentes de esta arquitectura, como su nombre lo indica son: Modelo, Vista y Presentador. Además, esta arquitectura reconoce tres componentes más: Selecciones, Comandos e Interactor. Nace principalmente para responder a dos requerimientos principales: La gestión de los datos y el control del código relacionado a la interfaz de usuario.

Figura 4. **Arquitectura MVP**

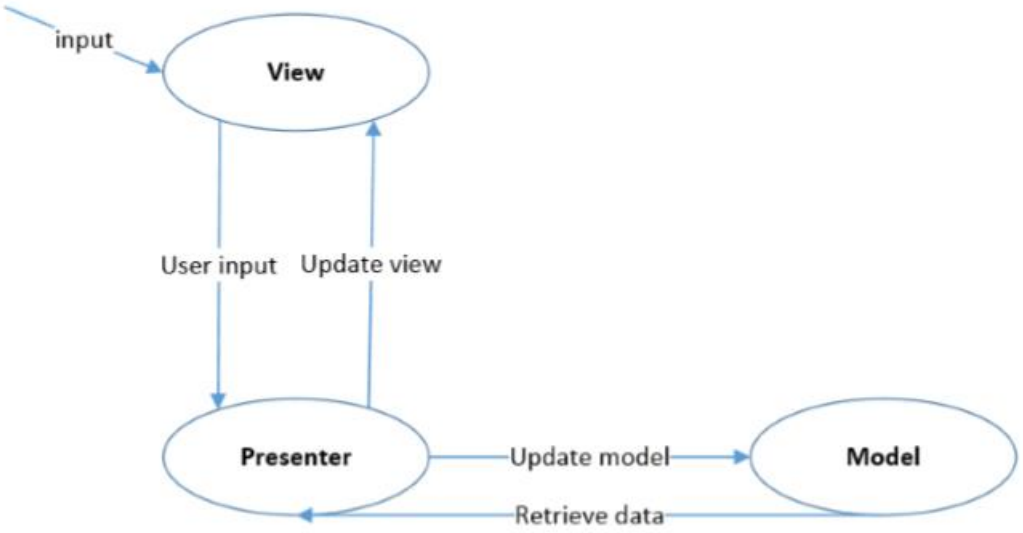


Fuente: Potel (1996). *MVP: Model-View-Presenter the Taligent programming model for C++ and Java.*

Cada componente definido en esta arquitectura se encuentra ligeramente acoplado y tiene responsabilidades propias definidas claramente. Según la necesidad de cada proyecto, se suelen utilizar únicamente las tres capas principales, implementando las responsabilidades de las Selecciones, Comandos e Interactor dentro del Presentador.

Al aplicar esta arquitectura en Android se reduce altamente el acoplamiento que se suele tener entre la vista y el modelo porque la vista no tiene conocimiento alguno del modelo. Ver figura 5.

Figura 5. **Interacción de componentes MVP Android**



Fuente: Lou (2016). *A Comparison of Android Native App Architecture–MVC, MVP and MVVM.*

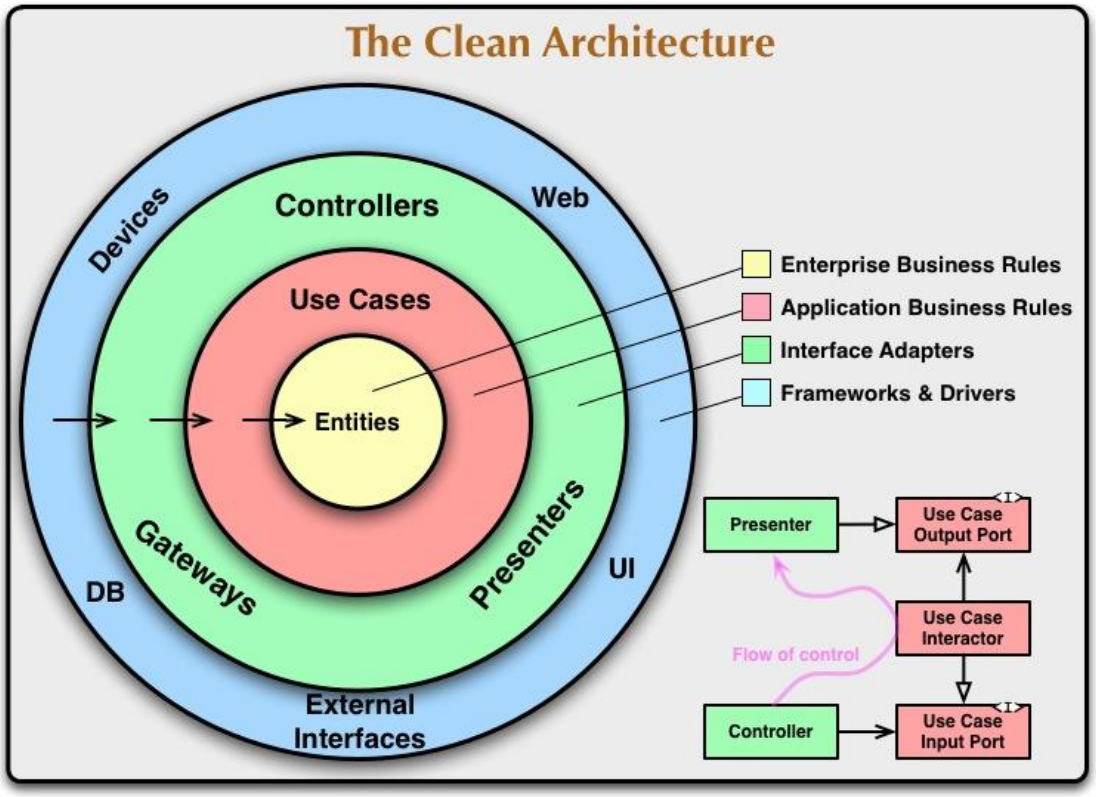
Se ha demostrado que la independencia del modelo permite a los desarrolladores cambiar y evolucionar las estructuras de datos y formatos de archivos a su beneficio sin cambiar nada sobre como los datos son desplegados o procesados en el resto del programa y a su vez lograr introducir persistencia y acceso remoto de datos (Potel, 1996).

7.5.3. Arquitectura Clean

La arquitectura Clean ha sido definida basándose en ideas de otras arquitecturas como la Arquitectura Hexagonal (*Hexagonal Architecture*), la Arquitectura Cebolla (*Onion Architecture*), entre otras. Busca realizar una integración de estas ideas para unirlas en una sola idea.

Se presenta el diagrama de la Arquitectura Clean en la figura 6. Los círculos representan las capas del software y en general, entre más al centro se encuentra una capa se está desarrollando software de alto nivel.

Figura 6. **Arquitectura Clean**



Fuente: Martin (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition.*

La idea central es La Regla de Dependencia que indica que las dependencias de código únicamente pueden apuntar hacia adentro. De ninguna manera algo del círculo central puede conocer algo del círculo exterior. Para cruzar los límites de una capa, estas únicamente pueden comunicarse con la siguiente capa hacia adentro y se terminan ejecutando de regreso gracias al

concepto de inversión de dependencias en algunos lenguajes, lo que permite mantener el flujo correcto a través de los límites.

Los datos pueden fluir a través de estructuras de datos simples. Lo importante en todo momento es que estos datos se encuentren aislados y simples sin violar la regla de dependencia al fluir a través de sus límites. Por ejemplo, en la capa de casos de uso o de interfaz de usuario no deben utilizarse modelos que representen entidades de la base de datos.

7.6. Acoplamiento de código

Se define como una medida que expresa el nivel de interdependencia entre los módulos de un software. Un buen software tendrá un nivel bajo de acoplamiento y por el otro lado, un mal software tendrá un alto nivel de acoplamiento. Cuando los módulos del sistema se encuentran altamente acoplados provocan que un software sea difícil de mantener y especialmente difícil de modificar.

Los altos niveles de acoplamiento ocurren principalmente por dos razones:

- La evolución de un software. Es algo inherente a un software conforme pasa el tiempo. Mantener un software normalmente implica realizar cambios que resultan en un incremento en el nivel de acoplamiento gradualmente.

- Un pobre diseño de arquitectura inicial. Es necesario definir una arquitectura a seguir durante el desarrollo y mantenimiento del software para evitar problemas de acoplamiento debido al poco orden de su estructura.

Operaciones de refactorización y reestructuración de módulos son aplicadas generalmente para resolver este problema y remover la erosión que se genera debido a la evolución de un software (Candela, Bavota, Russo y Oliveto, 2016).

7.6.1. Técnicas de medición de acoplamiento

Conocer el nivel de acoplamiento de una aplicación ayuda a identificar la dificultad que tendrán tareas de modificación como agregar una nueva funcionalidad o arreglar un problema. Para analizar el nivel de acoplamiento de una aplicación hay que conocer sus componentes y las interacciones que existen entre ellos.

Los niveles de acoplamiento se pueden resumir en tres principales de acuerdo con las características mostradas en la figura 7.

Figura 7. **Niveles de acoplamiento**

Category	Coupling level	Characteristics
High levels	Content	A component uses data or control maintained by another component.
	Common	Components share global data items.
	External	Components are tied to external entities such as devices or external data.
Moderate levels	Control	Controls flow across components.
Low levels	Data structured	Structured data are transferred among components.
	Data	Primitive data or arrays of primitive data are passed among components.
	Message	Components communicate through standardized interfaces.

Fuente: Lou (2016). *A Comparison of Android Native App Architecture–MVC, MVP and MVVM.*

Con base en esta separación de categorías por nivel de acoplamiento, el análisis que debe realizarse en una comparación de arquitecturas debe incluir:

- El componente fuente.
- El componente destino.
- La conexión que existe.
- El nivel de acoplamiento.

Estos campos deben llenarse realizando un análisis de acuerdo con las características que se identifiquen en los flujos de acuerdo con cada arquitectura y los componentes definidos.

7.7. Servicios web

Un servicio web es un sistema de software que se utiliza para permitir la interacción de dos o más sistemas mediante una comunicación que utiliza un formato definido junto a un contrato descrito. Pueden ser desarrollados en

diferentes lenguajes y desplegados en diferentes tecnologías, pero se caracterizan por garantizar la interoperabilidad y porque adoptan estándares para facilitar el intercambio de datos (Morales, 2010).

Estos servicios pueden ser desarrollados en diferentes tipos de lenguajes y sobre diferentes tipos de redes computacionales. Por esta razón se hace necesario adoptar protocolos y estándares abiertos para lograr la interoperabilidad entre los sistemas. Entre los protocolos más conocidos están HTTP, FTP y SMTP. Algunos estándares que han sido muy populares por sus diferentes características son REST, XML, JSON, WSDL, entre otros.

7.7.1. *Firebase*

Desarrollar diferentes aplicaciones a la vez puede ser complicado para una sola persona. Al desarrollar una aplicación móvil se suele tener la necesidad de conectarse a un servicio *web* para almacenar información y consultar datos que necesitan de una lógica de negocio fuera de la misma aplicación. Manejar diferentes proyectos a la vez para tareas complejas como el manejo de base de datos, autenticación y autorización, mensajería y otras características de las aplicaciones incrementa significativamente el tiempo y el esfuerzo necesario para construir una aplicación (Maroney, 2017).

Debido a estas necesidades, Google lanzó el servicio llamado *firebase* en el año 2016, buscando proveer las herramientas e infraestructura necesaria para construir aplicaciones exitosamente de una manera ágil para el desarrollador. Provee servicios que comúnmente son necesarios como autenticación, notificaciones, *back-end*, entre otros.

La ventaja principal de utilizar un servicio de este tipo es que el desarrollador se puede enfocar en la aplicación móvil delegando la responsabilidad de las tecnologías de los servicios a *firebase*. Esta herramienta es compatible para construir tanto aplicaciones Android como iOS.

8. PROPUESTA DE ÍNDICE DE CONTENIDOS

9. METODOLOGÍA

9.1. Tipo de estudio

Para obtener una medida comparativa entre las versiones de la aplicación móvil desarrollada se realizará un tipo de estudio cualitativo. Utilizar una arquitectura correcta brinda muchas ventajas a los desarrolladores y reduce riesgos asociados a su implementación. Analizar un software y sus características suele ser complejo debido a que no existe una forma única de desarrollarlo, por esta razón el análisis se desarrollará sobre las calidades del software.

Es necesario obtener datos precisos que demuestren las diferencias principales al utilizar la arquitectura propuesta. Los datos obtenidos deben ser estructurados para facilitar su análisis posteriormente. Esto ayudará a tener datos que se relacionen con los objetivos planteados en la investigación.

9.2. Diseño

El estudio se desarrollará de manera experimental para tener un papel activo en el desarrollo y tener un mayor control sobre el factor principal del estudio, el cual es la arquitectura de las aplicaciones. Esto implica diseñar y desarrollar cada aplicación personalmente y luego realizar mediciones sobre el nivel de acoplamiento en las fases del estudio.

Las mediciones del nivel de acoplamiento se realizarán en las capas definidas por cada arquitectura al finalizar el desarrollo de cada aplicación

siguiendo el método definido por Lou, 2016. Esto brindará la habilidad de obtener una medición de sus características y poder desarrollar un análisis posteriormente.

9.3. Alcance

Para desarrollar la investigación se utilizará dos tipos de estudio. De manera principal se desarrollará como un estudio correlacional haciendo una comprobación entre el nivel de acoplamiento de las aplicaciones y mostrando que se puede desarrollar la misma aplicación separada en distintos módulos representados por cada capa de la arquitectura. De esta manera se aportará a demostrar la importancia en utilizar una arquitectura adecuada al desarrollar una aplicación móvil Android.

De manera secundaria, se incorporará parte del estudio siguiendo el tipo de investigación descriptiva para especificar las características de la aplicación algunas fases del estudio. Para identificar que existe una mejora en el software al utilizar la arquitectura planteada, es necesario medir los atributos de sus componentes, esto hará posible realizar posteriormente un análisis comparativo que ayude a demostrarlo.

9.4. Variables e indicadores

A continuación, se describen las variables e indicadores que se relacionan con la investigación.

Tabla I. **Variables e indicadores**

VARIABLES	DEFINICIÓN	SUB-VARIABLES	INDICADORES
Nivel de acoplamiento.	Variable independiente. Es la interdependencia que existe entre módulos de un software. Esta medida sirve para comprender con qué fuerza están relacionados los módulos que han sido implementados.	Características de las interacciones entre módulos.	1. Cantidad de interacciones entre módulos. 2. Categoría de nivel de acoplamiento en la escala descrita por el método de Lou, 2016.
Módulos independientes desarrollados.	Variable dependiente. Cantidad de módulos desarrollados de acuerdo con las capas de la arquitectura que son integrados en un proyecto.		1. Cantidad de módulos definidos en un proyecto para su reutilización mediante los scripts de compilación Gradle.

Fuente: elaboración propia.

9.5. Técnicas de recolección de información

Como parte del estudio se realizará un análisis documental de fuentes previas que aportan conceptos e información relevante para el estudio. Dentro

del trabajo se incluye la bibliografía de estas fuentes y se hace referencia a ellas con el fin de hacer uso de este conocimiento y soportar la presente investigación.

Para recolectar la información sobre los niveles de acoplamiento en las aplicaciones Android desarrolladas en las fases del estudio se utilizará el modelo de Lou, 2016.

Para contar con un control adecuado y organizar la información recolectada se utilizará un formato de tabla de registro que se rellenará al analizar las interacciones de los módulos en cada aplicación. El formato de la tabla de registro se encuentra descrito en la tabla II.

Tabla II. **Registro de nivel de acoplamiento**

Aplicación:			
Arquitectura utilizada:			
Cantidad de módulos desarrollados:			
MÓDULO FUENTE	MÓDULO DESTINO	DESCRIPCIÓN	NIVEL DE ACOPLAMIENTO

Fuente: elaboración propia.

Es necesario rellenar esta tabla de registro cada vez que se finalice el desarrollo y análisis de las diferentes aplicaciones desarrolladas como parte de las fases del estudio. Documentar esta información dentro del estudio será la base para el posterior análisis de la información y la elaboración de las conclusiones y recomendaciones.

9.6. Fases del estudio

El estudio se desarrollará en fases que siguen una estructura que introduce al tema conceptualmente para luego enlazarse con el trabajo realizado y sus resultados.

9.6.1. Revisión documental

Investigación y consulta de tesis previas, artículos, revistas y libros de carácter científico sobre los conceptos relacionados que sustentan la investigación.

Actividades:

- Investigación y definición de conceptos relacionados. Consulta de bibliografía y recolección de material teórico útil para la investigación.
- Descripción técnica de herramientas y tecnologías incluidas como parte de la investigación.
- Descripción de técnicas de medición en niveles de acoplamiento de software. Soporte previo de investigación científica para la medición de niveles de acoplamiento de software orientado a aplicaciones móviles Android.

Entre los temas más importantes de la investigación se encuentra el acoplamiento de código en una aplicación. Para poder analizar y medir esto dentro del software es necesario conocer las técnicas que existen e investigar sobre estudios previos y sus resultados.

Las aplicaciones móviles han evolucionado ampliamente, principalmente en la última década, gracias a los avances tecnológicos que han surgido alrededor del mundo. Android ha sido de los principales sistemas operativos en el mercado de teléfonos inteligentes y por esto es necesario profundizar y dar a conocer los conceptos relacionados a esta tecnología incluyendo temas de tecnologías utilizadas y tendencias para desarrollar aplicaciones en este sistema.

Se describirán las arquitecturas a utilizar dentro de la investigación, así como las fuentes sobre las cuales se basó la arquitectura planteada como solución dentro del presente trabajo.

9.6.2. Diseño de aplicaciones

Es necesario presentar y definir las funcionalidades que las aplicaciones contendrán, así como también las tecnologías que se utilizarán dentro de cada aplicación. Estas aplicaciones son:

- Aplicación de catálogo de comercios.
- Aplicación de catálogo de museos.

Dentro del diseño de cada aplicación deben describirse las pantallas que contendrá incluyendo la siguiente información:

Tabla III. **Diseño de funcionalidades**

Nombre: Describir funcionalidad principal.
Interfaz de Usuario: Diseño de la interfaz de usuario.
Casos de Uso: Descripción de las interacciones del usuario.
Fuente de la información: Describir las fuentes que obtendrán la información de la pantalla.
Tecnologías: Describir las tecnologías necesarias para desarrollar cada aplicación.

Fuente: elaboración propia.

9.6.3. Implementación MVC en aplicación de catálogo de comercios

Se implementará la arquitectura MVC en la aplicación de catálogo de comercios. Esta aplicación necesita ser ajustada para que demuestre una correcta separación entre las capas que define esta arquitectura y pueda ser analizada. Esta fase es importante debido a que esta aplicación es la base del estudio y será utilizada en las fases posteriores.

La aplicación debe mostrar claramente una separación en las capas definidas por esta arquitectura:

- Modelo
- Vista
- Controlador

9.6.4. Aplicación de arquitectura propuesta en aplicación de catálogo de comercios

Dentro de esta fase se realizará la implementación en la aplicación de catálogo de comercios de la arquitectura propuesta. El resultado de aplicar esta arquitectura será el desarrollo de los siguientes módulos independientes:

- Presentación
- Casos de uso
- Negocio
- Repositorio
- Interfaces externas

9.6.5. Análisis de acoplamiento en aplicación de catálogo de comercios

El resultado de las dos fases anteriores será contar con dos versiones de la misma aplicación de catálogo de comercios, con la diferencia que cada una habrá sido desarrollada utilizando diferente arquitectura.

En esta fase se realizará un análisis del nivel de acoplamiento en los módulos desarrollados de estas aplicaciones. Para este análisis se utilizará una tabla descriptiva analizando cada interacción existente entre las capas definidas por cada arquitectura. Esta tabla se encuentra descrita anteriormente en las técnicas de recolección de información siguiendo el modelo presentado por Lou, 2016.

9.6.6. Desarrollo de aplicación de catálogo de museos

Se desarrollará una nueva aplicación de catálogo de museos reutilizando la capa de vista desarrollada en la aplicación de catálogo de comercios al utilizar la arquitectura propuesta. Para el desarrollo de esta aplicación se continuará utilizando esta arquitectura, pero se realizarán cambios en la fuente de datos, que serán obtenidos de manera remota utilizando el servicio *firebase*, tal como se muestra en la fase de diseño de aplicaciones.

Para reutilizar el módulo de la vista se utilizarán scripts de compilación Gradle y únicamente se realizarán cambios en los recursos definidos para modificar textos sin realizar cambio alguno en el código de la lógica de la vista.

El objetivo principal de esta fase es aportar al objetivo general demostrando la eficiencia de la arquitectura al permitir la reutilización del código en la capa de presentación sin la necesidad de realizar modificaciones en esta parte.

10. TÉCNICAS DE ANÁLISIS DE LA INFORMACIÓN

Para desarrollar el análisis de la información recopilada dentro de la fase experimental y descriptiva del proyecto se utilizarán un análisis numérico de los datos.

10.1. Correlación

Calcular la correlación que existe entre las variables descritas. Este dato es importante para dar a conocer el impacto que genera una reducción o incremento en el nivel de acoplamiento del código y la capacidad de reutilizar módulos en un proyecto.

Para calcular este valor se utilizará la fórmula para calcular el coeficiente de correlación de Pearson para datos no agrupados, el cual puede encontrarse mediante la siguiente fórmula.

Figura 8. **Fórmula coeficiente de correlación de Pearson**

$$coeficiente\ de\ correlación = \frac{\sum xy}{\sqrt{(\sum x^2)(\sum y^2)}}$$

$$x = X - \bar{X}$$

$$y = Y - \bar{Y}$$

Fuente: elaboración propia.

10.2. Tabulación cruzada

Desarrollar una tabulación cruzada para realizar un análisis descriptivo sobre las interacciones que ocurren entre las capas de las arquitecturas. En las tablas 4 y 5 se describe la estructura que debe contener esta información.

Tabla IV. **Tabulación cruzada, arquitectura MVC**

	Modelo	Vista	Controlador
Modelo			
Vista			
Controlador			

Fuente: elaboración propia.

Tabla V. **Tabulación cruzada, arquitectura propuesta**

	Presentación	Casos de uso	Lógica de negocio	Repositorio	Interfaces
Presentación					
Casos de uso					
Lógica de negocio					
Repositorio					
Interfaces					

Fuente: elaboración propia.

10.3. Tabulación de datos

Realizar una tabulación de datos para estructurarlos y analizarlos de manera estadística mediante medidas de tendencia central, de dispersión y realizar un análisis estadístico.

Tabla VI. **Tabulación de datos, interacciones y nivel de acoplamiento**

Capa	Cantidad de interacciones	Nivel de acoplamiento

Fuente: elaboración propia.

Tabla VII. **Tabulación de datos, acoplamiento y módulos independientes**

Arquitectura	Nivel de acoplamiento	Módulos independientes

Fuente: elaboración propia.

10.4. Resumen estadístico

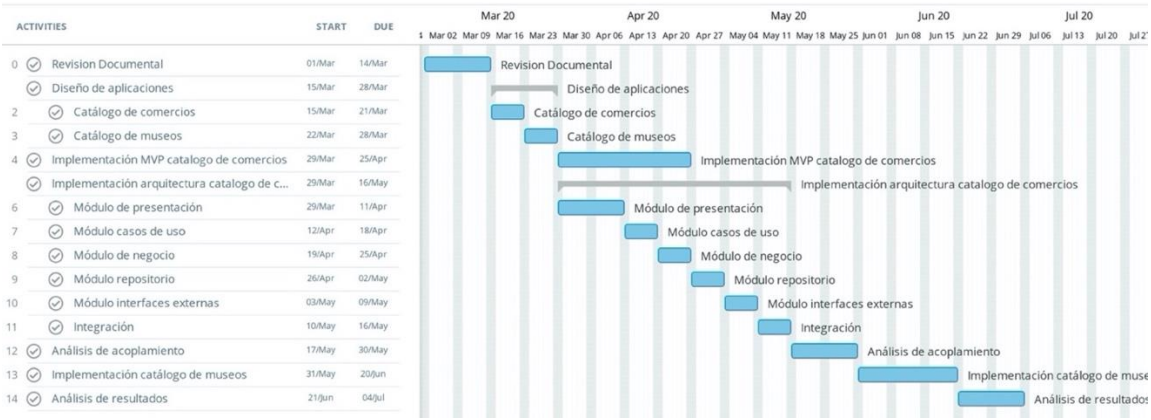
Se elaborará un resumen con información estadística relevante sobre los datos obtenidos. Dentro de esta información se incluirán los siguientes elementos:

- Medidas de tendencia central.
  - Media
  - Mediana
  - Moda

- Medidas de dispersión.
  - Desviación estándar
  - Varianza
  - Rango

11. CRONOGRAMA

Figura 9. **Cronograma**



Fuente: elaboración propia.

12. FACTIBILIDAD DEL ESTUDIO

12.1. Factibilidad operativa

Se describe y analiza la capacidad que tiene el estudio para ser desarrollado de manera satisfactoria.

12.1.1. Recursos humanos

Para el desarrollo de la investigación, desarrollo del proyecto y análisis de resultados es necesario contar con los siguientes roles principales:

- Arquitecto: encargado de diseñar la arquitectura utilizada en cada aplicación y asegurar que la implementación sea correcta.
- Diseñador: la función principal es definir el diseño de la interfaz de usuario y describir la experiencia de usuario.
- Desarrollador: a cargo de implementar el código de las aplicaciones, siguiendo los lineamientos de la arquitectura definida y utilizando las tecnologías descritas en esta investigación. También se encuentra a cargo de corregir cualquier error o mal funcionamiento que se encuentre en la aplicación al ser desarrollada.
- Aseguramiento de calidad: realizar pruebas para asegurar que la aplicación está cumpliendo con el funcionamiento esperado.
- Analista: desarrollar el análisis en el nivel de acoplamiento de las aplicaciones.

El rol de desarrollador para la aplicación siguiendo un patrón MVP será cubierto por un desarrollador externo subcontratado con el fin de obtener una base justa para su posterior comparación en la investigación.

Todos los demás roles descritos serán desarrollados por el investigador del presente trabajo.

Como parte de la investigación es necesario realizar revisiones al trabajo y para ello es necesario contar con la disponibilidad del siguiente recurso humano:

- Revisores de contenido de Escuela de Postgrado de Ingeniería: brindar retroalimentación durante el desarrollo de la investigación para asegurar que el trabajo cumple con los estándares requeridos por la Escuela de Postgrado de Ingeniería en la Universidad de San Carlos de Guatemala.
- Asesor del trabajo: persona que ayudará a guiar el trabajo de investigación haciendo uso de la experiencia previa, con la que debe contar en el ámbito de ingeniería y trabajos a nivel de maestría.

12.1.2. Acceso a la información

Para realizar la revisión documental es necesario contar con acceso a la información que permita soportar los conceptos relacionados a la investigación y trabajos previos que se relacionen con el tema, para ello son necesarios los siguientes recursos:

- Acceso a internet.
- Acceso a sitios de investigación.

12.1.3. Permisos

La información que brindará las aplicaciones será información de carácter pública por lo cual no es necesario contar con permisos especiales para incluirla. Esta información únicamente debe ser corroborada para poder brindar información correcta al usuario.

Dentro de la información principal necesaria a recolectar para ser incluida dentro de las aplicaciones se encuentra el siguiente listado de elementos:

- Nombre del lugar.
- Ubicación del lugar.
- Horario de atención al público.
- Costo para ingresar.
- Información de contacto.

12.1.4. Equipo

Para desarrollar aplicaciones móviles Android se puede hacer uso del emulador que provee el SDK nativo. Con esto se elimina la necesidad de contar con un dispositivo físico durante el desarrollo.

Para el análisis de la implementación y presentación de resultados se recomienda utilizar con un dispositivo físico debido a que esto dará un mayor grado de certeza a que la aplicación ha sido desarrollada correctamente y a comprobar que es funcional. Para esto se cuenta con un dispositivo Motorola G6 plus de uso personal que se pone a disposición de la investigación.

12.1.5. Conclusión

No se han encontrado limitantes que detengan el desarrollo de la investigación y se cuenta con los recursos operativos necesarios para elaborarla, por lo que se puede concluir que la operación está garantizada.

12.2. Factibilidad técnica

Para desarrollar exitosamente el proyecto se debe considerar si los recursos técnicos actuales son suficientes o deben complementarse.

12.2.1. Disponibilidad tecnológica

La necesidad de acceso a las tecnologías es una parte fundamental del proyecto por la naturaleza de este. Es importante tener la libertad de implementar las aplicaciones sin restricciones para tener control sobre las pruebas desarrolladas dentro de la investigación.

Las librerías y marcos de trabajo principales que se utilizarán son los siguientes:

- SDK de Android
- Java
- *Firebase*
- Github
- Dagger

Todas estas tecnologías brindan un acceso libre y sin incurrir en ningún costo. Esto brinda las características perfectas para desarrollar la investigación y tener la disponibilidad tecnológica necesaria.

12.2.2. Conclusión

Gracias al tiempo disponible que se estima para desarrollar la investigación y a la experiencia previa que se tiene desarrollando este tipo de aplicaciones, se concluye desde una perspectiva técnica que los recursos con los que se cuentan son suficientes para alcanzar exitosamente los objetivos de la investigación.

12.3. Factibilidad económica

Como se ha detallado en la factibilidad operativa y técnica, no se han encontrado carencias en los recursos que se tengan que solventar mediante inversiones económicas.

A continuación, en la tabla VIII se detallan los costos naturales necesarios para desarrollar el proyecto.

Tabla VIII. **Descripción de costos**

Descripción	Costo (Q.)
Tiempo de recurso humano: 1. Arquitecto 2. Diseñador 3. Desarrollador (externo) 4. Desarrollador (investigador) 5. Aseguramiento de calidad 6. Analista **Sub total:**	 Q. 7,000.00 Q. 2,000.00 Q. 4,000.00 Q. 10,000.00 Q. 3,000.00 **Q. 26,000.00**
Asesor de Tesis	Q. 2,500.00
Revisores de contenido de escuela de postgrados de ingeniería	Q. 0.00
Motorola G6 Plus (costo aproximado en Guatemala)	Q. 1,800.00
Servicio de almacenamiento *Firebase*	Q. 0.00
SDK y herramientas Android	Q. 0.00
TOTAL:	**Q. 30,300.00**

Fuente: elaboración propia.

12.3.1. Conclusión

Al analizar los datos económicos necesarios que han sido presentados se concluye que el proyecto es económicamente factible ya que el investigador posee la capacidad de absorber dichos gastos para poder desarrollar el proyecto.

12.4. Conclusión de factibilidad

Se ha llegado a la conclusión que el proyecto es completamente factible luego de realizar el análisis desde los puntos de vista de factibilidad operativa, técnica y económica. No se han encontrado factores determinantes que bloqueen el desarrollo del proyecto ni inversiones fuertes que deban realizarse para suplir alguna carencia. Dentro de los recursos económicos necesarios para desarrollar el proyecto, el investigador cuenta con la capacidad necesaria para cubrirlos.

13. REFERENCIAS BIBLIOGRÁFICAS

1. Abrahamsson P., Hanhineva A., Jäälinoja J. (2005). *Improving Business Agility Through Technical Solutions: A Case Study on Test-Driven Development in Mobile Software Development.* Baskerville R.L., Mathiassen L., Pries-Heje J., DeGross J.I. (eds) Business Agility and Information Technology Diffusion. Conferencia IFIP International Federation for Information Processing, vol 180. Springer, Boston, MA.

2. Alghamdi, J. (2008). Measuring software coupling. *The Arabian Journal for Science and Engineering.* Volumen (33), pp. 119-129.

3. Android Jetpack. (2019). *Guide to app architecture.* San Francisco, California. Recuperado de: https://developer.android.com/jetpack/docs/guide

4. AndroidX Releases. (2019). *Architecture Components Release Notes Archive.* San Francisco, California. Recuperado de: https://developer.android.com/jetpack/androidx/releases/archive/arch

5. Arisholm, E., Briand L.C., Foyen A. (2004) Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering.* Volumen (30), pp. 491-506.

6. Calero, C., Moraga, M., y Piattini, M. (2010). *Calidad del producto y proceso del software.* España: Editorial RA-MA.

7. Cheng, Y. & Olivares, A. (2018). *Advance Android App Architecture.* Estados Unidos: Editorial RayWanderlich.

8. Clement, J. (2019). *Mobile app usage - Statistics & Facts.* Alemania: Statista. Recuperado de: https://www.statista.com/topics/1002/mobile-app-usage/

9. Kim, H., Choi, B., y Yoon, S. (2009). Performance testing based on test-driven development for mobile applications. *III Conferencia Internacional Ubiquitous Information Management and Communication.* Suwon, Korea.

10. Perry, D., & Wolf, A. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*. Volumen (17) pp. 40-52.

11. Diaz, G. (2015). *Carpooling gt, aplicación para compartir vehículos.* (Tesis de Maestría) Universidad de San Carlos de Guatemala. Guatemala.

12. Google Inc. (2019). *Android*. Recuperado de http://www.android.com/.

13. Báez, M., Borrego, A. y Cordero, J. (2013). *Introducción a Android.* Madrid, España: Editorial E.M.E.

14. Zandbergen, P. (2011) *Positional Accuracy of Assisted GPS Data from High-Sensitivity GPS-enabled Mobile Phones*. Cambridge University Press, 64(3).

15. Martin, R. (2017) *Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition*. Estados Unidos: Prentice Hall.

16. Potel, M. (1996). *MVP: Model-View-Presenter the Taligent programming model for C++ and Java*. Taligent Inc, 20.

17. Lou, T. (2016). *A Comparison of Android Native App Architecture–MVC, MVP and MVVM*. (Tesis de Maestría). Aalto University. Finlandia.

18. Candela, I., Bavota, G., Russo, B., y Oliveto, R. (2016). *Using Cohesion and Coupling for Software Remodularization: Is It Enough?* ACM Transactions on Software Engineering and Methodology (TOSEM), 25(3).

19. Morales, C. (2010). *Estado del Arte: Servicios Web.* (Tesis de Maestría). Universidad Nacional de Colombia. Colombia.

20. Moroney, L. (2017). *The Definitive Guide to Firebase.* Berkeley, CA. Seattle, USA: Apress.