Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Estudios de Postgrado
Maestría en Artes en Tecnologías de la Información de la Comunicación

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

Ing. Francisco Rogelio Anzueto Marroquín

Asesorado por el M.A. Ing. Héctor Alberto Heber Mendía Arriola

Guatemala, noviembre de 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

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

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

ING. FRANCISCO ROGELIO ANZUETO MARROQUÍN
ASESORADO POR EL M.A. ING. HECTOR ALBERTO HEBER MENDIA ARRIOLA

AL CONFERÍRSELE EL TÍTULO DE

MAESTRO EN ARTES EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIÓN

GUATEMALA, NOVIEMBRE DE 2021

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. Kevin Armando Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Mtro. Ing. Edgar Darío Álvarez Cotí
EXAMINADOR	Mtro. Ing. Marlon Antonio Pérez Türk
EXAMINADOR	Mtro. Ing. Estuardo Enrique Echeverría Nova
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:

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.

Ing. Francisco Rogelio Anzueto Marroquín

USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

Decanato
Facultad de Ingeniería
24189101 - 24189102

DTG. 735.2021

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 Estudios de Postgrado, al Trabajo de Graduación titulado: **DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO,** presentado por el **Ingeniero Francisco Rogelio Anzueto Marroquín,** estudiante de la **Maestría en Tecnologías de la Información y Comunicación** 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

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, noviembre de 2021.

AACE/cc

Guatemala, noviembre de 2021

LNG.EEP.OI.147.2021

En mi calidad de Director de la Escuela de Estudios de Postgrado de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor, verificar la aprobación del Coordinador de Maestría y la aprobación del Área de Lingüística al trabajo de graduación titulado:

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

presentado por **Francisco Rogelio Anzueto Marroquín** quien se identifica con carné **200611559** correspondiente al programa de **Maestría en artes en Tecnologías de la información y la comunicación** ; apruebo y autorizo el mismo.

Atentamente,

"Id y Enseñad a Todos"

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

Guatemala, 10 de julio 2021

M.A. Edgar Darío Álvarez Cotí
Director
Escuela de Estudios de Postgrado
Presente

M.A. Ingeniero Álvarez Cotí:

Por este medio informo que he revisado y aprobado el **TRABAJO DE GRADUACIÓN** titulado: "DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO" del estudiante **Francisco Rogelio Anzueto Marroquín** quien se identifica con número de carné **200611559** del programa de **Maestría en Tecnologías de la Información y la Comunicación.**

Con base en la evaluación realizada hago constar que he evaluado la calidad, validez, pertinencia y coherencia de los resultados obtenidos en el trabajo presentado y según lo establecido en el *Normativo de Tesis y Trabajos de Graduación aprobado por Junta Directiva de la Facultad de Ingeniería Punto Sexto inciso 6.10 del Acta 04-2014 de sesión celebrada el 04 de febrero de 2014.* Por lo cual el trabajo evaluado cuenta con mi aprobación.

Agradeciendo su atención y deseándole éxitos en sus actividades profesionales me suscribo.

Atentamente,

MARLON ANTONIO PEREZ TURK
INGENIERO EN CIENCIAS Y SISTEMAS
COLEGIADO No. 4492

MA. Ing. Marlon Antonio Pérez Türk
Coordinador
Maestría en Tecnologías de la Información y la Comunicación
Escuela de Estudios de Postgrado

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Estudios de Postgrado
Coordinador de Área

Guatemala, 22 de septiembre de 2021.

M.A. Ing. Edgar Darío Álvarez Cotí
Director
Escuela de Estudios de Postgrados
Presente

Estimado M.A. Ing. Álvarez Cotí:

Reciba un cordial y atento saludo, a la vez aprovecho la oportunidad para hacerle de su conocimiento que he revisado y aprobado el trabajo de graduación titulado: "DESARROLLO DE ARQUITECTURA MÓVIL PARA ANDROID QUE ASEGURA LA REDUCCIÓN DE ACOPLAMIENTO EN EL CÓDIGO" del estudiante FRANCISCO ROGELIO ANZUETO MARROQUÍN del Programa de Maestría en Tecnología de la Información y Comunicación, identificado con número de carné: *200611559*.

Agradeciendo su atención y deseándole éxitos en sus actividades profesionales me suscribo.

Atentamente,

"Id y enseñad a todos"

Ing. Hector Alberto H. Mendia Arriola
Ciencias y Sistemas
Col 10,057

M.A. Ing. Héctor Alberto Heber Mendía Arriola
Colegiado No. 10,057
Asesor

Cc: Archivo/LA

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, amor, esfuerzo, carácter y trabajo; pero, 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.
Andrea Castillo	Por inspirarme y motivarme siempre a seguir adelante. Por acompañarme en cada paso que he dado y darme ánimos siempre que lo necesito.

Mi familia

Porque todos son el mayor motivo de alegría que tengo en esta vida. Todos mis logros son para aquellos que están hoy aquí y para los que nos esperan allá en el cielo.

Mis compañeros

Por todas las experiencias vividas dentro y fuera de la universidad. Agradezco todo el apoyo recibido y la amistad que me han brindado estos años.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser el 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	M.A. Ing. Héctor Alberto Heber Mendía Arriola, por haberme guiado durante el trabajo de graduación.
Docentes	Agradezco todos los conocimientos y experiencias compartidas.
Xoom, a PayPal Service	Por todo el apoyo recibido que me permitió tomar este programa de maestría.
Familia y amigos en general	

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
$	Dólar estadounidense
=	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.
Gradle	Sistema de automatización de construcción de código de software que construye sobre los conceptos de *Apache Ant* y *Apache Maven* e introduce un lenguaje específico del dominio basado en Groovy.
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	Kit de desarrollo nativo (*Native Development Kit)* es un kit de desarrollo de software basado en una interfaz de aplicaciones nativa que permite desarrollar software directamente en una plataforma en lugar de utilizar una máquina virtual.
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 de 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 escrita en lenguaje C.
Teléfonos Inteligentes	Es un tipo de ordenador o computadora de bolsillo con las capacidades de un teléfono móvil o celular.

VIPER	*View, Interactor, Presenter, Entity y Router*. Arquitectura basada en principios de responsabilidad única.
VVMIR	Vista, Modelo de vista, *Interactor* y Repositorio.
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.

Primordialmente, 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.

En la presente investigación se desarrolla la arquitectura VVMIR que permite reducir el nivel de acoplamiento de código en una aplicación de catálogo

de comercios y la reutilización de sus componentes una segunda aplicación de catálogo de museos.

Para verificar la eficiencia de la arquitectura en la implementación de una aplicación se ha desarrollado la misma aplicación varias veces utilizando diferentes arquitecturas y se ha medido el nivel de acoplamiento en cada una de estas.

La arquitectura VVMIR resulta efectiva reduciendo el nivel de acoplamiento de la aplicación analizada y facilita la reutilización del código de la aplicación de catálogo de comercios al desarrollar la aplicación de catálogo de museos.

Utilizar herramientas como Firebase y Gradle durante el desarrollo de los experimentos ha sido beneficioso para alcanzar los objetivos de la investigación.

La arquitectura VVMIR resuelve el problema de acoplamiento de código. Se recomienda a desarrolladores, analistas, diseñadores y arquitectos de software contar con conocimientos de diversas arquitecturas de software para tomar la decisión correcta sobre la arquitectura a utilizar en sus proyectos de acuerdo con sus necesidades.

PLANTEAMIENTO DEL PROBLEMA Y FORMULACIÓN DE PREGUNTAS ORIENTADORAS

Desarrollar una aplicación Android es algo a lo que muchas empresas se dedican o se sienten atraídas hoy en día por el alto índice de uso que tienen estos dispositivos. Estas aplicaciones pueden ser desarrolladas por equipos pequeños y en ocasiones con poca experiencia. Esto causa que algunos proyectos de desarrollo no sean completados satisfactoriamente o 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 puedan apegarse 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 posterior que necesite realizarse, provocando que existan atrasos en las entregas. Además, provoca 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 utilizan para mejorar la calidad en el código y en el producto final. 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 ayudará 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 formal como sugerencia para los desarrolladores Android.

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. 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.

Se cuenta con una aplicación Android previamente desarrollada que funciona como un catálogo de comercios en Guatemala. Fue desarrollada utilizando un patrón Modelo-Vista-Controlador (MVC) y se espera reutilizar su código 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?

- ¿Se puede utilizar Gradle como herramienta para reutilizar módulos en diferentes aplicaciones móviles Android?

OBJETIVOS

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.

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 Gradle como herramienta que permita la reutilización de módulos de la aplicación de catálogo de comercios en la aplicación de catálogo de museos.

MARCO METODOLÓGICO

- Tipo de estudio

Para obtener una medida comparativa entre las versiones de la aplicación móvil desarrollada se realizó un estudio de tipo cualitativo. 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 desarrolló sobre las cualidades del software. Utilizar una arquitectura correcta brinda muchas ventajas a los desarrolladores y reduce riesgos asociados a su implementación.

Los datos obtenidos han sido estructurados para facilitar su análisis. Esto ayuda a tener datos que se relacionen con los objetivos planteados en la investigación. Estos datos muestran las diferencias principales al utilizar la arquitectura propuesta.

- Diseño

Se utilizó un diseño 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 implicó diseñar y desarrollar cada aplicación personalmente; luego, realizar mediciones sobre el nivel de acoplamiento en las fases del estudio.

Las mediciones del nivel de acoplamiento se realizaron en las capas definidas por cada arquitectura al finalizar el desarrollo de cada aplicación siguiendo el método definido por Lou (2016) en su tesis de maestría. Esto brinda la habilidad de medir las características de cada aplicación y permite desarrollar un análisis posteriormente.

- Alcance

Para desarrollar la investigación se utilizaron dos tipos de estudio. De manera principal se desarrolló 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 demuestra la importancia en utilizar una arquitectura adecuada al desarrollar una aplicación móvil Android.

De manera secundaria, se desarrolló parte del estudio siguiendo el tipo de investigación descriptiva para especificar las características de la aplicación y 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 para realizar posteriormente un análisis comparativo y soportar la demostración.

- Variables e indicadores

En la tabla I, se describen las variables e indicadores de 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. 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.	Esfuerzo realizado	1. Cantidad de módulos definidos mediante los scripts de compilación Gradle. 2. Líneas de código. 3. Cantidad total de archivos de código.

Fuente: elaboración propia.

- Técnicas de recolección de información

Como parte del estudio se realizó 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 utilizó el modelo de la investigación de Lou (2016), presentado dentro del marco teórico.

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

Tabla II. **Registro nivel de acoplamiento**

Aplicación:		
Arquitectura utilizada:		
Cantidad de módulos desarrollados:		
Capas distinguibles desarrolladas:		
CAPA	**DESCRIPCIÓN**	**NIVEL DE ACOPLAMIENTO**

Fuente: elaboración propia.

Documentar esta información dentro del estudio ha sido la base para hacer posible el análisis de la información y desarrollar las conclusiones y recomendaciones.

- Fases del estudio

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

- Fase I. 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 describen las arquitecturas utilizadas dentro de la investigación, así como las fuentes sobre las cuales se basó la arquitectura planteada como solución dentro del presente trabajo.

- Fase II. Diseño de aplicaciones

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

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

Dentro del diseño de cada aplicación debe ejemplificarse el diseño de cada pantalla individual, así como también describirse los detalles con la información que contendrán incluyendo la información que se describe en la Tabla III.

Tabla III. **Diseño de funcionalidades**

Nombre: Nombre que describa la funcionalidad principal.
Descripción: Descripción general 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.

- Fase III. Implementación MVC en aplicación de catálogo de comercios

Se implementó la arquitectura MVC en la aplicación de catálogo de comercios. Esta aplicación necesitó ser ajustada para tener una mejor visualización de las capas que define esta arquitectura y poder ser analizada de una mejor manera. También fue necesario actualizarla al lenguaje *Kotlin* debido a que este es el nuevo estándar que se utiliza para desarrollar aplicaciones Android. Esta fase es importante debido a que esta aplicación es la base del estudio y es utilizada en las fases posteriores.

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

- Modelo
- Vista
- Controlador

- Fase IV. Aplicación de arquitectura MVVM en aplicación de catálogo de comercios

Se actualizó la aplicación de catálogo de comercios siguiendo la arquitectura MVVM sugerida en la documentación de Google para desarrolladores. El resultado de aplicar esta arquitectura muestra una separación en las siguientes capas:

- Vista
- Vista de modelo (*ViewModel*)
- Repositorio

- Fase V. Aplicación de arquitectura VVMIR en aplicación de catálogo de comercios

Dentro de esta fase se actualizó la aplicación de catálogo de comercios utilizando la arquitectura VVMIR. El resultado de aplicar esta arquitectura fue el desarrollo de los siguientes módulos independientes:

- Vista
- Vista de modelo (*ViewModel*)
- Negocio (*Interactor*)
- Interfaces de negocio (*Interactor*)
- Repositorio
- Interfaces de repositorio

- Fase VI. Análisis de acoplamiento en aplicación de catálogo de comercios

Como resultado de las fases anteriores se cuenta con tres versiones de la misma aplicación de catálogo de comercios, con la diferencia que cada una ha sido desarrollada utilizando diferente arquitectura.

En esta fase se realizó un análisis del nivel de acoplamiento en las capas y módulos desarrollados en estas aplicaciones. Para este análisis se utilizó una tabla descriptiva mostrando cada interacción existente entre las capas definidas por cada arquitectura. Esta tabla se encuentra descrita en los resultados del estudio y sigue el modelo de medición presentado por Lou (2016).

- Fase VII. Desarrollo de aplicación de catálogo de museos

Se desarrolló una nueva aplicación de catálogo de museos realizando el cambio de la fuente de datos en las tres aplicaciones desarrolladas anteriormente para cada arquitectura. Estos datos son obtenidos de manera remota utilizando una instancia diferente del servicio Firebase, tal como se muestra en el diseño de las aplicaciones.

Se realizaron mediciones de reutilización de código a cada aplicación para verificar y comparar el nivel de reutilización de archivos y líneas de código que cada aplicación permite.

Al utilizar la arquitectura VVMIR, se reutilizó la capa de vista, modelo de vista y capa de negocios (Interactor), únicamente realizando cambios en la fuente de datos.

Para reutilizar los módulos se utilizaron *scripts* de compilación Gradle y únicamente se realizaron cambios en el repositorio y en los recursos definidos para modificar textos sin realizar cambio alguno en el código de la lógica de la vista, modelo de vista y capa de negocio.

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 y de negocio sin la necesidad de realizar modificaciones en ninguna de estas áreas.

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 de dos o más personas que se encuentran distribuidos en distintas partes del mundo. 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 desarrolla la presentación de resultados, definiendo y delimitando todas las funcionalidades que incluyen las aplicaciones, así como las tecnologías que se utilizan. Se muestran las versiones desarrolladas de la

aplicación de catálogo de comercios que sirven como punto de comparación y permiten medir las características del código.

También se muestran los resultados de implementar la aplicación de catálogo de museos de Guatemala reutilizando los módulos posibles de la aplicación de catálogo de comercios, demostrando que la arquitectura planteada lo permite. Este capítulo incluye 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, demostrando la efectividad obtenida.

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

1. 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.

Clement (2019) en su publicación titulada *Mobile app usage - Statistics & Facts* menciona que en el año 2018 se descargaron 205.4 billones de aplicaciones móviles alrededor del mundo. 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 como lo menciona Alghamdi (2008) en su publicación titulada *Arabian Journal for Science & Engineering*.

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 Android al ser Java uno de sus lenguajes principales soportados.

Abrahamsson, Hanhineva, y Jäälinoja (2005) mencionan en su publicación *Improving business agility through technical solutions: A case study on test-driven development in mobile software development,* que 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.

Debido a las calendarizaciones cortas en un proyecto se obvian aspectos importantes como la arquitectura de software y las buenas prácticas de desarrollo. Esto trae como consecuencia la tardía aparición de problemas en el software y que sea más difícil identificar, rastrear y modificar las fallas como lo menciona Kim, Choi y Yoon (2009) en su trabajo *Performance testing based on test-driven development for mobile applications*.

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. Arisholm, Briand, y Foyen (2004) en su trabajo *Dynamic coupling measurement for object-oriented software,* explican que 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.

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 como mencionan Calero, Moraga y Piattini (2010) en su trabajo *Calidad del producto y proceso software*.

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.

Cheng y Olivares (2018) explican en su publicación *Advance Android App Architecture,* que en Estados Unidos que 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.

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.

2. 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 pueden 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 pueden tener las aplicaciones móviles que implementan un patrón MVC tradicional han sido propuestas arquitecturas más robustas como MVVM, MVP, *Clean*, VIPER, entre otras, para solucionar este problema. Estas arquitecturas ayudan 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 sola.

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.

Es importante diseñar una arquitectura que facilite el mantenimiento de las aplicaciones porque se encuentran en un ecosistema cambiante en el que pueden necesitar ser actualizadas constantemente.

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 evidencian 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 al contar con un canal adicional para la difusión de información.

3. ALCANCES

3.1. Perspectiva investigativa

La perspectiva investigativa que se trabajará se explica a continuación.

- Investigación y recopilación de información de estudios previos sobre arquitecturas disponibles para desarrollar aplicaciones móviles Android.

- Investigación sobre técnicas para medir el nivel de acoplamiento en software que se utilizan para analizar el código de las aplicaciones Android desarrolladas en este trabajo.

- Investigación de temas relacionados a las herramientas que permiten la reutilización de módulos en aplicaciones móviles Android.

3.2. Perspectiva técnica

La arquitectura propuesta está basada en la arquitectura MVVM, arquitectura para controlar la capa de presentación y en las arquitecturas VIPER y *Clean* para desarrollar módulos independientes en capas ajenas a la capa de presentación. Se implementó en la aplicación utilizando el lenguaje de programación *Kotlin* con el SDK nativo de Android mediante la herramienta Android Studio.

Para realizar la medición en el nivel de acoplamiento de código en las aplicaciones se utilizó la escala definida en la investigación de Lou (2016), que

define cinco niveles diferentes de acuerdo con las características en las interacciones entre las capas de una arquitectura.

Se utilizó el sistema de compilación Gradle para configurar los proyectos y poder reutilizar los módulos desarrollados. Estos scripts han sido desarrollados utilizando el lenguaje Groovy.

3.3. Perspectiva de resultados

La perspectiva de resultados se presentará en cuatro factores los cuales se explican a continuación.

- El despliegue de la arquitectura propuesta en la aplicación de catálogo de comercios mostrando claramente una segmentación en los módulos desarrollados de acuerdo con las capas definidas en la arquitectura.

- Medición del nivel de acoplamiento de código en las aplicaciones al utilizar la escala definida en la investigación de Lou (2016).

- Análisis comparativo que permita validar la eficiencia de la arquitectura respecto al nivel de acoplamiento de código en las aplicaciones.

- Módulo de presentación reutilizado al desarrollar la aplicación de catálogo de museos de Guatemala

4. MARCO TEÓRICO

4.1. Aplicaciones móviles

Las tecnologías móviles son parte de la vida diaria de las personas en diferentes países alrededor del mundo. “Las aplicaciones móviles, se han convertido en un pilar del desarrollo de las tecnologías de la información y comunicación, dando a los usuarios diferentes beneficios” (Diaz, 20158, p. 16).

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.

4.2. Tipos de aplicación móvil

De acuerdo con la forma en que una aplicación móvil es desarrollada, estas pueden ser categorizadas en tres tipos: Aplicaciones nativas, aplicaciones web y aplicaciones hibridas.

4.2.1. 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.

4.2.2. 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.

4.2.3. 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.

4.3. Sistema operativo Android

“Android es un sistema operativo y una plataforma software, basado en Linux para teléfonos móviles. Además, también usan este sistema operativo (aunque no es muy habitual), tabletas, netbooks, reproductores de música e incluso PC’s” (Báez, Borrego, Cordero, Cruz, González, Hernández, Zapata, 20169, p. 1).

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.

4.4. 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 está orientado a objetos y se caracteriza por correr sobre una máquina virtual. Android soporta la versión 7 y 8 de este lenguaje (Android, 2019c).

- 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 (Android, 2019c).

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.

4.4.1. 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.

4.4.2. SQLite como almacenamiento de datos

> Android tiene integrado en el propio sistema una API completa que nos permite manejar BBDD en SQLite. SQLite es un motor de bases de datos que se ha ido popularizando en los últimos años dado que maneja archivos de poco tamaño, no necesita ejecutarse en un servidor, cumple el estándar SQL-92 y, además, es de código libre. (Báez *et. al.*, 2019, p. 55)

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.

4.4.3. 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 moviles.

“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 y Barbeau, 2011, p. 383).

4.5. 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.

Inicialmente se hablaba más de diseño de software como una actividad ajena a la implementación. “Creemos que la década de 1990 será la década de la arquitectura de software. Usamos el término arquitectura, en contraste con el diseño, para evocar nociones de codificación, de abstracción, de estándares, de formación formal, y de estilo” (Perry y Wolf, 1992, p. 40).

> En general, aunque todas estas arquitecturas varían algo en sus detalles, son muy similares. Todos tienen el mismo objetivo, que es la separación de componentes. Todos logran esta separación dividiendo el software en capas. Cada uno tiene al menos una capa para las reglas comerciales y otra capa para las interfaces de usuario y del sistema. (Martin, 2018, p. 191)

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

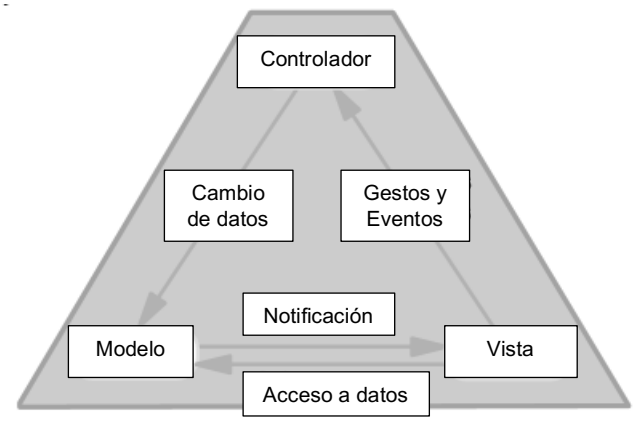
- Independencia a marcos de trabajo

- Facilidad de pruebas
- Independencia de interfaz de usuario, base de datos y cualquier agente externo

4.5.1. 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 1. **Arquitectura MVC**



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

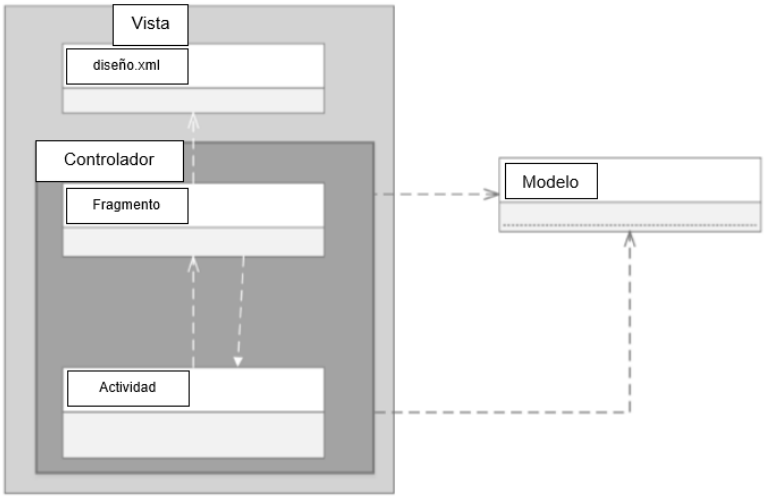
En un diseño perfecto, la vista y el controlador siempre van de la mano. Están estrechamente acoplados. En otras palabras, una vista tiene un controlador específico y el controlador solo se hace cargo de su vista

específica. Un par de vista / controlador tiene un modelo, pero un modelo puede tener más de un par. (Lou, 2016, p. 10)

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 2 puede observarse el acoplamiento que existe entre la vista y el controlador.

Figura 2. **Arquitectura MVC en Android**

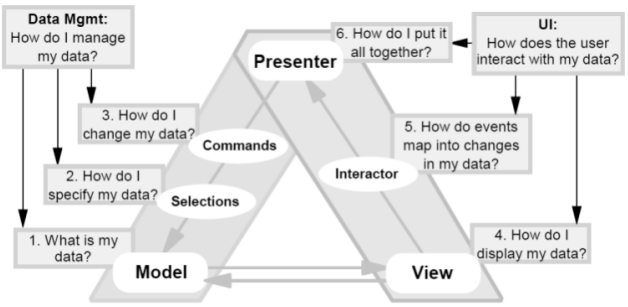


Fuente: Lou (2016). *A Comparison of Android Native App Architecture–MVC, MVP and MVVM*.

4.5.2. 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* como se observa en la figura 3. 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 3. **Arquitectura MVP**

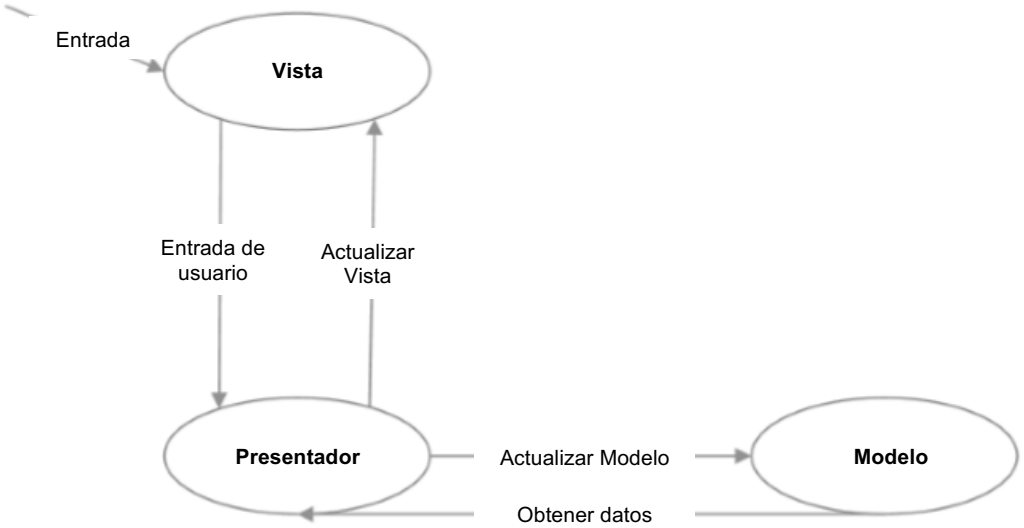


Fuente: Lou (2016). A Comparison of Android Native App Architecture–MVC, MVP and MVVM.

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, tal como se observa en la figura 4.

Figura 4. **Arquitectura MVP en Android**



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

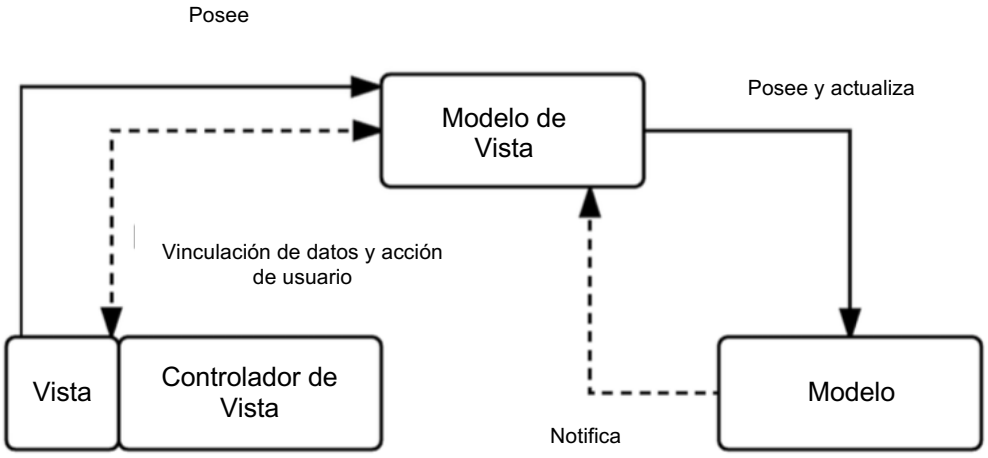
4.5.3. Arquitectura MVVM

Esta arquitectura presenta tres capas principales, siendo una variante de la arquitectura tradicional MVC. Estas capas son: Modelo, Vista de Modelo y Vista. De estas capas es que toma su nombre por sus siglas en inglés: *Model, View, View Model*.

Esta arquitectura se basa en un mecanismo de enlace de datos que facilita la separación de la capa de la vista del resto de capas mediante la eliminación de lógica en el código de esta capa. Ver figura 5.

Esto se logra encapsulando la lógica de la presentación y estados en la capa de modelo de vista. Para maximizar las oportunidades de reutilización, el *ViewModel* no debe tener ninguna referencia a las clases específicas de interfaz de usuario, elementos, controles o comportamiento.

Figura 5. **Arquitectura MVVM**



Fuente: Aljamea y Alkandari (2018). *MMVMi: A Validation Model for MVC and MVVM Design Patterns in iOS Applications*.

Esta arquitectura permite que la capa de la vista pueda ser desarrollada por diseñadores en lugar de desarrolladores. Al no contener lógica permite que sea desarrollada por un rol que no necesita conocimientos profundos de programación.

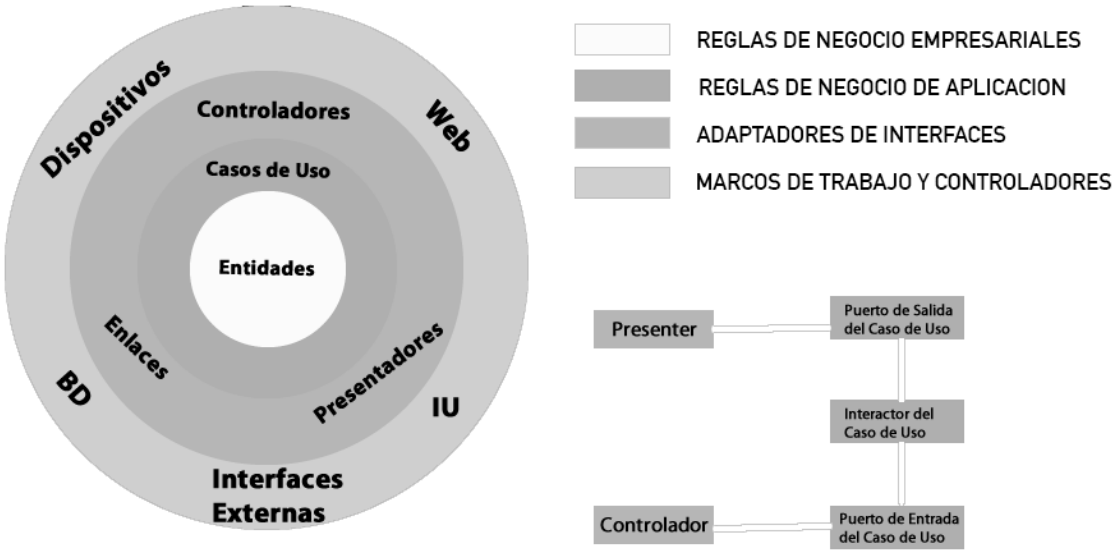
4.5.4. Arquitectura *Clean*

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 (2018). *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, éstas 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.

4.6. 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, p. 1).

4.6.1. 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 tabla IV.

Tabla IV. **Niveles de acoplamiento**

Categoría	Nivel de acoplamiento	Características
Nivel Alto	Contenido	Un componente usa datos o control mantenido por otro componente.
	Común	Los componentes comparten elementos de datos globales.
	Externo	Los componentes están anclados a entidades externas tales como dispositivos o datos externos.
Nivel Moderado	Control	Flujos de control a través de componentes.
Nivel Bajo	Estructura de datos	Datos estructurados son transferidos a través de componentes.
	Datos	Datos primitivos o arreglos de datos primitivos son transferidos entre componentes.
	Mensaje	Componentes se comunican a través de interfaces estandarizadas.

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.

4.7. Servicios web

"Los servicios web son desarrollos de software que permiten el intercambio de datos y funcionalidad de aplicaciones sobre una red. Esta soportado en diferentes estándares que garantizan la interoperabilidad entre servicios" (Machuca, 2010, p. 1).

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.

4.7.1. Firebase

Según Moroney (2017), 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.

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, *backend*, 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.

5. PRESENTACIÓN DE RESULTADOS

5.1. Diseño de aplicaciones

La investigación cuenta con una aplicación base de catálogo de comercios que ha sido actualizada para implementar las arquitecturas que están siendo evaluadas. Debido a que posteriormente se busca utilizar código de esta aplicación como base para la aplicación de museos, se ha desarrollado una abstracción de la información de la aplicación para que las pantallas principales no sean únicamente específicas a comercios.

Se ha realizado una abstracción sencilla siguiendo la estructura maestro-detalle. La pantalla principal se ha denominado Categoría y la pantalla dependiente como Detalle. Esto ha simplificado el desarrollo de ambas aplicaciones porque en cada una se ha asignado el contenido a esta abstracción como se muestra en la tabla V.

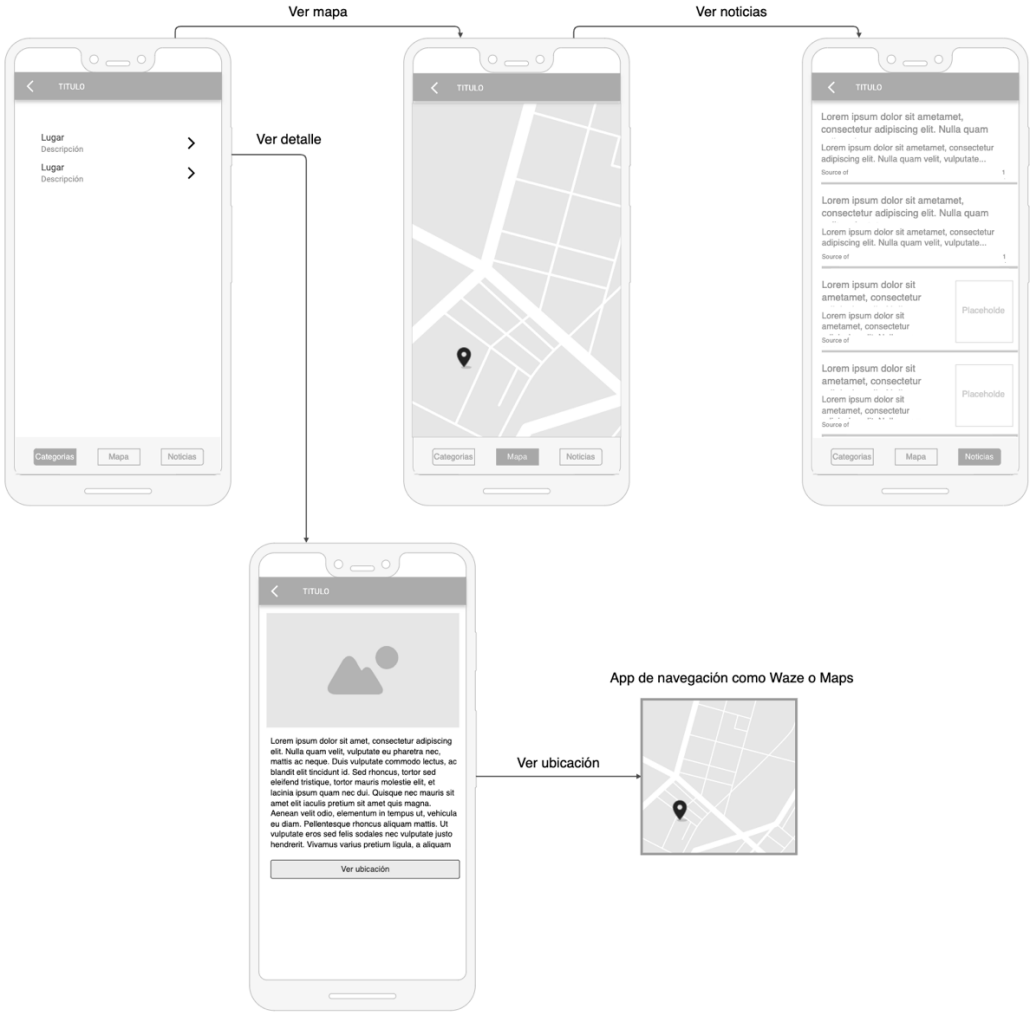
Tabla V. **Abstracción de contenido por aplicación**

Categoría	Detalle
Comercio	Información del comercio
Museo	Información del museo

Fuente: elaboración propia.

Esta abstracción colabora a que en ambas aplicaciones pueda compartirse un solo diseño general y que ambas aplicaciones puedan compartir el mismo flujo entre sus pantallas. Siendo la información y la lógica contenida dentro de su código los factores que las diferencia. Este diseño general y flujo conceptual se muestra en la figura 7.

Figura 7. **Diseño general de aplicaciones**



Fuente: elaboración propia, empleando Adobe Photoshop.

La funcionalidad principal de las aplicaciones es obtener información de una fuente de datos y mostrarla al usuario de una manera clara y sencilla. Esta información se muestra en distintas pantallas siguiendo la estructura maestro detalle.

El detalle del diseño de cada pantalla se encuentra descrito en las tablas: tabla VI, tabla VII, tabla VIII y tabla IX. Las bases del diseño gráfico de la interfaz de usuario de cada pantalla se encuentran en los anexos de la investigación.

Tabla VI. **Diseño pantalla 1 – Categorías**

Nombre: Categorías.
Descripción: Despliega un listado de las categorías disponibles y muestra una descripción corta.
Casos de Uso: • Ver detalle: El usuario hace clic en una categoría provocando que la aplicación navegue hacia la pantalla de detalle. • Ver mapa: El usuario hace clic en el ícono de mapa en el menú inferior. La aplicación navega hacia la pantalla de mapa. • Ver noticias: El usuario hace clic en el ícono de noticias en el menú inferior. La aplicación navega hacia la pantalla de noticias.
Fuentes de la información: • Base de datos en la nube alojada en servicio Firebase: CategoriasVVMIR
Tecnologías: • Firebase *SDK*

Fuente: elaboración propia.

Tabla VII. **Diseño pantalla 2 – Detalle**

Nombre: Detalle.
Descripción: Muestra información a detalle sobre la categoría seleccionada. Toda esta información es obtenida de la fuente de datos. Los elementos principales son: • El encabezado muestra una imagen asociada a la categoría. • Muestra el texto que contiene el detalle de la categoría. • Botón con texto que indica al usuario que puede navegar hacia la ubicación de esta categoría.
Casos de Uso: • Navegar: El usuario hace clic en el botón para navegar. Esta acción lleva al usuario a una aplicación externa de navegación GPS pasando los datos de ubicación como parámetro. • Volver: El usuario hace clic en la opción volver del dispositivo llevándolo de regreso a la pantalla de Categorías.
Fuentes de la información: • Base de datos en la nube alojada en servicio Firebase: CategoriasVVMIR
Tecnologías: • Firebase SDK • Picasso • Android *Intents*

Fuente: elaboración propia.

Tabla VIII. **Diseño pantalla 3 – Mapa**

Nombre: Mapa.
Descripción: Muestra un mapa ubicando todas las categorías disponibles en la fuente de datos.
Casos de Uso: • Mover mapa: El usuario puede interactuar con el mapa para ver distintas partes del mapa y las categorías ubicadas en estas secciones. • Ver categorías: El usuario hace clic en el ícono de categorías en el menú inferior. La aplicación navega hacia la pantalla de mapa. • Ver noticias: El usuario hace clic en el ícono de noticias en el menú inferior. La aplicación navega hacia la pantalla de noticias.

Continuación tabla VIII.

Fuentes de la información: • Base de datos en la nube alojada en servicio Firebase: CategoriasVVMIR
Tecnologías: • Firebase *SDK* • Google Maps

Fuente: elaboración propia.

Tabla IX. **Diseño pantalla 4 – Noticias**

Nombre: Noticias.
Descripción: Se realiza una carga interna dentro de la aplicación del sitio web relacionado a las noticias de la aplicación.
Casos de Uso: • Navegar: El usuario puede navegar libremente entre el sitio web de noticias. • Ver categorías: La aplicación navega hacia la pantalla de mapa. • Ver mapa: La aplicación navega hacia la pantalla de mapa.
Fuentes de la información: • Sitio web
Tecnologías: • Android Webviews

Fuente: elaboración propia.

5.1.1. Arquitectura tecnológica

Todas las aplicaciones desarrolladas en la investigación han sido desarrolladas utilizando el lenguaje de desarrollo Kotlin y el *SDK* de desarrollo nativo para Android.

Las librerías que fueron utilizadas son las siguientes:

- Google Maps: ha sido utilizada para mostrar todas las ubicaciones en un mapa dentro de una pantalla de la aplicación.

- Hilt: facilita la inyección de dependencias en Android.

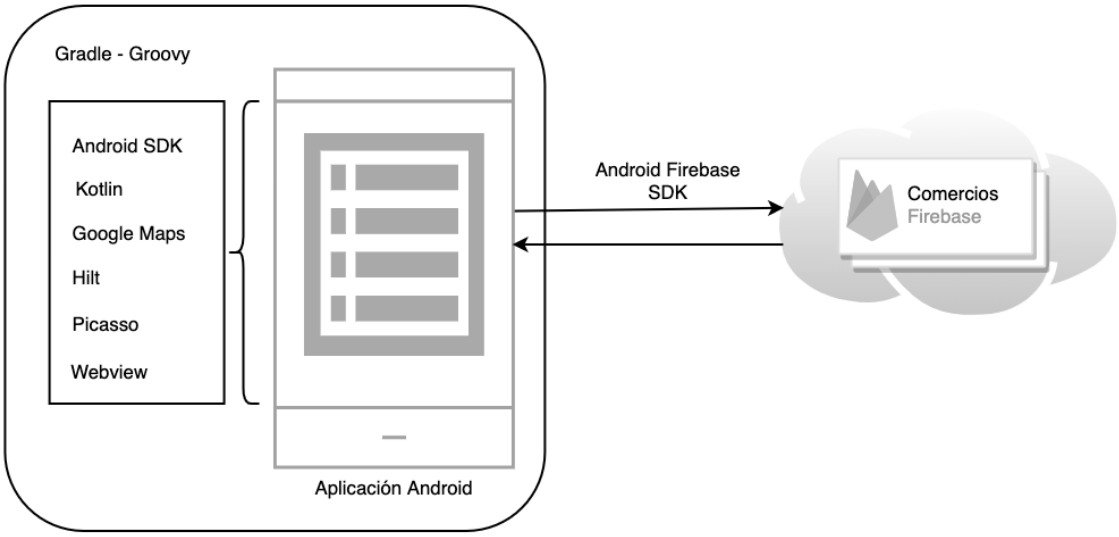
- Picasso: facilita la carga de imágenes dentro de una vista. Las imágenes mostradas son cargadas desde una dirección en internet.

- *Webview*: permite cargar un sitio web dentro de una aplicación nativa Android.

El sistema de compilación ha sido desarrollado utilizando la herramienta Gradle mediante el lenguaje de programación Groovy. Este sistema facilita la definición e importación de las dependencias externas de la aplicación. También ayuda a construir los archivos ejecutables finales con los que es distribuida la aplicación.

Para la comunicación con los servicios remotos en la nube de Firebase se ha utilizado el Android Firebase *SDK* que permite obtener la información del servicio de base de datos en tiempo real donde se almacena la información de las aplicaciones. La arquitectura tecnológica completa se ejemplifica en la figura 8.

Figura 8. **Arquitectura tecnológica**



Fuente: elaboración propia, empleando Diagrams.net.

5.1.2. Arquitectura de datos

Es en la arquitectura de datos donde las aplicaciones desarrolladas difieren. Cada aplicación accede a una fuente de datos distinta en las cuales la estructura también cambia.

Estas estructuras son las que comúnmente guían el desarrollo de un software y que llegan a estar ancladas hasta las interfaces de usuario. Para la arquitectura que se propone, las estructuras son un área muy importante y se logra aislar en una capa propia.

La aplicación de catálogo de comercios accede a una base de datos alojada en Firebase con el nombre VVMIR. Almacena la información en formato JSON y su estructura ha sido definida en inglés. La estructura de los datos en formato JSON se muestra en la figura 9.

Figura 9. **Base de datos VVMIR para catálogo de comercios**

```
{
    "categories": [
        {
            "detail": {
                "description": String,
                "imgUrl": String,
                "latitude": Long,
                "longitude": Long,
                "shortDescription": String
            },
            "name": String
        }
    ]
}
```

Fuente: elaboración propia, empleando Atom.

Por otro lado, la aplicación de catálogo de museos accede también a una base de datos alojada en Firebase con el nombre VVMIR-Museos y utiliza el formato JSON para almacenar la información. Esta base de datos posee una estructura con diferentes tipos de campos y definida en español. La estructura JSON de esta base de datos se muestra en la figura 10.

Reemplazar una estructura de datos en una aplicación fuertemente acoplada con las características que se muestran anteriormente resulta ser una tarea muy exigente, especialmente por la diferencia entre los tipos de los registros definidos y los idiomas de las bases de datos. Esta tarea se ha realizado únicamente en la aplicación desarrollada con la arquitectura propuesta con el fin de demostrar la simplificación de este proceso como se detalla más adelante.

Figura 10. **Base de datos VVMIR-Museos para catálogo de museos**

```
{
    "museos": [
        {
            "foto": String,
            "latitud": String,
            "longitud": String,
            "mensajeCorto": String,
            "mensajeLargo": String,
            "nombre": String
        }
    ]
}
```

Fuente: elaboración propia, empleando Atom.

5.2. Desarrollo de aplicación catálogo de comercios

La aplicación de catálogo de comercios base con la que se contaba anteriormente fue actualizada a Kotlin para que la investigación sea más relevante a las tecnologías que se utilizan en la actualidad. Se utilizó la aplicación anterior como guía para imitar las funcionalidades con las que ya se contaban.

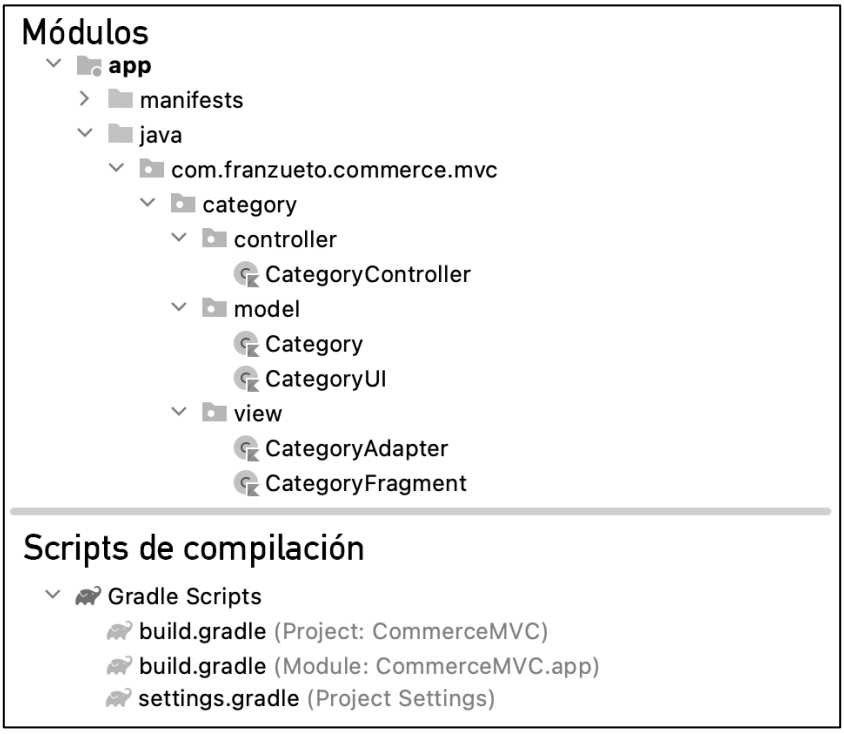
5.2.1. Arquitectura MVC

La primera aplicación ha sido desarrollada con la intención de ser una base para los siguientes pasos en los que fue actualizada siguiendo una arquitectura más moderna y posteriormente utilizando la arquitectura planteada en la investigación.

Esta arquitectura brinda un orden inicial al proyecto gracias a la definición de modelos que abstraen la información que es desplegada en la interfaz de usuario y a la vez obtenida desde los servicios remotos.

La estructura del código desarrollado al utilizar la arquitectura MVC se presenta en la figura 11.

Figura 11. **Estructura de código con arquitectura MVC**



Fuente: elaboración propia, empleando Adobe Photoshop.

Los detalles del código necesario para desarrollar esta aplicación se muestran en la tabla X. Esta información se encuentra relacionada con la complejidad y esfuerzo requerido para desarrollar la aplicación.

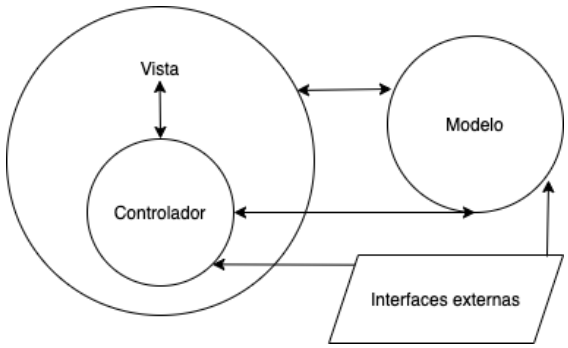
Tabla X. **Mediciones de código en aplicación *MVC***

Variable	Total
Cantidad de archivos Kotlin	10
Líneas de código Kotlin	262
Cantidad de archivos Gradle	3
Líneas de código Gradle	73

Fuente: elaboración propia.

Como se ha observado en investigaciones anteriores, debido a la estructura de las aplicaciones desarrolladas con el *SDK* nativo de Android, las capas de esta arquitectura quedan ligadas entre ellas. El resultado del análisis de las interacciones entre las capas desarrolladas dentro de la aplicación se describe en la figura 12.

Figura 12. **Diagrama de interacciones entre capas en aplicación MVC**



Fuente: elaboración propia, empleando Diagrams.net.

Los resultados de las mediciones realizadas al código para obtener el nivel de acoplamiento se encuentran descritos en la tabla XI y tabla XII.

Tabla XI. **Medición de acoplamiento en aplicación MVC**

	Nivel de acoplamiento						
	Bajo			**Moderado**		**Alto**	
	Mensaje	**Datos**	**Datos estructurados**	**Control**	**Externo**	**Común**	**Contenido**
Modelo		x	x	**x**			
Vista		x	x	**x**			
Controlador		x	x	x	**x**	**x**	

Fuente: elaboración propia.

Tabla XII. **Resultados aplicación MVC**

Aplicación: Catálogo de comercios		
Arquitectura utilizada: Modelo vista controlador (*MVC)*		
Cantidad de módulos desarrollados: 1		
Capas distinguibles desarrolladas: 3		
Capa	Descripción	Nivel de acoplamiento
Modelo	El control en esta capa fluye a través de distintos componentes y se vuelve dependiente de la vista.	Moderado
Vista	Tanto el modelo y el controlador poseen control sobre la vista.	Moderado
Controlador	Altamente asociado a entidades externas. El flujo de datos se une entre las estructuras relacionadas a la base de datos y modelos desplegados en la vista.	Alto

Fuente: elaboración propia.

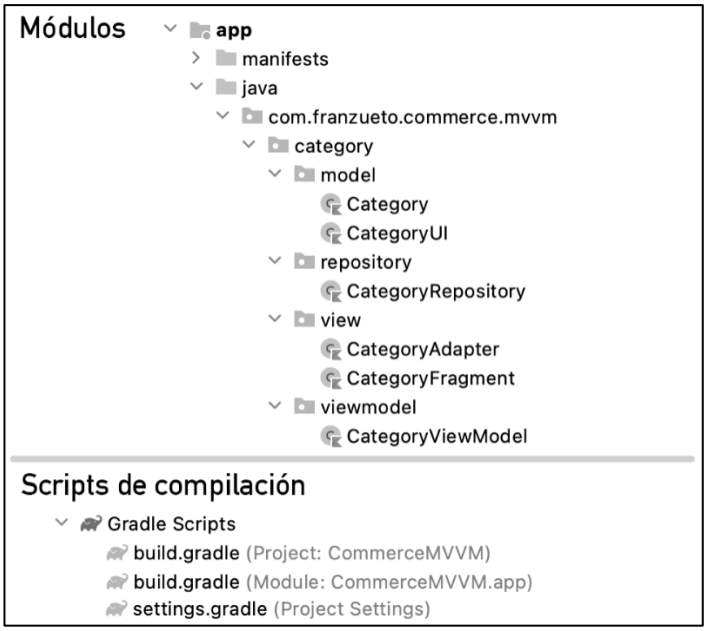
Por los resultados obtenidos al analizar las capas desarrolladas utilizando la arquitectura MVC puede resumirse que el código de la aplicación se encuentra acoplado a un nivel moderado-alto.

5.2.2. Arquitectura MVVM

La aplicación de catálogo de comercios ha sido desarrollada una segunda vez; ahora utilizando la arquitectura sugerida por Google actualmente para desarrollar aplicaciones nativas Android. Esta es la arquitectura MVVM, la cual se acopla mejor a las librerías modernas que provee el *SDK* de Android.

La estructura del código desarrollado al utilizar la arquitectura MVVM se presenta en la figura 13.

Figura 13. **Estructura de código con arquitectura MVVM**



Fuente: elaboración propia, empleando Adobe Photoshop.

Los detalles del código necesario para desarrollar esta aplicación se muestran en la tabla XIII.

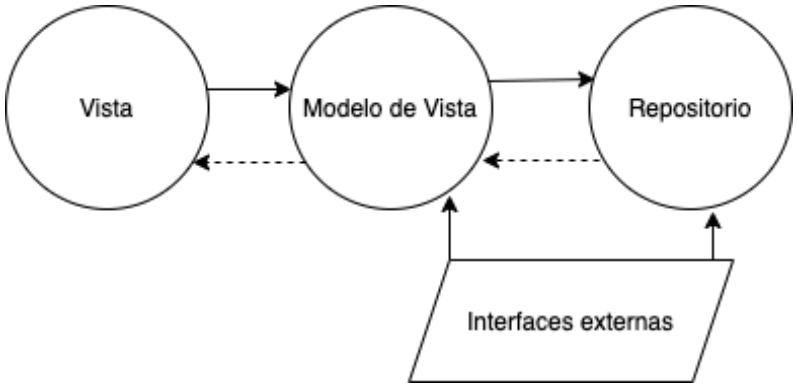
Tabla XIII. **Mediciones de código en aplicación MVVM**

Variable	Total
Cantidad de archivos Kotlin	17
Líneas de código Kotlin	362
Cantidad de archivos Gradle	3
Líneas de código Gradle	80

Fuente: elaboración propia.

Al analizar las interacciones entre las capas desarrolladas se observa una mayor separación entre sus distintos componentes. La vista se encuentra separada de la lógica de negocio gracias a que la capa modelo-de-vista realiza una separación completa. Las interfaces externas se encuentran ligadas al repositorio y pueden llegar a asociarse con la vista de modelo como se observa en la figura 14.

Figura 14. **Diagrama de interacciones entre capas en aplicación MVVM**



Fuente: elaboración propia, empleando Diagrams.net.

Los resultados de las mediciones realizadas al código para obtener el nivel de acoplamiento se encuentran descritos en la tabla XIV y tabla XV.

Tabla XIV. **Medición de acoplamiento en aplicación MVVM**

	Nivel de acoplamiento						
	Bajo			**Moderado**		**Alto**	
	Mensaje	**Datos**	**Datos estructurados**	**Control**	**Externo**	**Común**	**Contenido**
Vista		x	x				
Modelo de Vista		x	x	x	x		
Repositorio		x	x		x		

Fuente: elaboración propia.

Tabla XV. **Resultados aplicación MVVM**

Aplicación: Catálogo de comercios		
Arquitectura utilizada: Modelo de Vista-Vista-Modelo (MVVM*)*		
Cantidad de módulos desarrollados: 1		
Capas distinguibles desarrolladas: 3		
Capa	Descripción	Nivel de acoplamiento
Vista	La información fluye mediante datos primitivos y estructurados únicamente, manteniendo esta capa aislada del resto de la arquitectura.	Bajo
Modelo de Vista	El control se mantiene en comunicación mediante datos primitivos y estructurados, pero depende de otros componentes y puede depender de interfaces externas.	Alto

Continuación tabla XV.

Repositorio (Modelo)	El flujo de información se realiza mediante datos primitivos y estructurados. El componente depende directamente de interfaces externas.	Alto

Fuente: elaboración propia.

Por los resultados obtenidos al analizar las capas desarrolladas utilizando la arquitectura MVVM puede resumirse que el código de la aplicación se encuentra acoplado a un nivel moderado-alto, aunque la capa de la vista posee un nivel bajo de acoplamiento.

5.2.3. Arquitectura VVMIR

La arquitectura propuesta ha sido nombrada VVMIR; estas siglas representan los nombres en inglés de cada capa planteada en la arquitectura. Estas son:

- Vista (*View*): contiene las vistas desarrolladas directamente utilizando el SDK de Android. Se comunica directamente con el modelo de vista para mantener actualizado su estado mediante el patrón observador.

- Modelo de vista (*ViewModel*): esta capa es un modelo que representa el estado de la vista, pero no contiene ninguna comunicación con dicha capa. Depende del contrato definido en la capa de lógica de negocio (*Interactor*).
- Lógica de negocio (*Interactor*): contiene la lógica de negocio y los modelos específicos de la aplicación declarados en su contrato. Depende

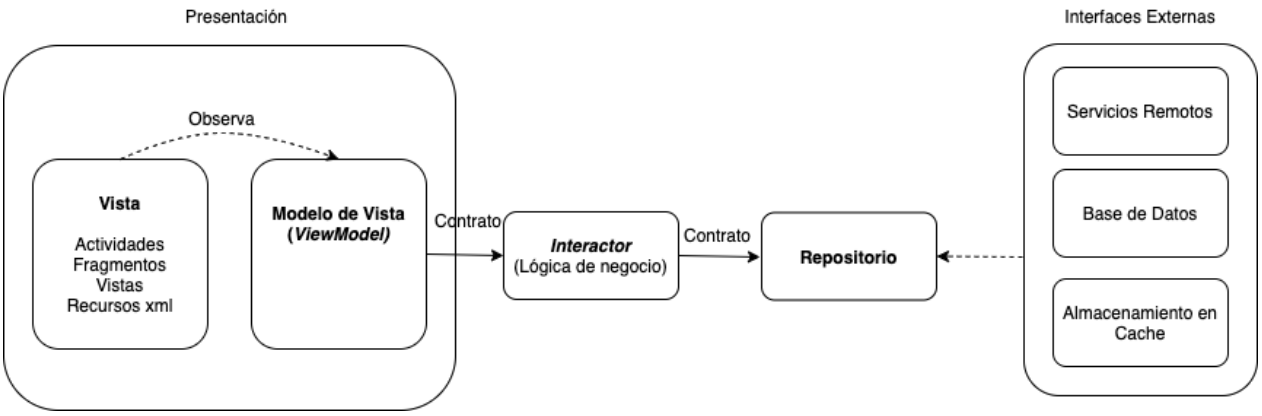
únicamente de la capa de repositorio mediante los contratos definidos. Las respuestas de esta capa son mediante datos primitivos o datos estructurados declarados en los contratos.

- Repositorio: realiza la definición abstracta de las acciones permitidas para obtener datos de cualquier fuente externa. Maneja el acceso a cualquier *framework* externo sin tener conocimiento directo del mismo mediante la definición de contratos y haciendo uso de los conceptos de inversión de control e inversión de dependencias.

Interfaces externas: no es parte directa de la arquitectura por ser un factor externo, pero cabe mencionar que la arquitectura sugiere realizar una abstracción que encapsule cualquier *framework* utilizado mediante definiciones de contratos abstractos para ser llamados desde el repositorio de datos sin que se conozca explícitamente cuales son. Aquí se incluye cualquier biblioteca externa o marco de trabajo de Android que pueda ser reemplazado en un futuro.

La arquitectura ha sido basada en la arquitectura MVVM que ya se ha analizado previamente debido a que hace un muy buen trabajo aislando la vista del resto de la lógica de la aplicación. También se ha basado en la arquitectura *Clean*, expandiendo en un mayor número las capas vistas en arquitecturas anteriormente para desacoplar las interfaces externas totalmente. El diseño de la arquitectura se detalla en la figura 15.

Figura 15. **Arquitectura VVMIR**



Fuente: elaboración propia, empleando Diagrams.net.

La aplicación de catálogo de comercios ha sido desarrollada por tercera vez; esta vez utilizando la arquitectura VVIMR. Los detalles del código necesario para desarrollar esta aplicación se muestran en la tabla XVI.

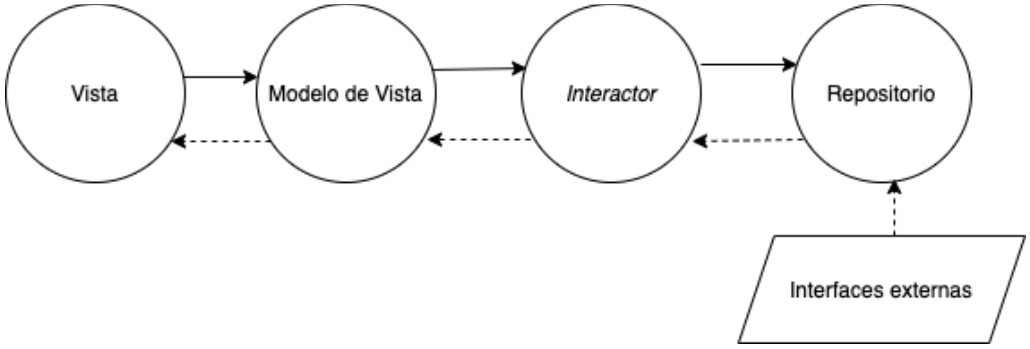
Tabla XVI. **Mediciones de código en aplicación VVMIR**

Variable	Total
Cantidad de archivos Kotlin	41
Líneas de código Kotlin	680
Cantidad de archivos Gradle	8
Líneas de código Gradle	254

Fuente: elaboración propia.

El resultado de analizar las interacciones entre las capas desarrolladas en esta aplicación se describe en la figura 16.

Figura 16. **Diagrama de interacciones entre capas en aplicación VVMIR**



Fuente: elaboración propia, empleando Diagrams.net.

Los resultados de la medición de acoplamiento realizadas al código de la aplicación que se desarrolló utilizando la arquitectura VVMIR se encuentran descritos en la tabla XVII.

Tabla XVII. **Medición de acoplamiento en aplicación VVMIR**

	Nivel de acoplamiento						
	Bajo			Moderado	Alto		
	Mensaje	Datos	Datos estructurados	Control	Externo	Común	Contenido
Vista		x	x				
Modelo de Vista	x	x	x	x			
Interactor	x	x	x	x			
Repositorio	x	x	x	x			

Fuente: elaboración propia.

En la tabla XVIII se amplían los detalles de los resultados de utilizar la arquitectura VVMIR. Estos resultados muestran que el código de la aplicación se

encuentra acoplado a un nivel moderado. Manteniendo la ventaja de la arquitectura MVVM que asegura que la capa de la vista posee un nivel bajo de acoplamiento.

Tabla XVIII. **Resultados aplicación VVMIR**

Aplicación: Catálogo de comercios		
Arquitectura utilizada: Arquitectura propuesta (VVMIR*)*		
Cantidad de módulos desarrollados: 4		
Capas distinguibles desarrolladas: 4		
Capa	Descripción	Nivel de acoplamiento
Vista	La información fluye mediante datos primitivos y estructurados únicamente, manteniendo esta capa aislada del resto de la arquitectura.	Bajo
Modelo de Vista	El flujo de información se realiza mediante datos primitivos y estructurados. Se comunica mediante interfaces abstractas. El control fluye entre componentes mediante un contrato definido.	Moderado
Lógica de negocio (*Interactor)*	El flujo de información se realiza mediante datos primitivos y estructurados. Se comunica mediante interfaces abstractas. El control fluye entre componentes mediante un contrato definido.	Moderado
Repositorio	El flujo de información se realiza mediante datos primitivos y estructurados. Se comunica mediante interfaces abstractas. El control fluye entre componentes mediante un contrato definido.	Moderado

Fuente: elaboración propia.

Para desarrollar esta aplicación se crearon módulos de proyectos independientes en *Android Studio.* Estos fueron unidos mediante un módulo de aplicación que configura las dependencias de cada módulo externo.

En la figura 17 se ilustran los módulos independientes que fueron desarrollados, los cuales son 7 en total, así como también los scripts de compilación Gradle que permiten configurar y compilar independientemente cada módulo.

Figura 17. **Módulos desarrollados en aplicación VVMIR**

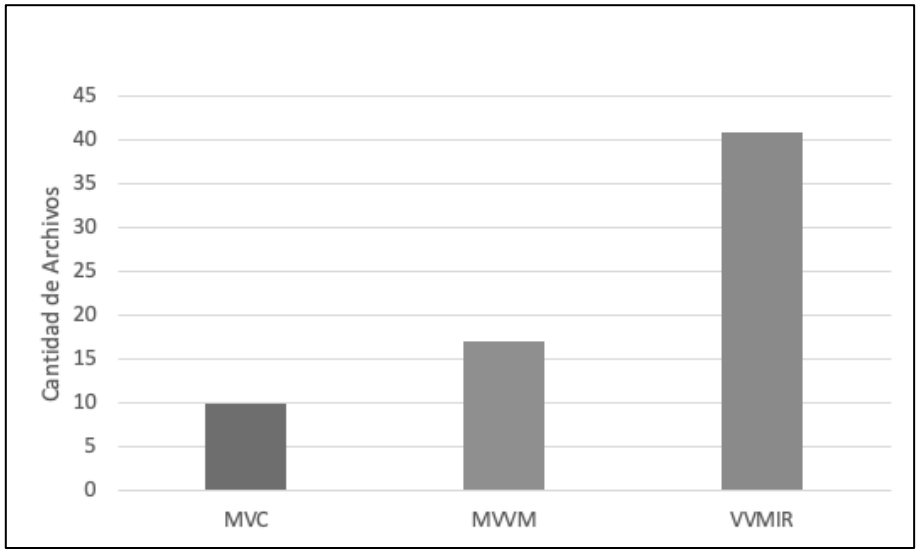
Fuente: elaboración propia, empleando Adobe Photoshop.

Se han desarrollado módulos independientes para la definición de interfaces abstractas para la capa de lógica de negocios y de repositorio. Esto hace posible que se puedan generar posteriormente distintas versiones de implementaciones y que realizar cambios requiera menos alteraciones en los demás módulos.

5.3. Descripción de resultados

Se han realizado mediciones sobre el código de cada versión desarrollada de la aplicación de catálogo de comercios para comprender el esfuerzo realizado. En la figura 18 se muestran los resultados de los archivos de código Kotlin que contienen toda la lógica de las aplicaciones.

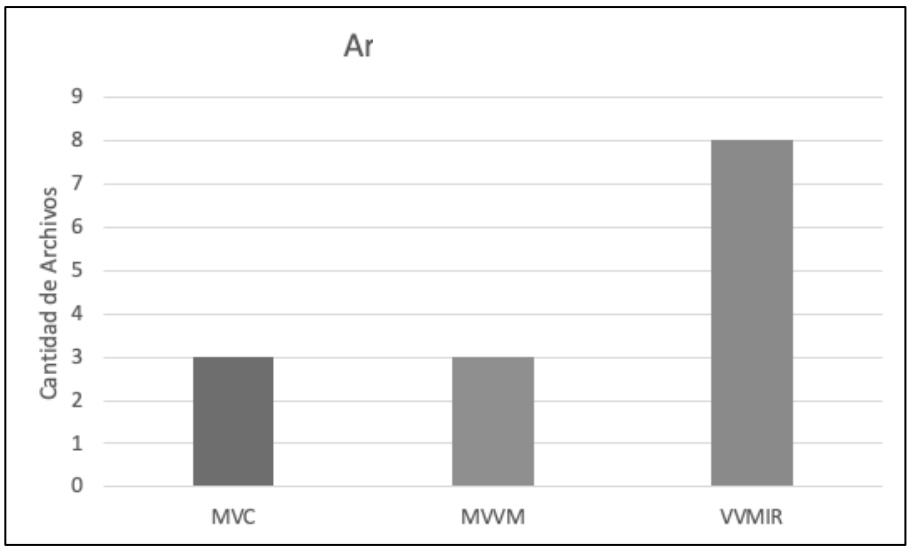
Figura 18. **Archivos Kotlin**



Fuente: elaboración propia, empleando Microsoft Excel.

En la figura 19 se presentan los resultados de los archivos de configuración Gradle que contienen la lógica de compilación y configuración de los proyectos.

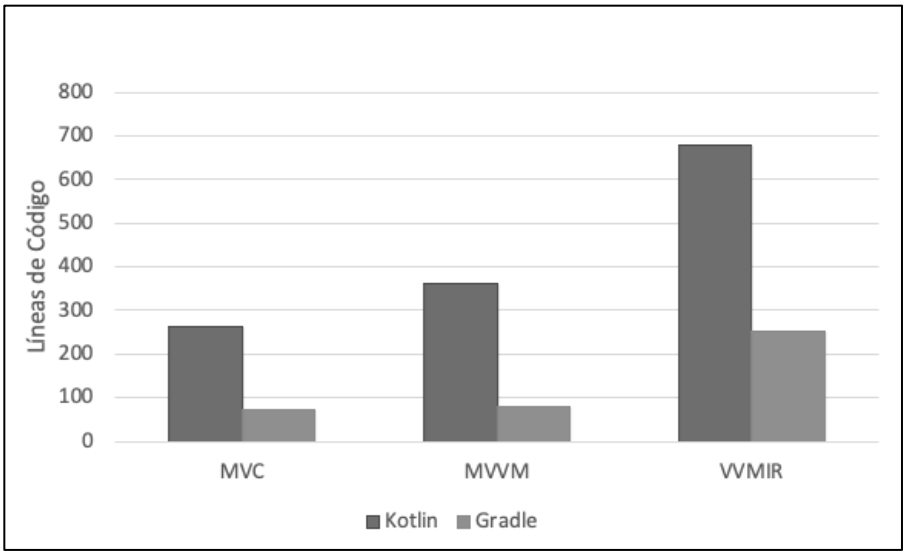
Figura 19. **Archivos Gradle**



Fuente: elaboración propia, empleando Microsoft Excel.

Las líneas de código son una medida que representa el esfuerzo realizado para desarrollar cualquier software. Esta medida es muy conocida por sus siglas en inglés *LOC*. Los resultados de medir esta variable en las aplicaciones se muestran en la figura 20.

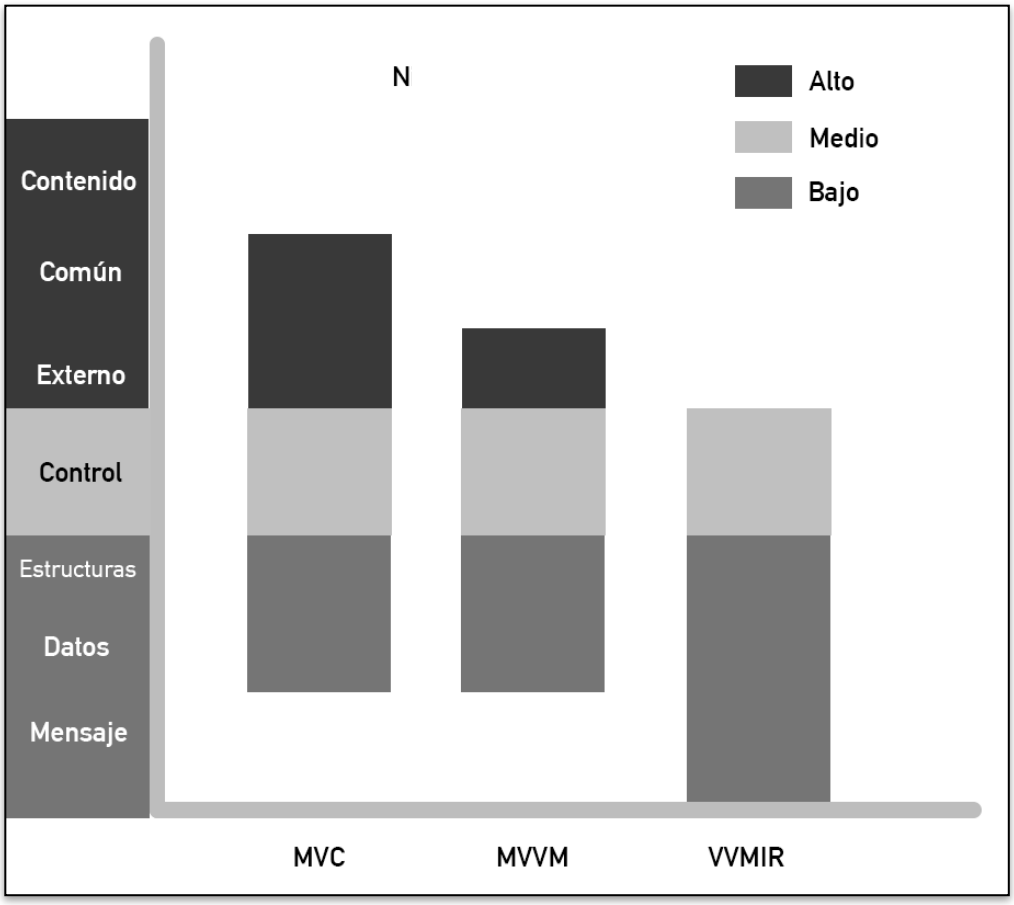
Figura 20. **Líneas de código**



Fuente: elaboración propia, empleando Microsoft Excel.

Los resultados al medir el nivel de acoplamiento fueron disminuyendo a medida que se aplicaba una arquitectura más detallada en las aplicaciones. La arquitectura MVC fue la que presentó un mayor nivel de acoplamiento y la arquitectura propuesta VVMIR la que presentó el menor nivel de acoplamiento. En la figura 21 se describen estos resultados.

Figura 21. **Resultados de nivel de acoplamiento**



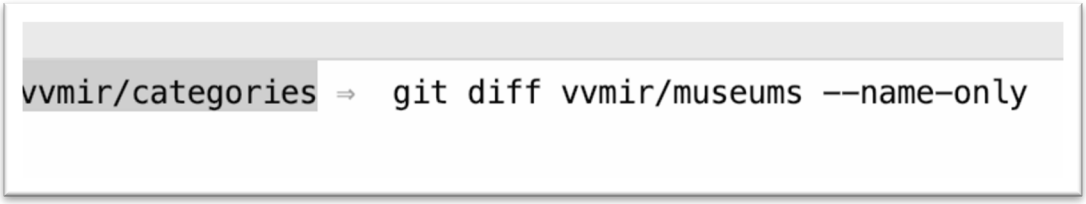
Fuente: elaboración propia.

5.4. Aplicación catálogo de museos de Guatemala

Para comprobar la eficiencia de la arquitectura VVMIR se desarrolló una aplicación de catálogo de museos de Guatemala modificando la aplicación de catálogo de comercios que se desarrolló con las arquitecturas MVC, MVVM y VVMIR.

Durante el desarrollo de todas las aplicaciones se utilizó la herramienta de control de versiones Git. Esta herramienta permite conocer las diferencias entre el código en distintos periodos de tiempo o estados, según se hayan configurado y guardado. Cada aplicación desarrollada fue almacenada en una rama de Git distinta. Para comparar los cambios realizados entre la aplicación de catálogo de comercios y la aplicación de catálogo de museos se ejecutó la instrucción mostrada en la figura 22.

Figura 22. **Comando git diff**



Fuente: elaboración propia, empleando iTerm.

Las diferencias mencionadas entre estos archivos y el resultado del comando ejecutado se detallan en la figura 23.

Figura 23. **Diferencias entre aplicación de comercios y museos VVMIR**

Fuente: elaboración propia, empleando Atom.

En la tabla XIX se detallan los resultados de la reutilización de código sobre archivos y *LOC* que se lograron sin necesitar modificación alguna para crear la aplicación de museos de Guatemala.

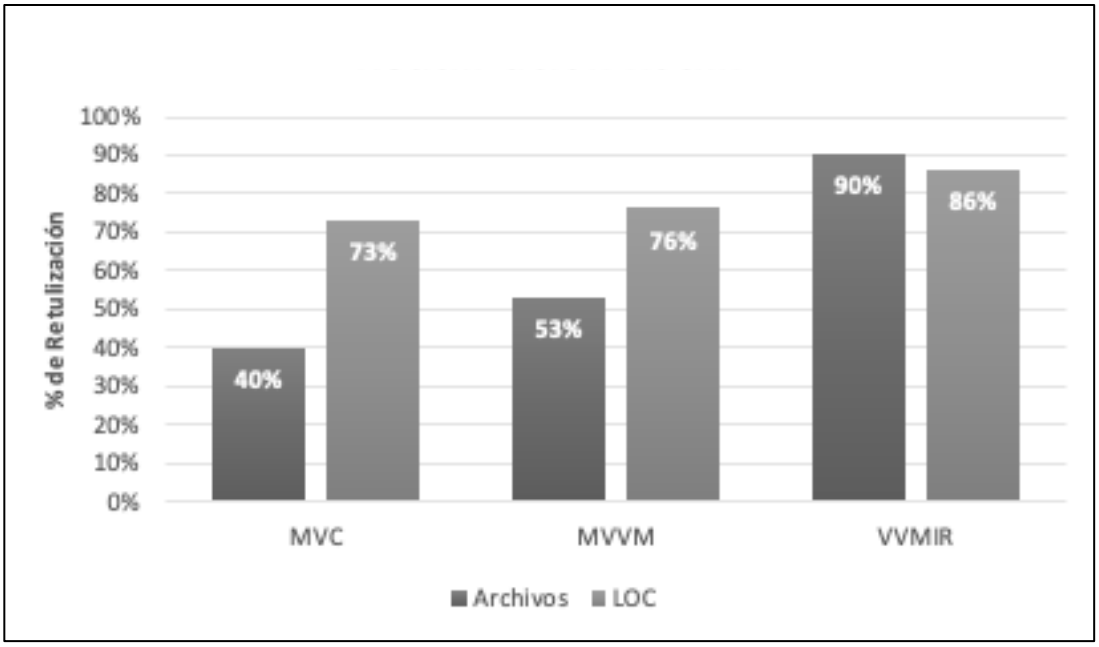
Tabla XIX. **Porcentaje de reutilización**

Lenguaje	Arquitectura	Archivos modificados	% archivos reutilizados	LOC modificados	% LOC reutilizados
Kotlin	MVC	6	40 %	70	73.28 %
	MVVM	8	52.95 %	86	76.24 %
	VVMIR	4	90.24 %	92	86.47 %
Gradle	MVC	3	100 %	0	100 %
	MVVM	3	100 %	0	100 %
	VVMIR	1	87.5 %	2	99.21 %

Fuente: elaboración propia.

Con la arquitectura VVMIR se obtuvo una reutilización de código Kotlin del 90.24 % del total de los archivos y el 86.47 **%** de líneas de código, siendo estos datos mayores comparados con los resultados obtenidos con las otras dos arquitecturas. Estos datos se observan en la figura 24.

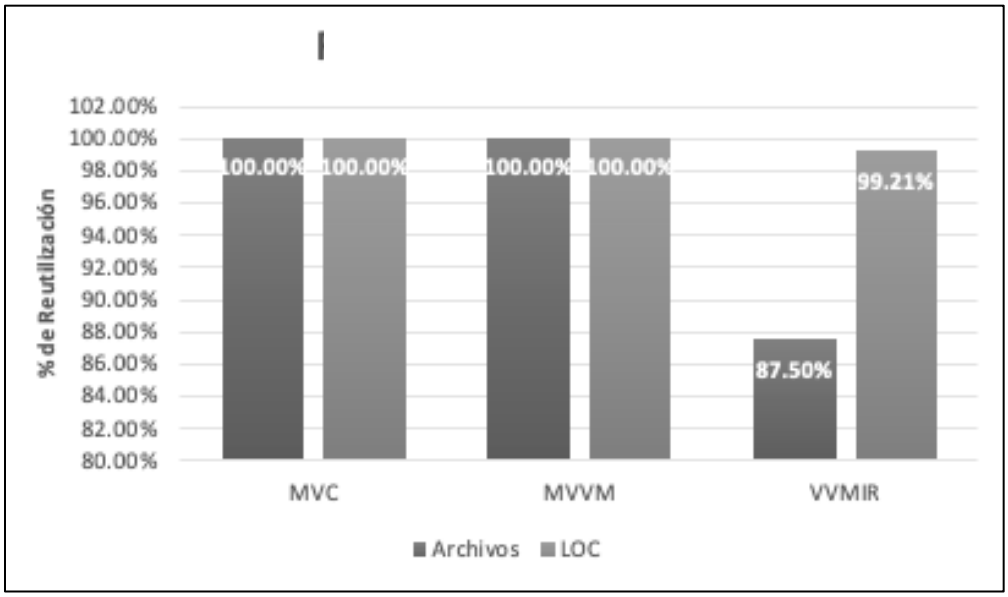
Figura 24. **Reutilización de código Kotlin**



Fuente: elaboración propia, empleando Microsoft Excel.

En el caso de los archivos Gradle, se requirió un mayor esfuerzo comparado con las otras arquitecturas, teniendo un resultado de reutilización del 87.5 % de los archivos. Fue únicamente necesario modificar 2 líneas de código que representan una reutilización de 99.21 % del total de las líneas de código tal como se presenta en la figura 25.

Figura 25. **Reutilización de código Gradle**



Fuente: elaboración propia.

Al utilizar la arquitectura VVMIR se identificaron dos capas principales en las que se aislaron los cambios necesarios para cumplir con los requerimientos de esta aplicación. Primero, fue necesario actualizar contenido estático que se encuentra en la capa de vista. Segundo, debido a que los modelos de la base de datos son completamente distintos, se identificó que una forma sencilla de realizar los cambios era desarrollando un nuevo módulo en la capa de repositorio para la aplicación de museos, para luego ser reemplazado mediante los archivos de configuración Gradle.

La configuración del nuevo módulo de repositorio en los archivos de compilación Gradle resultó bastante sencilla gracias a que la arquitectura se basa en interfaces abstractas y esto colaboró a que el resto de los módulos no necesitaran ser actualizados.

En total se necesitaron 8 archivos para desarrollar la nueva aplicación basándose en la aplicación anterior. De los cuales 4 archivos son nuevos (los de la capa de repositorio) y los otros 4 archivos (de configuración y en la capa de vista) fueron modificaciones menores a los archivos de contenido visual.

6. DISCUSIÓN DE RESULTADOS

Todas las aplicaciones desarrolladas en los experimentos de la investigación fueron implementadas satisfactoriamente cumpliendo con los requisitos que se esperaban dentro del diseño y cumpliendo el tiempo de desarrollo dentro de lo esperado.

Una decisión muy importante que influyó en este éxito fue haber dejado a un lado el desarrollo de un *backend* y haber decidido utilizar Firebase como servicio remoto de base de datos. Este servicio ayudó a que el desarrollo pudiera ser enfocado en las aplicaciones móviles sin invertir mucho tiempo y recursos en configuraciones de base de datos o servidores.

Gradle es un sistema de compilación que provee *Android Studio* integrado dentro de sus herramientas y ha funcionado de buena manera especialmente al desarrollar módulos independientes. Hacer uso de esta herramienta ha dado buenos resultados cuando se ha necesitado reemplazar módulos facilitando el proceso a un muy buen nivel.

6.1. Medición de niveles de acoplamiento

Para medir el nivel de acoplamiento en las aplicaciones se utilizó el método definido en la investigación de Lou (2016). Este método ha resultado muy efectivo para analizar las características del código en cada capa de las arquitecturas.

El método requiere que quien esté analizando el código conozca conceptos de programación a un buen nivel y no requiere que la persona conozca sobre la lógica de negocio o esté familiarizado con el código. Esta es una ventaja ya que el analista puede ser alguien que no necesariamente esté integrado al equipo de desarrollo de un proyecto.

Los resultados del método pueden ser tabulados fácilmente y representados de acuerdo con los niveles que el método indica para ser posteriormente analizados.

6.2. Rendimiento de la arquitectura

Con base en los resultados obtenidos se puede concluir que la arquitectura planteada VVMIR ha sido exitosa en alcanzar los objetivos planteados respecto a reducir el nivel de acoplamiento en una aplicación. Esto se observa al comparar los resultados del nivel de acoplamiento entre las tres aplicaciones como se detalla en la tabla XX.

Tabla XX. **Resumen de acoplamiento en arquitecturas**

Arquitectura	Nivel de acoplamiento
MVC	Moderado-alto
MVVM	Moderado-alto
VVMIR	Moderado

Fuente: elaboración propia.

Se ha comprobado mediante el desarrollo de la aplicación de catálogo de museos que la arquitectura ha permitido reutilizar el 90 % del código de la aplicación de catálogo de comercios sin la necesidad de realizar modificaciones. Esta es información muy importante para la investigación porque coincide con los datos obtenidos en la medición del nivel de acoplamiento porque la arquitectura ha ayudado a reducir estos niveles, permitiendo que los módulos desarrollados con base en las capas de la arquitectura puedan ser fácilmente reutilizados o modificados.

6.3. Impacto tecnológico

La arquitectura que ha sido planteada e implementada es bastante robusta y segrega el software en distintas capas, impactando directamente al ciclo de desarrollo y a las personas involucradas. Los roles impactados por la arquitectura son:

- Desarrolladores: las personas a cargo de este rol deben estar correctamente capacitados en la arquitectura que se está implementando para contar con una idea general de lo que será el producto final debido a que, por la naturaleza de la arquitectura de segregar el código en capas y módulos independientes, los desarrolladores pueden trabajar en pequeñas partes del código independiente. Se recomienda que tengan una idea clara del proyecto en conjunto para estar alineados al producto final.

- Diseñadores gráficos: este rol se ve afectado indirectamente por la implementación de la arquitectura. Al igual que la arquitectura MVVM, permite que personas con un menor conocimiento técnico en desarrollo de software puedan trabajar en la capa de la vista al contar con conocimientos

básicos de componentes visuales en las librerías de Android. Los diseñadores gráficos suelen contar con habilidades que les permiten hacer este tipo de trabajos en el desarrollo de sitios web por estar familiarizados con componentes visuales y gracias a la aplicación de esta arquitectura se puede iniciar a desarrollar esta práctica en aplicaciones nativas Android.

- Arquitectos, analistas y diseñadores de software: aplicar una arquitectura como la planteada en un proyecto requiere de una o varias personas en el equipo de desarrollo con un alto nivel de conocimientos en desarrollo de software. Se recomienda que exista una persona con el rol de arquitecto y que sea responsable de dirigir el proyecto para iniciar a implementar esta arquitectura. Aplicarla de una manera desorganizada provocaría que un proyecto se complique innecesariamente. Implementar una arquitectura como la que se ha planteado es una decisión que debe ser tomada por una persona, o un equipo de personas, con estos roles. Entre las ventajas que la arquitectura trae a estos roles es que pueden segregar tareas fácilmente y organizar equipos grandes de desarrollo de tal manera que distintas partes del software sean desarrolladas en paralelo.

- Gerentes, empresas y clientes: todo software tiene una finalidad y una persona a quien debe serle útil o entregado como producto final. Para estas personas resulta beneficioso que una aplicación sea construida con una arquitectura robusta porque reduce la complejidad en el mantenimiento y aumenta la capacidad de reutilización de partes del software en caso de necesitar o querer escalar el producto.

6.4. Impacto económico

Existen empresas que han desarrollado aplicaciones móviles Android para fines específicos. Estas aplicaciones tienden a quedar desactualizadas por distintas razones, entre las que se encuentra comúnmente el hecho que son difíciles de modificar porque el código suele estar altamente acoplado. Esto provoca que las actualizaciones necesiten una cantidad considerable de los recursos de la empresa para ser desarrolladas. Al aplicar la arquitectura VVMIR las aplicaciones pueden ser actualizadas por módulos, simplificando el proceso de actualización e impactando favorablemente a los recursos económicos de las empresas.

La implementación de la arquitectura también ayudaría a las empresas a continuar desarrollando nuevas aplicaciones de software mediante la reutilización de los módulos independientes de una aplicación. Esto abriría las puertas a nuevos productos o a actualizaciones más eficientes a sus productos actuales.

Dentro de la investigación los costos económicos se han mantenido bajos gracias a que únicamente fue necesario contratar a un desarrollador externo para desarrollar la aplicación base y que también colaboró con el desarrollo de módulos independientes en la aplicación de catálogos de museos. Esto ha demostrado que las empresas pueden destinar a sus colaboradores a desarrollar partes independientes de las aplicaciones e ir completando el software iterativamente conforme se van obteniendo resultados en los módulos implementados. Llevar un proyecto de manera ordenada ayuda a utilizar eficientemente los recursos de una empresa impactando favorablemente su economía.

6.5. Acciones futuras

La investigación se ha centrado en reducir el nivel de acoplamiento en el código de las aplicaciones con el fin de contribuir a empresas y desarrolladores a facilitar el mantenimiento de sus aplicaciones y reutilizar componentes de código que ya tengan desarrollados. Pero, existen otros factores con los que una arquitectura puede ser evaluada, como los siguientes:

- Rendimiento (*Performance*)
- Capacidad de ser probado (*Testability*)
- Confiabilidad (*Reliability*)
- Seguridad (*Security*)

Estas características no han sido analizadas dentro de la investigación porque están fuera de los alcances, pero son áreas de valor que se pueden indagar para verificar que la arquitectura VVMIR se desempeña correctamente.

En la información encontrada se observó que conforme se fue implementando una arquitectura más robusta, la cantidad de archivos y líneas de código necesarias se incrementó. Estos datos reflejan que es necesario un mayor esfuerzo, que se traduce en tiempo y recursos, para desarrollar una aplicación con estas arquitecturas. Por esta razón es necesario mencionar que los arquitectos, analistas y diseñadores de software son quienes deben tomar la decisión de en qué casos utilizar cada arquitectura. Debe hacerse la aclaración que para productos pequeños o empresas con equipos pequeños utilizar una arquitectura tan robusta puede no ser la mejor opción inicialmente.

Para mejorar el trabajo posteriormente se sugiere exponer la arquitectura a aplicaciones en las que interactúen más desarrolladores con el fin de obtener más información y analizar en qué punto es beneficioso para un equipo iniciar a utilizar una arquitectura como esta.

La arquitectura VVMIR puede ser implementada en otros ambientes de aplicación como por ejemplo aplicaciones iOS o *Web*. La arquitectura tuvo un buen desempeño en el desarrollo de una aplicación Android porque las tecnologías que brinda el *SDK* de Android se adaptan correctamente a las capas definidas. Un aporte futuro sería el implementar aplicaciones en otras plataformas y comparar los resultados con la presente investigación.

Por último, se sugiere realizar los experimentos siguiendo distintas metodologías de desarrollo, sobre todo las que son más populares en el desarrollo de software actualmente, como SCRUM o *Test Driven Development*, para comprobar cómo se ajusta la arquitectura a estas metodologías y las personas involucradas en los equipos de desarrollo.

CONCLUSIONES

1. Utilizar la arquitectura VVMIR para desarrollar la aplicación Android de catálogo de comercios ayudó a reducir el nivel de acoplamiento en el código del software y redujo la complejidad en las modificaciones necesarias para reutilizar su código en el desarrollo de la aplicación de catálogo de museos de Guatemala. Se logró un rendimiento en la reutilización de archivos de 90.24 % y 86.47 % de líneas de código, reduciendo el acoplamiento a un nivel moderado con características de un acoplamiento bajo.

2. Se implementó satisfactoriamente la arquitectura VVMIR, permitiendo el desarrollo de módulos independientes en la aplicación de catálogo de comercios. Se desarrollaron 4 módulos independientes con niveles de acoplamiento moderado-bajo.

3. Se implementó el método de Lou (2016), para medir el nivel de acoplamiento en las aplicaciones desarrolladas, validando la eficiencia de la arquitectura VVMIR. La información que brindó este método permitió traducir las características del software a información medible y representable para ser analizada.

4. Emplear el sistema de compilación Gradle para administrar los módulos desarrollados en un proyecto de *Android Studio* permitió reutilizar y reemplazar módulos eficientemente. En la aplicación de catálogo de museos con VVMIR se reutilizaron 2 módulos de la aplicación de catálogo de comercios al 100 % sin necesitar modificaciones.

5. La arquitectura VVMIR ha sido implementada satisfactoriamente en las aplicaciones Android y ha comprobado que puede reducir el nivel de acoplamiento en el código. Esta arquitectura también se puede implementar en otros ambientes de aplicaciones como lo son iOS o Web, donde aún se debe investigar los resultados que la arquitectura proporciona.

RECOMENDACIONES

1. Desarrollar una aplicación utilizando la arquitectura VVMIR requiere que una persona con el rol de arquitecto defina los contratos en los modelos que la arquitectura solicita para comunicar las diferentes capas. Configurar los módulos independientes a través de herramientas como Gradle.

2. Realizar la medición de acoplamiento de código resulta más fácil analizando las aplicaciones en las que se han desarrollado módulos independientes para cada capa debido a que únicamente es necesario analizar cada archivo una vez. En las aplicaciones que no se han segregado por módulos, las capas se entrelazan y es necesario analizar el mismo archivo varias veces para analizar las interacciones de cada capa. Se recomienda crear módulos con Gradle cuando la arquitectura lo permite.

3. Desarrollar el análisis de archivos y características de código asegurando que el proyecto se encuentra libre de archivos de compilación generados por herramientas como Gradle o cualquier librería externa. Es común que se autogenere código mediante otras herramientas y este no debe ser incluido dentro del análisis. Por esto se debe asegurar que el proyecto se encuentre limpio y libre de estos archivos antes de iniciar la fase de análisis.

4. Implementar aplicaciones en otras plataformas, como iOS o Web, y comparar los resultados con la presente investigación para determinar cómo se ajusta la arquitectura a otros ambientes y validar que la

arquitectura sigue cumpliendo con el objetivo de reducir el nivel de acoplamiento.

BIBLIOGRAFÍA

1. Abrahamsson, P., Hanhineva, A. y Jäälinoja, J. (mayo, 2005). Improving business agility through technical solutions: A case study on test-driven development in mobile software development. *IFIP International Working Conference on Business Agility and Information Technology Diffusion, 180*(1), 227-243.

2. Alghamdi, J. (abril, 2008). Measuring software coupling. *Arabian Journal for Science & Engineering, 33(*1b), 119-129*.*

3. Aljamea, M., y Alkandari, M. (agosto, 2018). MMVMi: A validation model for MVC and MVVM design patterns in iOS applications. *IAENG Int. J. Comput. Sci, 45*(3), 377-389.

4. Android. (5 de octubre, 2019a). *Arquitectura Android*. [Mensaje en un blog]. Recuperado de http://www.android.com/.

5. Android. (5 de octubre, 2019b*). Android Jetpack.* [Mensaje en un blog]. Recuperado de architecture:https://developer.android.com/jetpack/guide.

6. Android. (5 de octubre, 2019c). *AndroidX Releases - Architecture Components Release Notes Archive*. [Mensaje en un blog]. Recuperado de https://developer.android.com/jetpack/androidx/releases/archive/arch.

7. Android. (2 de julio, 2020). *Hilt and Dagger annotations cheat sheet.* [Mensaje en un blog]. Recuperado de https://developer.android.com/training/dependency-injection/hilt-cheatsheet?hl=es-419.

8. Arcos-Medina, G., Menéndez, J. y Vallejo, J. (enero, 2018). Comparative Study of Performance and Productivity of MVC and MVVM design patterns. *Simposio Iberoamericano en Programación Informática (Ibero-American Symposium on Computer Programming) KnE Engineering, 1*(2), 241-252.

9. Arisholm, E., Briand, L. y Foyen, A. (septiembre, 2004). Dynamic coupling measurement for object-oriented software. *IEEE Transactions on software engineering, 30*(8), 491-506.

10. Báez, M., Borrego, Á., Cordero, J., Cruz, L., González, M., Hernández, F. y Zapata, Á. (2019). *Introducción a android.* Madrid, España: Editorial E.M.E.

11. Calero, C., Moraga, A. y Piattini, M. (2010). *Calidad del producto y proceso software.* Madrid, España: Editorial Ra-Ma.

12. 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), 1-28.

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

14. Clement, J. (5 de octubre, 2019). *Mobile app usage - Statistics & Facts.* [Mensaje en un blog]. Recuperado de Statista: https://www.statista.com/topics/1002/mobile-app-usage/.

15. Diaz, G. (2015). *Carpooling GT, aplicación para compartir vehículos.* (Tesis de maestría). Universidad de San Carlos de Guatemala, Guatemala.

16. Kim, H., Choi, B. y Yoon, S. (enero, 2009). Performance testing based on test-driven development for mobile applications. *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication.* Congreso llevado a cabo en Suwon, Corea.

17. Lou, T. (2016). *A comparison of android native app architecture–mvc, mvp and mvvm.* (Tesis de maestría). Eindhoven University of Technology, Países Bajos.

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

19. Martin, R. (2018). *Clean architecture: a craftsman's guide to software structure and design.* New Jersey, Estados Unidos: Prentice Hall.

20. Moroney, L. (2017). *The Deefinitive Guide to Firebase: Build Android Apps on Google's Mobile Platform.* Seattle, Estados Unidos: Apress.

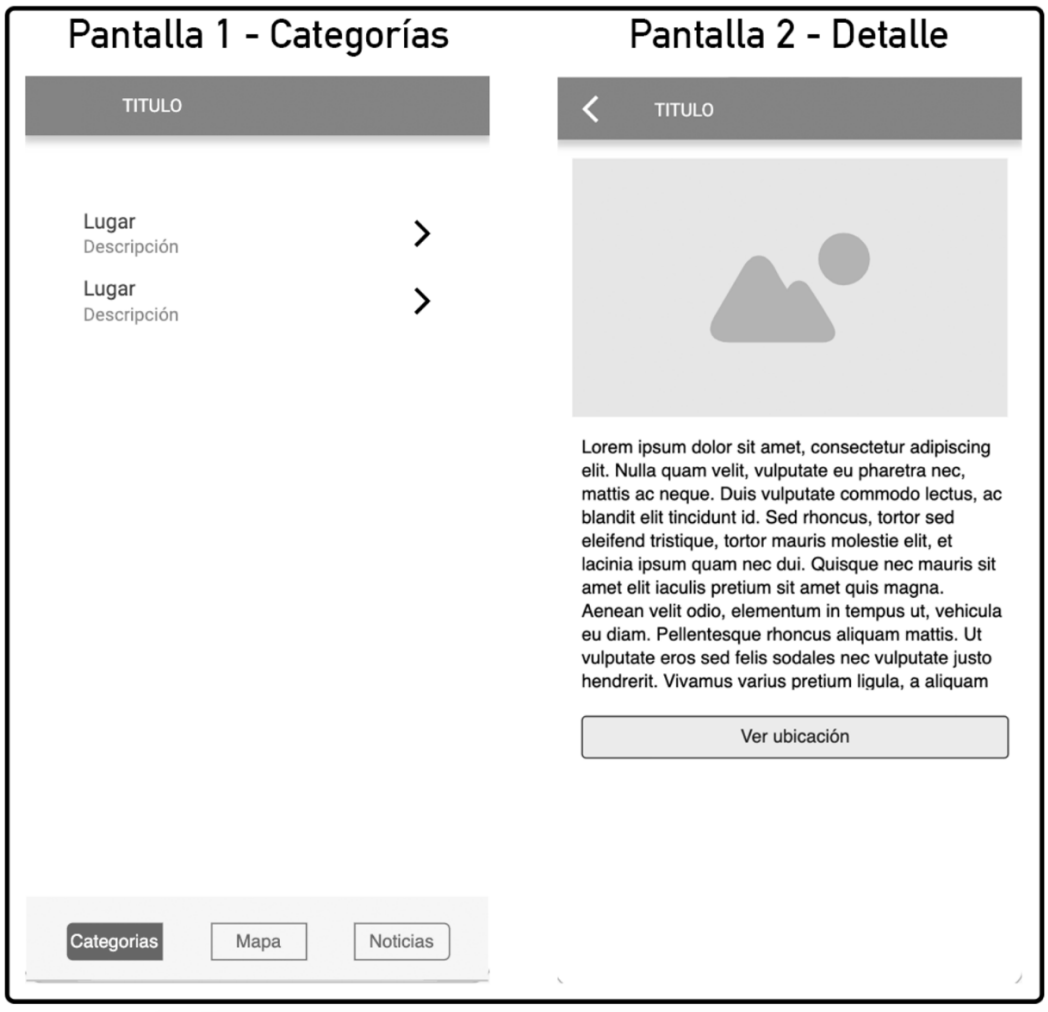
21. Perry, D., y Wolf, A. (octubre, 1992). Foundations for the study of software architecture. *Software engineering notes, 17*(4), 40-52. Recuperado de http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf.

22. Potel, M. (enero, 1996). MVP: Model-View-Presenter the Taligent programming model for C++ and Java. *Taligent Inc, 1*(1), 20-33.

23. Zandbergen, P., y Barbeau, S. (julio, 2011). Positional accuracy of assisted GPS data from high-sensitivity GPS-enabled mobile phones. *The Journal of Navigation, 64*(3), 381-399.

APÉNDICES

Apéndice 1. **Diseño Interfaz de usuario pantalla 1 y 2**



Fuente: elaboración propia, empleando Adobe Photoshop.

Apéndice 2. **Diseño Interfaz de usuario pantalla 3 y 4**

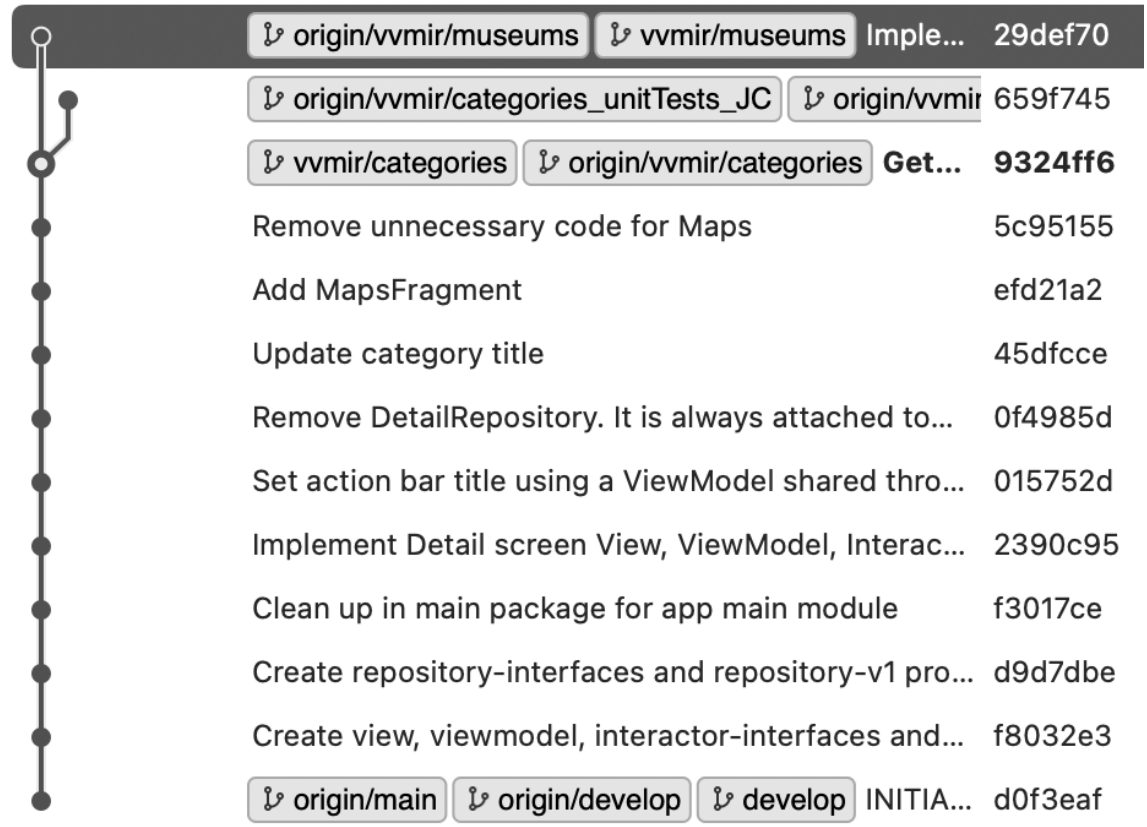
Fuente: elaboración propia, empleando Adobe Photoshop.

Apéndice 3. **Descripción de costos**

Descripción	Costo (Q.)
Tiempo de recurso humano:	
1. Arquitecto	Q. 7,000.00
2. Diseñador	Q. 2,000.00 Q. 4,000.00
3. Desarrollador (externo)	Q. 10,000.00
4. Desarrollador (investigador)	Q. 3,000.00
5. Aseguramiento de calidad	**Q. 26,000.00**
6. Analista	
Sub total:	
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 ($. 230.00)
Servicio de almacenamiento Firebase	Q. 0.00
SDK y herramientas Android	Q. 0.00
TOTAL:	**Q. 30,300.00**

Fuente: elaboración propia.

Apéndice 4. **Historial Git**



Fuente: elaboración propia, empleando SourceTree.

ANEXOS

Anexo 1. **Anotaciones Hilt y Dagger**

Annotation	Usage	Code Sample
@HiltAndroidApp	Kicks off Hilt code generation. Must annotate the Application class.	`@HiltAndroidApp` `class MyApplication : Application() { ... }`
@AndroidEntryPoint	Adds a DI container to the Android class annotated with it. This requires using Hilt's Gradle Plugin.	`@AndroidEntryPoint` `class MyActivity : AppCompatActivity() { ... }`

@Inject	Constructor Injection. Tells which constructor to use to provide instances and which dependencies the type has.	`class AnalyticsAdapter @Inject constructor(`
		`  private val service: AnalyticsService`
		`) { ... }`

	Field injection. Populates fields in @AndroidEntryPoint annotated classes.
	Fields cannot be private.
		`@AndroidEntryPoint`
		`class MyActivity : AppCompatActivity() {`
		`  @Inject lateinit var adapter: AnalyticsAdapter`
		`  ...`
		`}`

@HiltViewModel	Tells Hilt how to provide instances of an Architecture Component ViewModel.	`@HiltViewModel`
		`class MyViewModel @Inject constructor(`
		`  private val adapter: AnalyticsAdapter,`
		`  private val state: SavedStateHandle`
		`): ViewModel() { ... }`

@Module	Class in which you can add bindings for types that cannot be constructor injected.	`@InstallIn(SingletonComponent::class)`
		`@Module`
		`class AnalyticsModule { ... }`

@InstallIn	Indicates in which Hilt-generated DI containers (`SingletonComponent` in the code) module bindings must be available.	`@InstallIn(SingletonComponent::class)`
		`@Module`
		`class AnalyticsModule { ... }`

@Provides	Adds a binding for a type that cannot be constructor injected:
	- Return type is the binding type.
	- Parameters are dependencies.
	- Every time an instance is needed, the function body is executed if the type is not scoped.
		`@InstallIn(SingletonComponent::class)`
		`@Module`
		`class AnalyticsModule {`
		`  @Provides`
		`  fun providesAnalyticsService(`
		`  converterFactory: GsonConverterFactory`
		`  ): AnalyticsService {`
		`  return Retrofit.Builder()`
		`  .baseUrl("https://example.com")`
		`  .addConverterFactory(converterFactory)`
		`  .build()`
		`  .create(AnalyticsService::class.java)`
		`  }`
		`}`

@Binds	Shorthand for binding an interface type:
	- Methods must be in a module.
	- @Binds annotated methods must be abstract.
	- Return type is the binding type.
	- Parameter is the implementation type.
		`@InstallIn(SingletonComponent::class)`
		`@Module`
		`abstract class AnalyticsModule {`
		`  @Binds`
		`  abstract fun bindsAnalyticsService(`
		`  analyticsServiceImpl: AnalyticsServiceImpl`
		`  ): AnalyticsService`
		`}`

Scope Annotations:
@Singleton
@ActivityScoped
...
	Scoping object to a container.
	The same instance of a type will be provided by a container when using that type as a dependency, for field injection, or when needed by containers below in the hierarchy.
		`@Singleton`
		`class AnalyticsAdapter @Inject constructor(`
		`  private val service: AnalyticsService`
		`) { ... }`

Qualifiers for predefined Bindings:
@ApplicationContext
@ActivityContext
	Predefined bindings you can use as dependencies in the corresponding container.
	These are qualifier annotations.
		`@Singleton`
		`class AnalyticsAdapter @Inject constructor(`
		`  @ApplicationContext val context: Context`
		`  private val service: AnalyticsService`
		`) { ... }`

Fuente: Android (2020). *Hilt and Dagger annotations cheat sheet.*