TecnoLógicas
ISSN: 0123-7799
ISSN: 2256-5337
tecnologicas@itm.edu.co
Instituto Tecnológico Metropolitano
Colombia

Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android

Arias-Orezano, José Francisco; Reyna-Barreto, Benjamín David; Mamani-Apaza, Guillermo
Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android
TecnoLógicas, vol. 24, núm. 52, e2104, 2021
Instituto Tecnológico Metropolitano, Colombia
Disponible en: https://www.redalyc.org/articulo.oa?id=344268257016
DOI: https://doi.org/10.22430/22565337.2104

Artículos de investigación

Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android

Impact of Clean Architecture and ISO/IEC 25010 on the Maintainability of Android Applications

José Francisco Arias-Orezano
Universidad Peruana Unión, Perú
josearias@upeu.edu.pe
https://orcid.org/0000-0001-6265-4569

Benjamín David Reyna-Barreto
Universidad Peruana Unión, Perú
reyna_b@upeu.edu.pe
https://orcid.org/0000-0002-4359-5732

Guillermo Mamani-Apaza
Universidad Peruana Unión, Perú
guillepiter@upeu.edu.pe
https://orcid.org/0000-0002-2713-8769

DOI: https://doi.org/10.22430/22565337.2104
Redalyc: https://www.redalyc.org/articulo.oa?id=344268257016

Recepción: 10 Junio 2021
Aprobación: 11 Noviembre 2021
Publicación: 17 Diciembre 2021

Resumen:

La constante actualización de los aplicativos móviles está relacionada con el desarrollo continuo que demandan las necesidades del usuario, la tecnología y, sobre todo, los nuevos dispositivos. En efecto, esta ininterrumpida evolución, y la complejidad misma del aplicativo, hace que su mantenimiento no garantice la estabilidad cuando se agregan nuevas funcionalidades o se actualicen las versiones del sistema operativo. El objetivo de este estudio fue establecer el impacto de la implementación de arquitectura limpia y de la norma ISO/IEC 25010 en la mantenibilidad del aplicativo móvil Educar Teacher. El diseño de la investigación fue *ex post facto* cuasi experimental de corte transversal, considerando los aplicativo Educar Teacher y CRM Distribución como grupo experimental y de control, respectivamente, donde se evaluó y se comparó la mantenibilidad de ambos, considerando como unidad de análisis los paquetes, clases y líneas de código. La variable independiente fue arquitectura limpia y norma ISO/IEC 25010, y la dependiente fue mantenibilidad, la cual se trabajó con los criterios de analizabilidad, estabilidad, testeabilidad y cambiabilidad. La muestra fue censal y estuvo conformada por 693 paquetes, 2037 clases y 168 217 líneas de código del aplicativo Educar Teacher. De acuerdo con los resultados, se concluye que al desarrollar con arquitectura limpia y norma ISO/IEC 25010, el aplicativo Educar Teacher logra una repercusión positiva en la mantenibilidad basado en los criterios de analizabilidad, estabilidad, testeabilidad y cambiabilidad de 7 %, 56 %, 0.7 %, 0.9 %, respectivamente.

Palabras clave: Aplicaciones móviles, Android, Arquitectura de software, Arquitectura limpia, Calidad de software.

Abstract:

The constant evolution of mobile applications is related to the continuous development demanded by user needs, technology and, especially, new devices. This continuous evolution and the complexity of the application itself, means that its maintenance does not guarantee stability when new functionalities are added, or versions of the operating system are updated. The aim of this study was to establish the impact of the implementation of Clean Architecture & ISO/IEC 25010 on the maintainability of the Educar Teacher mobile application (www.icrmedu.com). The research design was quasi-experimental, cross-sectional, considering the Educar Teacher and CRM Distribution applications as experimental and control groups, respectively, where the maintainability of both was evaluated and compared, considering the packages, classes, and lines of code as the unit of analysis. The independent variable was Clean Architecture & ISO/IEC 25010, and the dependent variable was maintainability, which was worked with the criteria of analyzability, stability, testability, and changeability. The sample was census-based and consisted of 693 packages, 2.037 classes and 168.217 lines of code from the Educar Teacher application. According to the results, it is concluded that by developing

with Clean Architecture & ISO/IEC 25010, the Educar Teacher application achieves a positive impact on maintainability based on the analyzability, stability, testability, and changeability criteria of 7 %, 56 %, 0.7 % and 0.9 %, respectively.
Keywords: Mobile applications, Android, Software architecture, Clean architecture, Software quality.

1. INTRODUCCIÓN

En la actualidad el interés por desarrollar aplicaciones móviles ha aumentado exponencialmente por causa de la popularidad de los teléfonos inteligentes. En el 2020, los usuarios de Google Play descargaron 108.5 mil millones de aplicaciones móviles, frente a 76 mil millones del 2018 [1]. En abril del 2021, la cantidad de aplicaciones que alberga Google Play en su catálogo aumentó a 3.48 millones de aplicaciones móviles, lo que la convierte en la tienda con mayor cantidad de aplicaciones móviles disponibles [2]; sin embargo, el 14 % de todas las aplicaciones Android son de baja calidad de *software* [3].

El aumento de aplicaciones Android que son de baja calidad está estrechamente relacionado con una baja mantenibilidad, esto es debido, en primer lugar, al desarrollo de aplicaciones sin una arquitectura de *software* adecuada; en segundo lugar, por la necesidad de subir actualizaciones constantes causados por la exigencia de usuarios de nuevas funcionalidades, por los avances tecnológicos, por las nuevas políticas de privacidad de Google Play y por los errores en las nuevas funcionalidades [4]. Finalmente, la baja mantenibilidad tiene su efecto en el código de aplicación, efectos como la alta complejidad, la baja cohesión, el alto acoplamiento, la poca comprensibilidad, la poca reutilización de métodos, el gran tamaño de las clases y el poco uso del polimorfismo, la variabilidad y el encapsulamiento [5].

La mantenibilidad es una característica más importante de la calidad interna del *software* descrita en el modelo Calidad de Software de la ISO/IEC 25010, en donde describe a la característica de la mantenibilidad como la capacidad de una aplicación de *software* a modificarse. La mantenibilidad ha englobado gran parte del ciclo de vida del *software* y está formada por los siguientes criterios: analizabilidad, que es la facilidad de buscar deficiencias e identificar los componentes; cambiabilidad, que permite hacer modificaciones; estabilidad, que cuenta con la capacidad de evitar efectos inesperados después de alguna modificación; capacidad de ser probado o testeabilidad, la cual valida los cambios; y por último, el cumplimiento de estándares, que fue eliminada en la actualización de la norma ISO/IEC 9126 a la nueva ISO/IEC 25010 [6].

Con el fin de medir la mantenibilidad, LogisCope Technology, empresa de aseguramiento de calidad de software, desarrolló una herramienta que permitió evaluar la característica de la mantenibilidad según cuatro criterios: analizabilidad, cambiabilidad, estabilidad y testeabilidad, además de asociarlo a un grupo de métricas y a un umbral para las métricas (valor mínimo o valor máximo) [7], [8]. Asimismo, para poder disminuir los efectos de la baja mantenibilidad en las aplicaciones, se construyeron y modificaron arquitecturas de *software*, entre las que destaca arquitectura limpia, debido a que proporciona una forma para estructurar aplicaciones que soporten una evolución continua. Esta arquitectura fue propuesta en el 2012 por Robert Cecil Martin en su sub blog, en donde describe a *arquitectura limpia* como la unión de varias arquitecturas que persiguen el objetivo de la separación de preocupaciones mediante el uso de una capa de negocio y otra capa de interfaces [9].

En años recientes, la corporación BPQL (Business Prosperous Quality of Life) ha incursionado en el desarrollo de aplicaciones móviles para satisfacer a sus clientes. Sin embargo, tuvo malas experiencias con su aplicación CRM Distribución, debido a que se identificaron problemas en la fase de mejora y mantenimiento, además de producir malestar en sus clientes por fallos inesperados por los cambios realizados y por las demoras en las correcciones. Este resultado llevó a la corporación BPQL a querer desarrollar aplicaciones móviles que soportaran la evolución continua. En esta investigación, financiada por la corporación BPQL, se implementó arquitectura limpia y la ISO/IEC 25010 en el desarrollo del nuevo aplicativo, Educar Teacher, con el objetivo de demostrar que la implementación de arquitectura limpia y la ISO/IEC 25010 son un factor de mejora en la característica de la mantenibilidad.

2. ESTADO DEL ARTE

2.1 Mantenibilidad

En el 2012, Emanuel Irrazábal, en su tesis doctoral, construyó un entorno de la medición de calidad de *software* denominado KEMIS, solamente evaluando las características de mantenibilidad de ISO/IEC 25010. Su modelo de emisión consta de cuatro criterios: analizabilidad, cambiabilidad, estabilidad y capacidad de ser probado. Las herramientas que usó para evaluar los criterios fueron Java NCSS, PMD/CPD, CheckStyle, FindBugs, JDepend, CCCC, StyleCop, FxCop, Junit, CPPUnit, NUnit y EMMA [10].

Dos años después, en el 2014, Abdulrhman Albeladi y Rabe Abdalkareem evaluaron la capacidad de mantenimiento de dos aplicaciones de código abierto, MARF y GIPS, usando ISO/IEC 25010, más la herramienta LogisCope, que midió la mantenibilidad usando la fórmula Mantenibilidad = Analizabilidad + Cambiabilidad + Estabilidad + Capacidad de ser probado, en donde los resultados mostraron que la herramienta LogisCope es capaz de medir correctamente los *softwares* [11].

Ese mismo año, Ivano Malavolta y Roberto Verdecchia investigaron la evolución de los problemas de mantenibilidad en las aplicaciones Android mediante un estudio empírico a 434 repositorios de aplicaciones de código abierto que estuvieran publicadas en Google Play, descubriendo que, al pasar el tiempo, los problemas de la mantenibilidad crecían hasta un punto en el que se estabilizaban, pero nunca se resolvían [12].

Por su parte, en 2015, Bo Wang desarrolló una herramienta para evaluar la calidad de aplicaciones Android denominado MetricsReloaded, con el objetivo de ayudar a los desarrolladores a medir las métricas de complejidad e implementarlo en el mercado de Android Studio [13].

Ya para el 2017, Billy Susanto Panca y Sukrisno Mardiyanto evaluaron la capacidad de mantenimiento de tres aplicaciones Android m-Learning, m-Health y m-Survey con diferentes patrones de diseño. El objetivo de esta investigación es encontrar la combinación de patrones de diseño que repercutan en la mantenibilidad. Finalmente, concluyeron que el uso de patrones influye en la mantenibilidad excepto el patrón de estado [14].

De igual forma, en 2017, Saifan y Areej Al-Rabadi evaluaron la capacidad de mantenimiento a tres aplicaciones Android de código abierto: Klyph Facebook, Anki-Android-develop y Facebook SDK, con el objetivo de elegir qué métrica tiene mayor impacto usando el modelo de la ISO/IEC 25010, LogiScope y MetricsReloaded. Finalmente, concluyeron que Android tenía unas características especiales que no les permitió encontrar la métrica con mayor impacto [15].

2.2 Arquitectura limpia

A finales de 2017, debido al existo del libro "The Clean Architecture, guía para especialistas en la estructura y el diseño de software" [16], un equipo de Android y colaboradores propusieron una arquitectura influenciada en arquitectura limpia denominada Architecture Blueprints [beta] - MVP + Clean Architecture, que publicaron en el repositorio de GitHub [17].

En el 2017, Tung Bui Duy implementó arquitectura limpia a la aplicación Sunshine, de la empresa C63 Studio, con el objetivo de ver si la implementación reduce el esfuerzo en el desarrollo. Al terminar la implementación concluyó que se redujo el esfuerzo de desarrollo a la hora de testear y modificar la aplicación [18].

Al año siguiente, 2018, Montes Anccasi incorporó arquitectura limpia al desarrollo móvil de la empresa GMD, mediante la implementación y documentación de arquitectura limpia a la aplicación Android Banco de la Nación - Banca Móvil, concluyendo que la incorporación de arquitectura limpia mejorar el desarrollo de los aplicativos en todo el ciclo de vida de *software* [19].

Para el año 2019, Shady Boukhary y Eduardo Colmenares propusieron una arquitectura de *software* basada en arquitectura limpia para el desarrollo de aplicaciones Flutter debido al problema de administrar sus componentes y la mantenibilidad que sufrían los desarrolladores. Finalmente implementaron su arquitectura y concluyeron que la implementación de Clean Arquitectura mejora la mantenibilidad y la administración de componentes [20].

3. METODOLOGÍA

3.1 Tipo y diseño de la investigación

El tipo de investigación es prescriptiva, puesto que realiza una propuesta para mejorar la mantenibilidad, y cuantitativa porque mide la variable de estudio.

Por la naturaleza de la investigación, y de acuerdo con Hernández Sampieri, el diseño de investigación es *ex post facto* cuasi experimental de corte transversal, puesto que se hizo una sola evaluación de la mantenibilidad al final del desarrollo del aplicativo Educar Teacher, que es el grupo experimental y este se comparó con la evaluación del aplicativo CRM Distribución, que es el grupo de control. En la Figura 1 se presenta el esquema de la investigación [21].

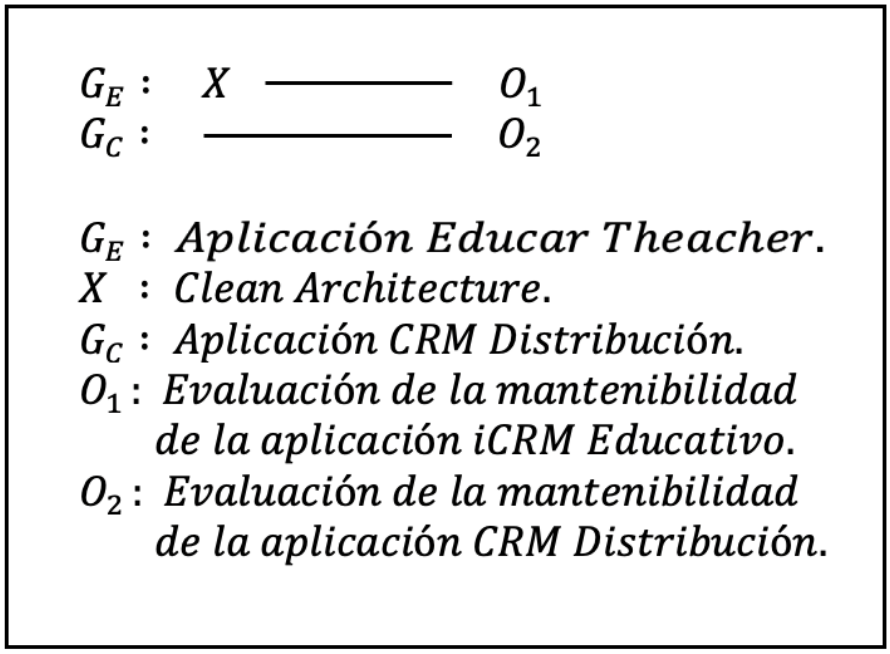

FIGURA 1.
Esquema del diseño de la investigación
Fuente: elaboración propia.

3.2 Construcción del aplicativo móvil Educar Teacher basada en arquitectura limpia

Para la construcción del aplicativo móvil en Android, esta investigación se basó en una implementación, publicada por Google en su repositorio de Github, denominado Architecture Blueprints [beta] - MVP + Clean Architecture.

Para poder construir la aplicación, se revisó la estructura de la nueva implementación de arquitectura limpia (ver Figura 2) y cómo se comunicaban sus componentes dentro y fuera de sus respectivas capas (ver

Figura 3). Donde la capa de presentación interactúa con la interfaz de usuario mediante el patrón de diseño MVP, la segunda capa es la del dominio que se encarga de contener la lógica de negocio y de almacenar mediante los casos de uso y por último la capa de datos.

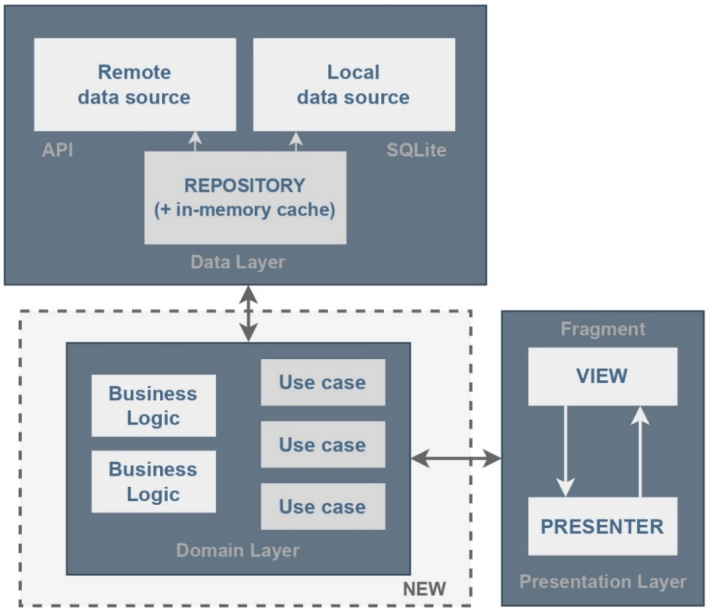

FIGURA 2.
Estructura de la implementación de arquitectura limpia aplicaciones Android
Fuente: [17].

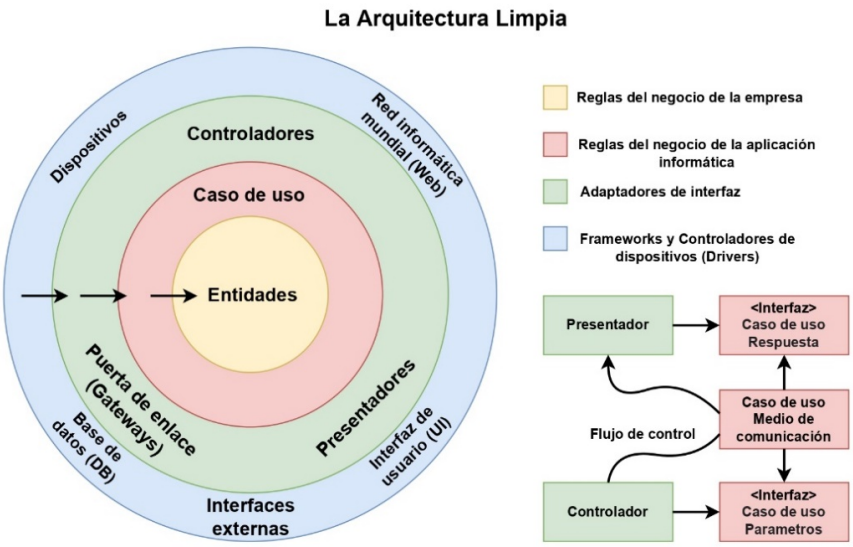

FIGURA 3.
La regla de la dependencia de arquitectura limpia
Fuente: [16].

Después de terminar de revisar la estructura de la nueva arquitectura (ver Figuras 4 y 5), se desarrolló la aplicación móvil Educar Teacher en el lenguaje Java para celulares Android desde 4.0 a superior. Los servicios Rest que usa esta aplicación están construidos en C#, además de estar publicados en un servicio de procesamiento Compute Engine de Google Cloud. A fin de almacenar la información de manera local, cuando el aplicativo funcionó en zonas sin internet se usó la base datos SQLite. Por otra parte, se implementó arquitectura limpia en el diseño de la aplicación, además de los cinco principios de diseño orientado a objetos de Robert Cecil Martin denominado SOLID, principios que ayudaron a comprender e implementar los patrones de diseño de una manera estratégica.

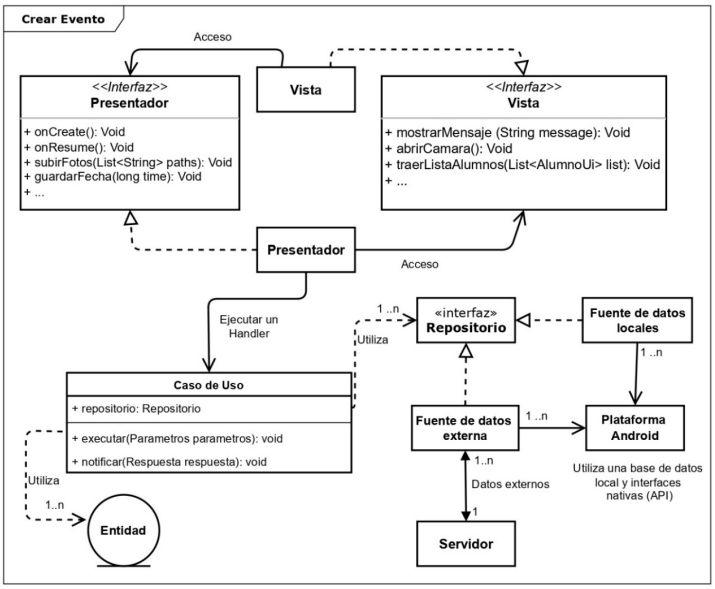

FIGURA 4.
Diagrama de flujo de uno de los módulos del aplicativo móvil Educar Teacher basado en arquitectura limpia
Fuente: elaboración propia.

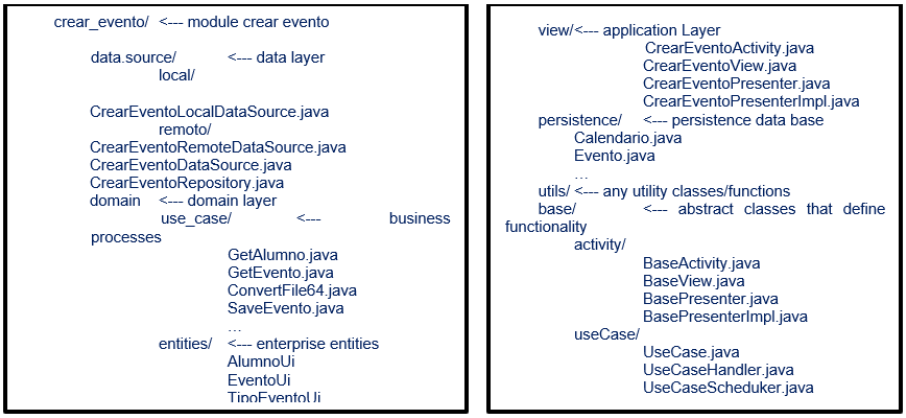

FIGURA 5.
Estructura de carpetas de uno de los módulos del aplicativo móvil del Educar Teacher basado en arquitectura limpia
Fuente: elaboración propia.

Finalmente, luego de terminar el desarrollo de aplicación móvil Educar Teacher, se describió las características de los dos grupos usados en esta investigación (ver Tabla 1).

TABLA 1
Características de las aplicaciones usadas en esta investigación

Aplicación	Grupo experimental	Grupo control
Nombre	Educar Teacher	CRM Distribución
Paquetes	693	19
Clases	2037	501
Líneas de código	168 217	32 530

Fuente: elaboración propia.

3.3. Evaluación de capacidad de mantenimiento

Esta evaluación se basó en las normas ISO/IEC 25010 y LogisCope para la medición de las métricas de los criterios de la mantenibilidad (ver Tablas 2 y 3). Se evaluó a la aplicación Educar Teacher (Grupo Experimental) y a la aplicación CRM Distribución (Grupo Control) (ver la Tabla 1) con las características de las dos aplicaciones en el entorno de desarrollo Android Studio, donde se implementó la extensión MetricsReloaded [13], [22] para obtener el resultado de cada métricas (ver Tabla 2) con los resultados. Finalmente, luego de obtener los resultados, se realizó el cálculo de las fórmulas de cada criterio de la mantenibilidad (ver Tabla 3).

TABLA 2
Resultado de las métricas de la mantenibilidad y el rango de valores aceptables

Métrica	Definición	Su efecto	Rango de valor aceptable		Valor del grupo	
			Min	Max	Control	Experimental
cl_wmc	Método ponderado por clase, mide la complejidad del método.	Complejidad, comprensibilidad y cohesión	0	11	26.20	24.58
in_base	Número de clases de las que la clase hereda directamente o no.	Complejidad y comprensibilidad.	0	9	3.15	2.34
cu_edused	El número de clases directamente utilizadas por otra clase.	Acoplamiento, complejidad y reutilización de métodos.	0	6	13.37	16.28
cl_cmof	La relación entre el número de líneas de comentarios y el número total de líneas del código.	El tamaño de la clase.	0 %	100 %	9.83 %	6.06 %
cl_stat	Número de declaraciones ejecutables de la clase.	El tamaño de la clase.	0	7	19.50	20.62
cl_data	El número total de atributos de la clase.	Complejidad y tamaño de la clase.	0	25	11.61	11.81
cl_func	El número total de funciones en la clase.	Complejidad y cohesión.	0	9	9.47	8.53
cl_func_publ	El número total de funciones públicas en la clase.	Acoplamiento y polimorfismo.	0	7	2.29	5.59
cu_edusers	El número de clases que utilizan directamente esta clase.	Acoplamiento, variabilidad y reutilización de métodos.	0	3	5.57	5.96
in_noc	El número de clases dentro de otra clase.	Reutilización de métodos y pruebas.	0	4	0.02	0.13
cl_data_publ	El número total de atributos públicos en la clase.	Encapsulación (ocultación de información)	0	7	1.36	6.22

Fuente: elaboración propia.

TABLA 3.
Resultado de los criterios de la mantenibilidad en función de las métricas de la Tabla 2

Criterios de la mantenibilidad	Fórmula	Valor del Grupo	
		Control	Experimental
Analizabilidad	cl_wmc + cl_cmof + in_base + cu_edused	52.55	49.26
Cambiabilidad	cl_stat + cl_func + cl_data	40.58	40.96
Estabilidad	cl_data_publ + cu_edusers + in_noc + cl_func_publ	9.24	20.90
Testeabilidad	cl_wmc + cl_func + cu_cdused	49.04	49.39
	Total	151.41	158.51

Fuente: elaboración propia.

4. RESULTADOS Y DISCUSIÓN

4.1 Descripción de las características de mantenibilidad

Los resultados obtenidos están representados en el diagrama de Kiviat (ver Figura 6) donde están en función a las métricas establecidas en el criterio analizabilidad (ver Tabla 3).

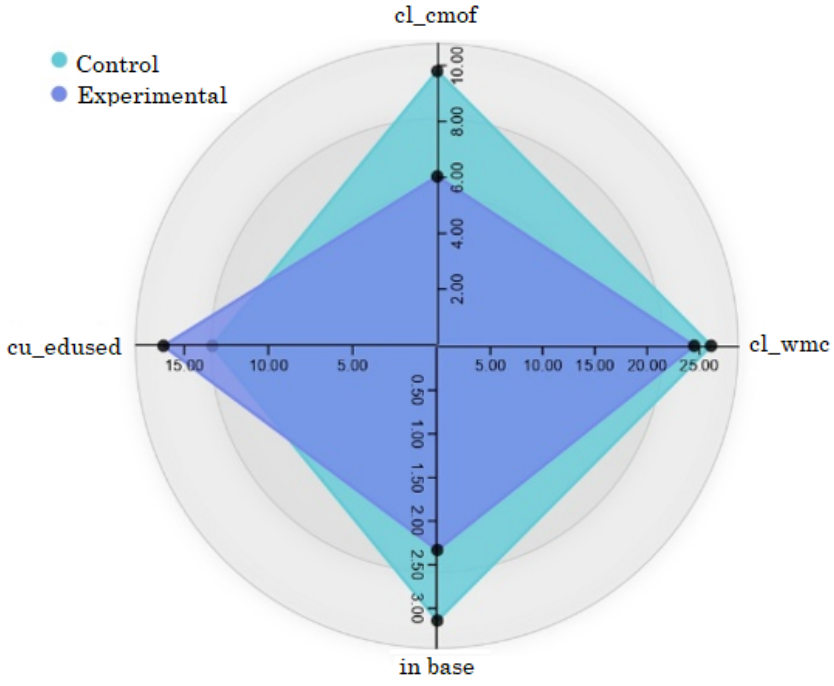

FIGURA 6.
Diagrama de Kiviat de las métricas de la analizabilidad del grupo control y grupo experimental
Fuente: elaboración propia.

Se observa en la Figura 6 que la métrica de (CL_WMC), en el aplicativo Educar Teacher desarrollado con arquitectura limpia, tiene un nivel de complejidad de 24.58, a diferencia del aplicativo CRM Distribución, que es de 26.2; de este resultado se deduce que al aplicar arquitectura limpia se reducen los niveles de complejidad frente a una aplicación que implementa arquitectura convencional MVC. De acuerdo con la

ISO/IEC 25010 y LogisCope, el rango aceptable del nivel de complejidad es 0 a 11, y en este estudio ambos grupos están fuera del rango, lo que indica que las aplicaciones son difíciles de entender y presentan una cohesión muy baja. A diferencia de la investigación de Abdulrhman Albeladi y Rabe Abdalkareem [11] que presentan un nivel de complejidad de 9 y 6, el cual están dentro del rango aceptable porque solo evaluaron un paquete de clases y no su totalidad. Sin embargo, en comparación de los resultados de la investigación de Ahmad A. Saifan y Areej Al-Rabadi [15], que es 16.397, 16.544 y 23.853, se asemejan con los resultados del nivel de complejidad del aplicativo Educar Teacher, puesto que ambos son aplicaciones Android.

Se observa en la métrica de la (CL_CMOF) de la Figura 6, que el aplicativo Educar Teacher desarrollado con arquitectura limpia tiene un grado de relación de 6.06 %. A diferencia del aplicativo CRM Distribución, que tiene un grado de relación de 9.83 %, de este resultado se deduce que la aplicación de arquitectura limpia sí mejora la métrica CL_CMOF, puesto que no tiene comentario de código que no se use y solo tiene comentarios de documentaciones frente a aplicaciones que implementen otra arquitectura MVC. De acuerdo con la ISO/IEC 25010 y LogisCope, el rango aceptable del grado relación es de 0 % a 100 %. En este estudio ambos grupos están dentro del rango aceptable. El caso es similar, tanto en la investigación de Ahmad A. Saifan y Areej Al-Rabadi [15], que tienen el grado de relación 12.83 %, 3.15 % y 20.95 %, como en la de Abdulrhman Albeladi y Rabe Abdalkareem [11], que presenta el grado de relación 89 % y 70 %, que no superan el rango aceptable. Sin embargo, ambos estudios tienen un grado mayor a los resultados de esta investigación, por lo cual se puede deducir que la arquitectura limpia tiene un mejor grado de relación que significa menos comentarios en las clases.

Los resultados obtenidos están representados en el diagrama de Kiviat (ver en Figura 7), en función a las métricas establecidas en el criterio cambiabilidad (ver Tabla 3).

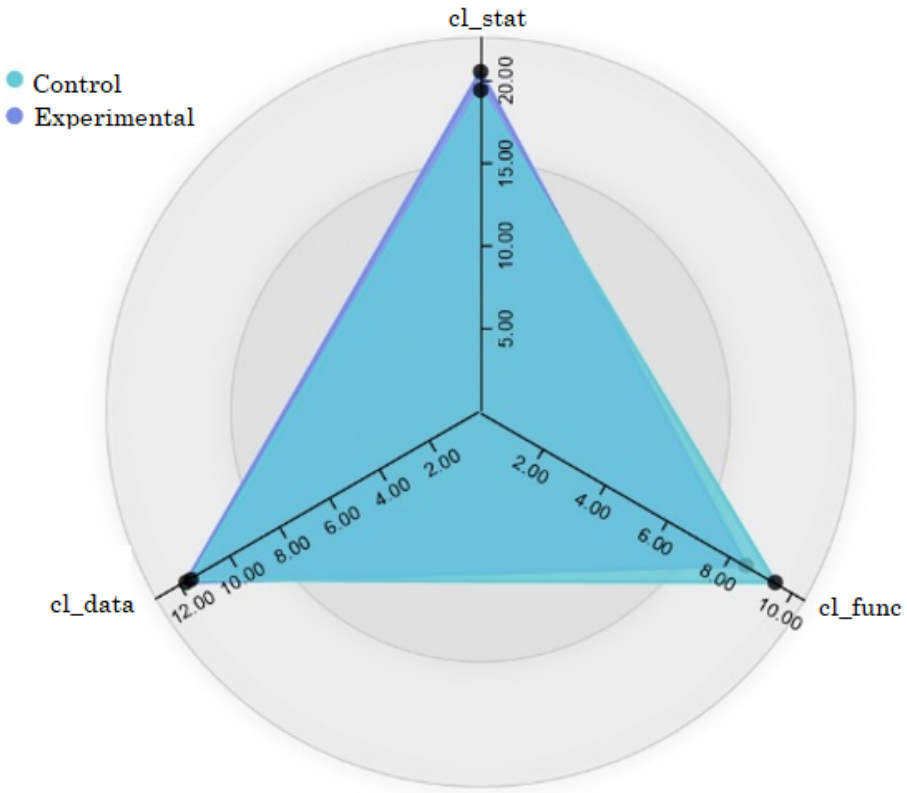

FIGURA 7.
Diagrama de Kiviat de las métricas de la cambiabilidad del grupo control y grupo experimental
Fuente: elaboración propia.

Se observa en la Figura 7 que la métrica de (CL_STAT) en el aplicativo Educar Teacher desarrollado con arquitectura limpia tiene un nivel de 20.62, a diferencia del aplicativo CRM Distribución, que tiene un nivel de 19.5; de este resultado se deduce que el aplicativo Educar Teacher tiene más variables que el aplicativo CRM Distribución. Esta diferencia se debe a la complejidad Educar Teacher que se presenta en la Tabla 1. De acuerdo con la ISO/IEC 25010 y LogisCope, el rango aceptable de CL_STAT es de 0 a 7. En efecto, el tamaño de las clases de las dos aplicaciones en estudio excede el rango aceptable. Por otra parte, en el estudio de Ahmad A. Saifan y Areej Al-Rabadi [15], el tamaño de CL_STAT de los tres aplicativos en estudio fueron de 100 700, 15 125 y 59 622, los cuales superan en gran medida el rango aceptable, a diferencia del aplicativo Educar Teacher.

Los resultados obtenidos están representados en el diagrama de Kiviat (ver Figura 8), en función de las métricas establecidas en el criterio estabilidad (ver Tabla 3).

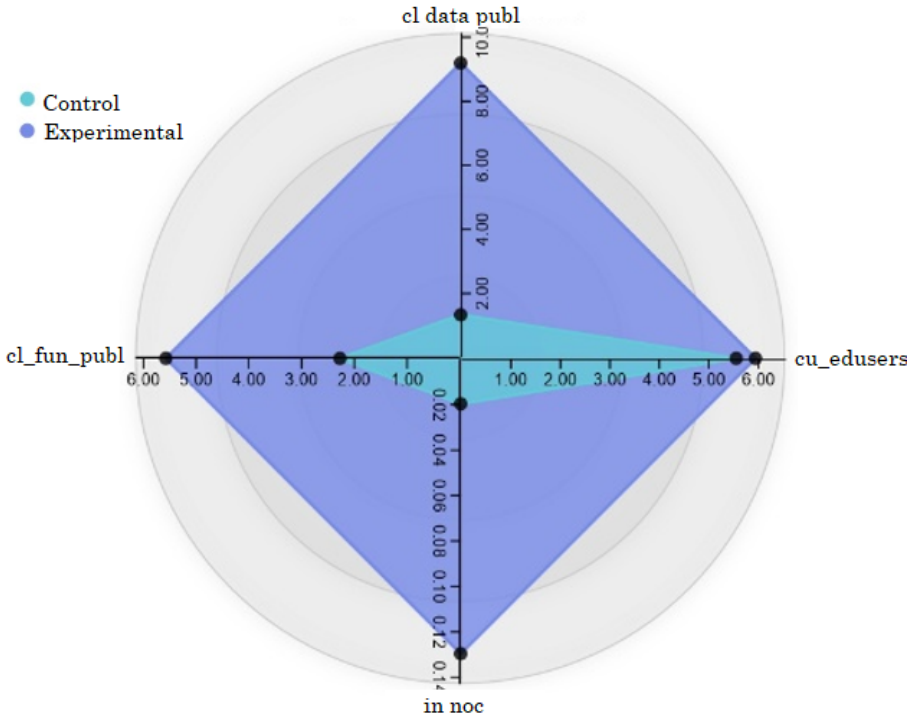

FIGURA 8.
Diagrama de Kiviat de las métricas de la estabilidad del grupo control y grupo experimental
Fuente: elaboración propia.

Se observa en la métrica (CL_DATA_PUBLIC) de la Figura 8 que el aplicativo Educar Teacher desarrollado con arquitectura limpia (grupo experimental) tiene un promedio de 6.22, a diferencia del aplicativo CRM Distribución (grupo control), que tiene un promedio de 1.36.

De este resultado se deduce que al aplicar arquitectura limpia sí mejora el CL_DATA_PUBLIC, puesto que se aproxima al límite superior del rango que es el ideal, lo que significa que tiene la cantidad de atributos que facilitan el mantenimiento, a diferencia de una arquitectura convencional MVC, que tiene un promedio muy bajo (un atributo por clase), lo que quiere decir que se crean muchas clases públicas con un solo atributo. De acuerdo con la ISO/IEC 25010 y LogisCope, el rango aceptable es de 0 a 7; y en este estudio ambos grupos están dentro del rango establecido, lo que indica que las aplicaciones tienen un nivel promedio de encapsulamiento.

Con respecto a la métrica del número total de funciones públicas en una clase pública (CL_FUNC_PUBLIC) que se muestra en la Figura 8, el aplicativo Educar Teacher desarrollado con arquitectura limpia tiene un promedio de 5.59, a diferencia del aplicativo CRM Distribución, que

tiene un promedio de 2.29. De este resultado se deduce que al aplicar arquitectura limpia sí mejora el CL_FUNC_PUBLIC, puesto que se aproxima al límite superior del rango que es el ideal, lo que significa que tiene la cantidad de funciones públicas que facilitan el mantenimiento, a diferencia de una arquitectura convencional MVC, que tiene un promedio muy bajo (dos funciones públicas por clase). De acuerdo con la ISO/IEC 25010 y LogisCope, el rango aceptable es 0 a 7; y en este estudio, en ambos grupos, están dentro del rango establecido, lo que indica que las aplicaciones tienen un nivel aceptable de polimorfismo y acoplamiento. Por otra parte, en el estudio de Ahmad A. Saifan y Areej Al-Rabadi [15], el tamaño de CL_FUNC_PUBLIC de los tres aplicativos en estudio fueron de 5493, 4528 y 5088, los cuales se aproximan a los resultados del aplicativo Educar Teacher, además de estar dentro del rango aceptado.

Los resultados obtenidos están representados en el diagrama de Kiviat (ver Figura 9), en función de las métricas establecidas en el criterio estabilidad (ver Tabla 3).

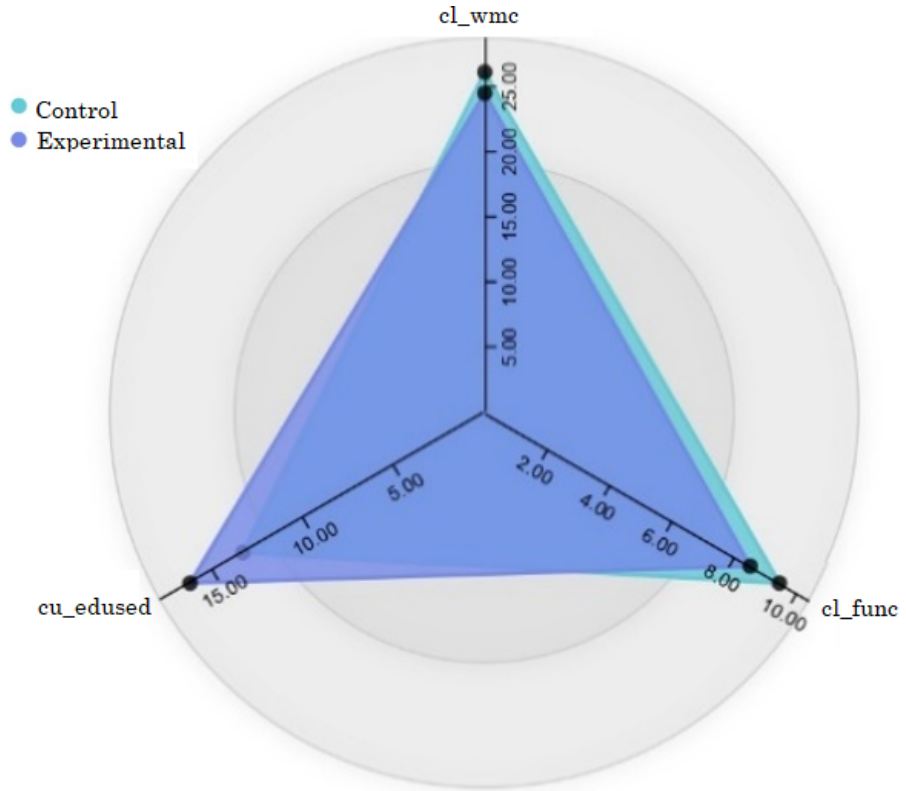

FIGURA 9.
Diagrama de Kiviat de las métricas de la estabilidad del Grupo Control y Grupo Experimental
Fuente: elaboración propia.

En la Figura 9 se observa que la métrica del número de clases directamente utilizadas por otra clase (CU_EDUSED) en el aplicativo Educar Teacher desarrollado con arquitectura limpia tiene un promedio de 16.28, a diferencia del aplicativo CRM Distribución, que tiene un promedio de 13.37. El aplicativo del Educar Teacher tiene un mayor CU_EDUSED debido al tamaño del proyecto que es cuatro veces más grande en número de clases frente a CRM Distribución. Por otra parte, el rango aceptable del CU_EDUSED es de 0 a 6 desacuerdo a la ISO/IEC 25010 y LogisCope. En efecto, las aplicaciones son complejas y presentan un alto acoplamiento puesto que sus CU_EDUSED están fuera del rango. Además, en el estudio de Abdulrhman Albeladi y Rabe Abdalkareem [11], el tamaño de NCDUOC de los dos aplicativos en estudio fueron de 9 y 6, el cual uno está dentro del rango aceptable y el otro está fuera, esto se debe a que solo evaluaron un paquete de clases y no todos los paquetes del proyecto.

Sin embargo, en comparación con los resultados de la investigación de Ahmad A. Saifan y Areej Al-Rabadi [15], con un tamaño de NCDUOC de 6811, 13 143 y 10 841 de sus tres aplicaciones en estudio, se asemejan con los resultados del aplicativo Educar Teacher, en el sentido que están fuera del rango y ambos son aplicaciones Android.

4.2. Descripción general de los niveles de mantenibilidad del aplicativo Educar Teacher

Los resultados obtenidos están representados en el diagrama de Kiviat (ver Figura 10) en función de las métricas establecidas en la característica de la estabilidad (ver Tabla 3).

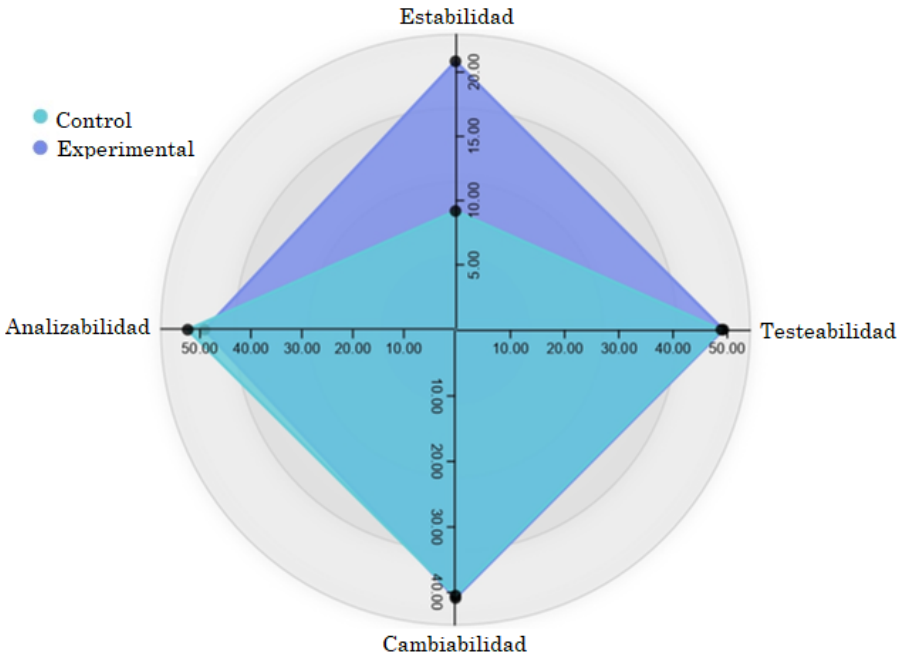

FIGURA 10.
Diagrama de Kiviat del resultado de características de la mantenibilidad del Grupo Control y Grupo Experimental
Fuente: elaboración propia.

Analizabilidad: en la Figura 10 se observa que al usar arquitectura limpia e ISO/IEC 25010 en el aplicativo Educar Teacher, se obtuvo una analizabilidad de 49.26, y del aplicativo CRM Distribución, que fue de 52.55, logrando reducir en 3.29, que representa un 7 % de mejora.

Estabilidad: en la Figura 10 también se observa que al usar arquitectura limpia e ISO/IEC 25010 en el desarrollo del aplicativo Educar Teacher, el promedio del nivel de estabilidad fue de 20.9 y 9.24 del aplicativo CRM Distribución; de aquí se deduce que hay una diferencia de 11.66 en la estabilidad del aplicativo cuando se desarrolla con arquitectura limpia, el cual se traduce en una mejora de 56 % en la estabilidad.

Testeabilidad: en la Figura 10 se observa que al desarrollar con arquitectura limpia e ISO/IEC 25010 en el aplicativo Educar Teacher, el promedio del nivel de testeabilidad es de 49.39 y 49.04 del aplicativo de CRM Distribución. Se observa que existe una mínima diferencia de 0.35, la cual se traduce en una mejora de solo del 0.7 % en el nivel de testeabilidad.

Cambiabilidad: finalmente en la Figura 10 se observa que al desarrollar con arquitectura limpia e ISO/IEC 25010 en el aplicativo Educar Teacher, el promedio del nivel la cambiabilidad es de 40.96 y 40.58 del aplicativo CRM Distribución. De este resultado se observa que hay una mínima diferencia de 0.38, el cual se traduce en una mejora mínima de 0.9 %.

5. CONCLUSIONES

De los resultados se concluye que al desarrollar con arquitectura limpia e ISO/IEC 25010, el aplicativo Educar Teacher tiene una repercusión del 7 % en el criterio de analizabilidad del aplicativo Android, logrando reducir en 3.29 frente a la analizabilidad del aplicativo CRM Distribución.

Con respecto a la estabilidad, de acuerdo con los resultados, se concluye que al desarrollar con arquitectura limpia e ISO/IEC 25010 el aplicativo de Educar Teacher tiene una repercusión del 56 % en el criterio de estabilidad, logrando así una diferencia de 11.66 frente al aplicativo del CRM Distribución.

De manera similar, y de acuerdo con los resultados, se concluye que al desarrollar con arquitectura limpia e ISO/IEC 25010, el aplicativo Educar Teacher tiene una repercusión el criterio de testeabilidad de 0.7 %, logrando así mejorar en 0.35 el nivel de promedio de la testeabilidad.

Finalmente, de acuerdo con los resultados se concluye que al desarrollar con arquitectura limpia e ISO/IEC 25010, el aplicativo Educar Teacher tiene una repercusión del 0.9 % en el criterio de cambiabilidad, logrando una mejora mínima de 0.38 el nivel del promedio.

Por lo tanto, se concluye que al desarrollar con arquitectura limpia e ISO/IEC 25010, el aplicativo Educar Teacher logra una repercusión positiva en la mantenibilidad con base a los criterios de analizabilidad, estabilidad, testeabilidad y cambiabilidad de 7 %, 56 %, 0.7 %, 0.9 %, respectivamente.

Agradecimientos

Como primer autor, me dieron el privilegio de expresar mis agradecimientos, los cuales van dirigidos, en primer lugar, a Dios, por la vida y salud que nos ha dado a mi familia y a mí en estos tiempos tan difíciles. Asimismo, quiero agradecer a mis padres porque fueron la inspiración en mi vida y la fuerza para lograr terminar este estudio. Por otro lado, agradezco a la Corporación BPQL, por el financiamiento y por aceptar el uso del proyecto de desarrollo Educar Teacher. Finalmente, mi agradecimiento a Mg. Benjamín David Reyna Barreto y al Dr. Guillermo Mamani Apaza, que fueron mis asesores asignados por mi alma máter, la Universidad Peruana Unión (Lima).

REFERENCIAS

[1] Satista, "Annual number of app downloads from the Google Play Store worldwide from 2016 to 2020,", 2021. https://www.statista.com/statistics/734332/google-play-app-installs-per-year/

[2] Statista, "Number of apps available in leading app stores as of 1st qurter 2021," 2021. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[3] AppBrain, "Number of Android applications on the Google Play," 2021. https://www.appbrain.com/stats/number-of-android-apps

[4] G. Hecht; O. Benomar; R. Rouvoy; N. Moha; L. Duchien, "Tracking the software quality of android applications along their evolution (T)," in *Proc. - 2015 30th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Lincoln. 2016, pp. 236-247. https://doi.org/10.1109/ASE.2015.46

[5] K. K. Aggarwal; Y. Singh; A. Kaur; R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," *Softw. Process Improv. Pract*., vol. 14, no. 1, pp. 39-62, Aug. 2008. https://doi.org/10.1002/spip.389

[6] G. M. Medina Sanes, "Definición y evaluación de un modelo de calidad en uso para un portal de bolsa de trabajo utilizando la norma ISO/IEC 25000," Trabajo de grado, Pontificia Univ. Católica del Perú, Lima, 2014. http://hdl.handle.net/20.500.12404/5383

[7] M. A. Servello, "Logiscope and the software maintenance crisis," in *Proc. Conf. Softw. Maint*., San Diego, 1990. https://doi.org/10.1109/icsm.1990.131333

[8] J. Meekel; M. Viala, "Logiscope: a tool for maintenance," in *Proc. Conf. Softw. Maint*., Scottsdale. 1988, pp. 328-334. https://doi.org/10.1109/icsm.1988.10184

[9] R. C. Martin, "The Clean Code Blog," 2012. https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[10] E. Irrazábal, "Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software," Tesis de Docorado, Univ. Rey Juan Carlos, España. 2012. C:\Users\adolfoescobar\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\RZ6EHN2T\ URL

[11] A. Albeladi; R. Abdalkareem; F. Agwaeten; K. Altoum; Y. Bennis; Z. Nasereldine, "Toward Software Measurement and Quality Analysis of MARF and GIPSY Case Studies - a Team 13 SOEN6611-S14 Project Report," arXiv, 1407.0063, 2014. http://arxiv.org/abs/1407.0063

[12] I. Malavolta; R. Verdecchia; B. Filipovic; M. Bruntink; P. Lago, "How maintainability issues of android apps evolve," in *2018 IEEE Int. Conf. Softw. Maint. Evolution (ICSME)*, pp. 334-344, Madrid, 2018. https://doi.org/10.1109/ICSME.2018.00042

[13] Bo Wang, "An Android studio plugin for calculating and measuring code complexity metrics in Android applications," Tesis de Maestría, Towson University, 2015. https://mdsoar.org/bitstream/handle/11603/2459/TF2015Wang_Redacted.pdf?sequence=1&isAllowed=y

[14] B. S. Panca; S. Mardiyanto; B. Hendradjaya, "Evaluation of Software Design Pattern on Mobile Application Based Service Development Related to the Value of Maintainability and Modularity," en *2016 Int. Conf. Data Softw. Eng. ICoDSE*, Denpasar, 2017. https://doi.org/10.1109/ICODSE.2016.7936132

[15] A. A. Saifan; A. Al-Rabadi, "Evaluating maintainability of android applications," in *ICIT 2017 - 8th Int. Conf. Inf. Technol. Proc*., Amman, pp. 518-523. https://doi.org/10.1109/ICITECH.2017.8080052

[16] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, 1st ed. Prentice Hall, 2017.

[17] Github.inc, "Android Architecture Blueprints [beta] - MVP + Clean Architecture,". https://github.com/googlesamples/android-architecture/tree/todo-mvp-clean/

[18] B. D. Tung, "Reactive Programming and Clean Architecture in Android Development," (Bachelor of Engineerin), Helsinki Metropolia University of Applied Sciences, 2017. https://www.theseus.fi/handle/10024/126982

[19] J. A. Montes Anccasi, "Clean architecture para mejorar el desarrollo de aplicaciones móviles en la empresa GMD", Trabajo de grado, Univ. Nac. Mayor de San Marcos, 2018. https://hdl.handle.net/20.500.12672/10218

[20] S. Boukhary; E. Colmenares, "A clean approach to flutter development through the flutter clean architecture package," en *2019 Int. Conf. Comp. Sci. Comp. Intel.*, Las Vegas, pp. 1115-1120. https://doi.org/10.1109/CSCI49370.2019.00211

[21] R. Hernández Sampieri; C. Fernández Collado; M. del P. Baptista Lucio, *Metodología de la investigación*, McGraw-Hill, 2010.

[22] JetBrains, "Touring Plugins: Software Metrics," 2014. https://blog.jetbrains.com/idea/2014/09/touring-plugins-issue-1/

Notas

- **CONFLICTOS DE INTERÉS**
  Los autores José Francisco Arias Orezano, Benjamín David Reyna Barreto y Guillermo Mamani Apaza tenemos un propósito en común que es el de dar a conocer los resultados de este estudio para el beneficio de la comunidad científica, y en este contexto, declaramos que no existe ningún conflicto de intereses económicos, profesionales o personales que influenciaran en los resultados de esta investigación.
- **CONTRIBUCIÓN DE LOS AUTORES**
  José Francisco Arias-Orezano se encargó de la conceptualización, recursos, escritura, análisis y procesamiento de datos. Benjamín David Reyna-Barreto, de la revisión y supervisión en el marco de ingeniería de software. Guillermo Mamani-Apaza contribuyó en operacionalizar las variables de estudio, metodología y afinar la redacción.

Información adicional

Cómo citar / How to cite: J. F. Arias-Orezano; B. D. Reyna-Barreto; G. Mamani-Apaza, "Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android", *TecnoLógicas*, vol. 24, nro. 52, e2104, 2021. https://doi.org/10.22430/22565337.2104