DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS E INFORMÁTICA

TEMA: "ESTUDIO COMPARATIVO ENTRE PARADIGMAS DE PROGRAMACIÓN Y SU INFLUENCIA EN UN PROYECTO DE DESARROLLO DE SOFTWARE"

AUTORES:

BRBORICH HERRERA, WLADYMIR ALEXANDER

OSCULLO ÑACATO, BRYAN WLADIMIR

DIRECTOR: ING. LASCANO, JORGE EDISON, PHD

SANGOLQUÍ

2020

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, ***"Estudio comparativo entre paradigmas de programación y su influencia en un proyecto de desarrollo de software"*** fue realizado por los señores ***Brborich Herrera, Wladymir Alexander*** y ***Oscullo Ñacato, Bryan Wladimir*** el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razón por la cual me permito acreditar y autorizar para que lo sustenten públicamente.

Sangolquí, 22 de enero del 2020

Firma:

.....

Ing. Jorge Edison Lascano, Ph.D.

C. C. 1710893114

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORÍA DE RESPONSABILIDAD

Nosotros, ***Brborich Herrera, Wladymir Alexander*** y ***Oscullo Ñacato, Bryan Wladimir,*** declaramos que el contenido, ideas y criterios del trabajo de titulación: "***Estudio comparativo entre paradigmas de programación y su influencia en un proyecto de desarrollo de software"*** es de nuestra autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí, 22 de enero del 2020

Firma

Brborich Herrera, Wladymir Alexander
C. C. 1724561921

Firma

Oscullo Ñacato, Bryan Wladimir
C. C. 1717196693

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

AUTORIZACIÓN

Nosotros, ***Brborich Herrera, Wladymir Alexander*** y ***Oscullo Ñacato, Bryan Wladimir,*** autorizamos a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: "***Estudio comparativo entre paradigmas de programación y su influencia en un proyecto de desarrollo de software***" en el Repositorio Institucional, cuyo contenido, ideas y criterios son de nuestra responsabilidad.

Sangolquí, 22 de enero del 2020

Firma

.....

Brborich Herrera, Wladymir Alexander
C. C. 1724561921

Firma

.....

Oscullo Ñacato, Bryan Wladimir
C. C. 1717196693

DEDICATORIA

Para mi increíble hermana, Sara.

Brborich Herrera, Wladymir Alexander

Para mi madre Jakeline, abuela Teresa, tía Blanca, hermana Stephany y novia Nicole, que son demasiado pacientes, demasiado amables y maravillosas para expresarlo en palabras. Pero lo intento de todos modos.

Oscullo Ñacato, Bryan Wladimir

AGRADECIMIENTO

A mi padre Wladymir, por su infinita sabiduría, enseñarme a pensar, a resolver problemas y siempre seguir adelante. Sin su guía, este trabajo no habría sido posible.

A mi madre Nora, por su inmensa paciencia y siempre apoyarme para cumplir todas mis metas.

A mis compañeros con quienes forme el club de software, ha sido una experiencia increíble aprender y compartir con ustedes.

A mi compañero de tesis Bryan por todo su esfuerzo, y a nuestro director Edison por su guía. Formamos un equipo increíble.

A todos los estudiantes que participaron en el experimento, por su colaboración, enseñarles fue una experiencia muy divertida.

Brborich Herrera, Wladymir Alexander

Empecé mi aventura universitaria en 2014, no han faltado los buenos y malos momentos, y aquellos peores que incitaban a renunciar una y otra vez. Pero aquí estoy, a muy poco de conseguir satisfactoriamente mi título y concluir toda una etapa de vida. Por eso no debería ser ninguna sorpresa que muchas personas me brindaran su apoyo todo este tiempo. Será imposible mencionarlos a todos, sin embargo, hay algunos partícipes importantes a los que me gustaría dar sinceramente las gracias.

En primer lugar, están mis padres y hermanos. Dieron mucho de sí para formar la persona en la que me he convertido, no solo han brindado apoyo y consejos para ver mis metas cumplidas, sino que han tenido que renunciar a mi durante periodos de tiempo en que la universidad lo requería.

Gracias también a Ivana Nicole, ha estado presente desde casi el inicio de esta experiencia universitaria, sin todo su cariño y compañía no habría sido lo mismo, ha llenado de luz y felicidad mi ser, más que mi pareja es mi compañera de vida. A mi tía Blanquita y abuelita Teresa, dos seres tan puros e inocentes que me criaron, ahora que ya soy un adulto es mi turno de protegerlas.

También a todos mis amigos del Club de Software, compañeros y profesores de la universidad, por sus enseñanzas, apoyo y todo el trabajo conjunto que se ha realizado a través de los años para mi formación como profesional. Finalmente, un agradecimiento especial a mi compañero de tesis Wladymir, sin el cual todo este trabajo no habría sido posible, y a nuestro tutor Edison, por su paciencia y guía para llevar de mejor manera todo este proceso; los tres hemos formado un excelente equipo.

Oscullo Ñacato, Bryan Wladimir

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Actualmente, según estadísticas obtenidas por el Standish Group, alrededor del 70% de los proyectos de software fracasan; una de las principales causas de este fracaso, relacionada con la mantenibilidad del proyecto, es el constante cambio de requerimientos. Se han investigado posibles soluciones, por ejemplo, establecer a la programación orientada a objetos como estándar en la industria del software principalmente por su facilidad para representar el diseño de un sistema, generando suposiciones sobre los efectos positivos de este paradigma en la mantenibilidad. Por lo cual, esta investigación pretende comprobar si el paradigma de programación influye en un proyecto de software a través de su mantenibilidad. Se desarrolló un experimento con estudiantes de la Universidad de las Fuerzas Armadas ESPE, donde se compara la efectividad y la eficiencia del mantenimiento que realizan los desarrolladores de software usando el paradigma procedimental y el paradigma orientado a objetos. Se encontró diferencias estadísticamente significativas entre ambos paradigmas. Los estudiantes, al usar el paradigma procedimental, en promedio, fueron 10% más efectivos e implementaron una tarea más por hora que al usar el paradigma orientado a objetos. Nuestras conclusiones indican que el paradigma de programación posiblemente sí influye en el mantenimiento de un proyecto de software. Sin embargo, es necesario replicar este experimento en diferentes contextos y con una muestra más amplia de sujetos para poder generalizar los resultados.

PALABRAS CLAVE:

- **PARADIGMA DE PROGRAMACIÓN ORIENTADO A OBJETOS (POO)**
- **PARADIGMA DE PROGRAMACIÓN PROCEDIMENTAL (PROC)**
- **EFECTIVIDAD Y EFICIENCIA DE LA MANTENIBILIDAD**

ABSTRACT

According to Standish Group, around seventy percent of software projects fail due to several causes, among them, poor maintainability related to constant requirement changes. Object oriented programming is currently the de facto standard for software development due to the ease of project abstraction and representation, there are many statements in consideration to positive effects of this paradigm regarding to software maintainability. This research aims to test these assertions, and examine whether the election of a specific programming paradigm over another has or not impact in a software project through its maintainability. We conducted an experiment with the participation of students who belong to the Computer Science Department of Universidad de las Fuerzas Armadas-ESPE. We compared developers' maintainability effectiveness and efficiency when coding software applications using object oriented programming and procedural programming paradigms. After analyzing the experiment results, we found statistically significant differences in favor of procedural programming. Subjects were on average ten percent more effective and around one task per hour faster when performing maintenance on procedural code. This points out that, the selection of programming paradigm may influence ease of maintainability of a software project. Nevertheless, it is not possible to generalize the results without replication in broader contexts, and with a larger subject sample.

KEY WORDS:

- **OBJECT ORIENTED PROGRAMING PARADIGM (OOP)**
- **PROCEDURAL PROGRAMING PARADIGM (PROC)**
- **MAINTAINABILITY EFFECTIVENESS AND EFFICIENCY**

CAPÍTULO I

INTRODUCCIÓN

1.1 Antecedentes

La Ingeniería de Software nace en 1969, como respuesta a múltiples problemas que se encontraban durante el desarrollo de proyectos software a gran escala. Generalmente estos sistemas no cumplían con la adecuación funcional necesaria, en su mayoría costaban más de lo que se presupuestaba en un inicio y sobrepasaban los tiempos planificados (Sommerville, 2009).

Su mayor reto en la actualidad es la creciente demanda en el desarrollo de proyectos de software, lo cual ha resultado en la búsqueda de mayor calidad de estos productos (Wang, Ai, & Wang, 2017). Se han introducido normas como la ISO/IEC 25000 que pretende establecer normas para la calidad del software (NORMAS ISO 25000, n.d.), aun así los sistemas de software actuales, debido a su complejidad, siguen lejos de eliminar todos sus defectos, incluso después de un riguroso proceso de aseguramiento de la calidad (Wang et al., 2017).

En general se ha estimado que debido a esta creciente complejidad, alrededor del setenta por ciento de ellos fracasa (Kikuno, 2005; Lu, Liu, & Ye, 2010; Lynch, 2018b; Ruiz, 2004). Como podemos observar en las diferentes fuentes, esta figura es recurrente y no ha disminuido a través del tiempo.

Los proyectos de software fracasan debido a una variedad de causas clasificadas dentro de macro categorías como: personas, métodos, tareas y medio ambiente, que no necesariamente están aisladas, más bien que se relacionan entre sí, por lo cual detectar el porqué de un fracaso es una tarea compleja (Lehtinen, Mäntylä, Vanhanen, Itkonen, & Lassenius, 2014).

En ingeniería de software se ha realizado una clasificación de las posibles localizaciones u orígenes de errores, en las siguientes áreas: Estudio y Análisis, Diseño, Código, Otros, donde la más propensa a generarlos es la de estudio y análisis con un 56% (Ruiz, 2004).

El área de estudio y análisis comprende el diagnóstico inicial del proyecto, es decir todo lo relacionado con el levantamiento de requerimientos (Ruiz, 2004), particularmente en la clara expresión de las demandas del proyecto por parte del cliente (Lu et al., 2010), por lo cual muchos de los trabajos realizados se han enfocado en resolver los problemas que se producen durante este proceso. En concreto, una de las soluciones propuestas con más fuerza actualmente son las metodologías ágiles de desarrollo de software. Las cuales nacieron para realizar este tipo de proyectos con un enfoque diferente a las metodologías tradicionales, sin embargo, su éxito ha sido por lo general anecdótico (Chow & Cao, 2008).

Estas metodologías ágiles tratan de resolver los problemas relacionados con los requerimientos mediante un desarrollo incremental, donde los costos para generar cambios dentro del proyecto se ven reducidos (Sommerville, 2009). Sin embargo, solo se considera el procedimiento de obtención de nuevos requisitos y a la construcción rápida de nuevas funcionalidades como una solución.

Por otra parte, en el área de código según (Simmonds, 2012b) ha existido una evolución en los diferentes paradigmas de programación desde la programación en lenguaje de máquina hasta la programación orientada a aspectos, pasando por los paradigmas procedimental y orientada a objetos, cada una de las cuales han influenciado en la representación del diseño de un sistema (Frederick P. Brooks, 1986).

1.2 Planteamiento del problema

En la actualidad alrededor del setenta por ciento de los proyectos de software fracasan, dado que no se entregan a tiempo y/o sobrepasan el presupuesto estimado (Kikuno, 2005; Lu et al., 2010; Lynch, 2018b; Ruiz, 2004). De hecho esta es una cifra recurrente desde el año 2005 y en general mientras pasa el tiempo, no se tiene indicios de un desarrollo tecnológico o metodológico que pueda ser considerado como una solución completa a todos los problemas presentes (Frederick P. Brooks, 1986).

Uno de los mayores retos de la ingeniería de software se encuentran en la parte de diseño y análisis (Ruiz, 2004) donde una de las "balas de plata" propuestas como soluciones para los problemas de desarrollo ha sido el paradigma de programación orientada a objetos. Esto puede ayudar con la estructuración del diseño de un sistema, es decir, la definición de: arquitectura, módulos, interfaces y datos para que un sistema cumpla con un set de requerimientos (Whitten, Bentley, & Dittman, 2004), sin embargo no reduce la dificultad del diseño en si (Frederick P. Brooks, 1986).

Debido a esto, en las últimas décadas, se ha tomado como un estándar de facto al paradigma orientado a objetos para el software comercial (Maleki, Fu, Banotra, & Zong, 2017), debido a esto se han realizado una gran cantidad de suposiciones sobre sus efectos positivos frente a su contraparte procedimental, aunque realmente no se ha provisto evidencia empírica para respaldar estas afirmaciones (Ahmad & Talha, 2002). Existen propuestas que nos indican que la programación estructurada puede producir código fácil de entender. Resultando en la reducción de errores en el código, ya que permite a los desarrolladores seguir constantemente el flujo de un programa (Asagba & Ogheneovo, 2008; Tošić, Milena Vujošević, 2008)

Una posible causa del fracaso de los proyectos de software es la elección del paradigma de programación. Según (Henry & Humphrey, 2002) el paradigma de programación influye en la complejidad del código y la agilidad para implementar cambios. Por lo tanto, también afecta a la mantenibilidad del sistema que está directamente relacionada con la facilidad de implementación de nuevas funcionalidades (Banker, 2014), convirtiéndose en un factor crítico para el éxito o fracaso de un proyecto (Sommerville, 2009).

Es importante analizar la elección entre los paradigmas de programación orientado a objetos y procedimental de manera experimental en donde se evalúe los beneficios reales de cada uno.

1.3 Justificación

Como se ha discutido previamente, existen varias suposiciones sin fundamento científico dentro de la industria acerca de la efectividad de cada uno de los paradigmas de programación (Ahmad & Talha, 2002; Henry & Humphrey, 2002). Si nos referimos a su influencia en el éxito de un proyecto, (Frederick P. Brooks, 1986) propone a la programación orientada a objetos como un posible tratamiento para el fracaso de proyectos de software, facilitando la representación del diseño de los mismos.

Sin embargo 33 años después, en una época donde POO se ha convertido en un estándar de facto dentro del desarrollo de software (Maleki et al., 2017) y se han desarrollado metodologías que permiten en teoría disminuir el costo de los cambios dentro del software (Sommerville, 2009), la cifra de proyectos que fracasan sigue siendo elevada (70%) (Kikuno, 2005; Lu et al., 2010; Lynch, 2018b; Ruiz, 2004) y el éxito de dichas metodologías, anecdótico (Chow & Cao, 2008).

Nuestra propuesta para realizar un experimento, tiene como propósito estudiar cuáles pueden ser los beneficios de los paradigmas de programación orientado a objetos y procedimental dentro de la eficiencia y efectividad en la implementación de cambios, ya que estos factores afectan a la mantenibilidad y por ende a la rápida adición de nuevos requerimientos en el proyecto (Varuna Gupta & Singhal, 2015). Así, proveer información que ayude a mitigar los problemas relacionados con estas áreas.

Además, nuestro experimento pretende ser la base de futuras investigaciones que ayuden a profundizar en este tema, y establecer la verdadera influencia del paradigma de programación en el mantenimiento de un proyecto de software.

1.4 Objetivos

1.4.1 Objetivo general

Analizar la elección entre paradigmas de programación procedimental y orientado a objetos, mediante un experimento, con el propósito de verificar su influencia en la efectividad y eficiencia de implementación de cambios, con respecto al mantenimiento de un proyecto software en el contexto de estudiantes de pregrado en Ingeniería de Software y Sistemas de la Universidad de las Fuerzas Armadas ESPE.

1.4.2 Objetivo específico

I. Realizar una revisión preliminar de literatura sobre la relación entre los paradigmas de programación y el mantenimiento de un proyecto de software incluyendo trabajos de comparación previos, para establecer las bases del estudio a realizar.

II. Determinar la validez del experimento propuesto para que los resultados sean representativos dentro del contexto de la investigación.

III. Diseñar un experimento sobre el desarrollo de una aplicación web, para comparar la efectividad y la eficiencia en la ejecución de cambios entre los paradigmas de programación: orientado a objetos y procedimental

IV. Realizar el experimento diseñado, dentro de los parámetros establecidos para obtener datos sobre la ejecución del proyecto y experiencia de los sujetos de estudio (estudiantes de pregrado ESPE).

V. Evaluar e interpretar los resultados del experimento, comparando la efectividad y la eficiencia en la ejecución de cambios para obtener información sobre la mantenibilidad del código producido por los paradigmas estudiados.

1.5 Alcance

Para determinar de manera correcta el alcance de la investigación planteada, se proponen varias preguntas de investigación relacionadas a los objetivos específicos, como se muestra en la ***Tabla 1***.

Tabla 1
Preguntas de investigación

Objetivo específico	Preguntas de investigación
I. **Realizar una revisión preliminar de literatura sobre la relación entre los paradigmas de programación y el mantenimiento de un proyecto de software incluyendo trabajos de comparación previos, para establecer las bases del estudio a realizar.**	a) ¿Que comparativas se han realizado previamente entre paradigmas de programación? b) ¿Cuáles son las áreas donde existe mayor tendencia a errores en proyectos Software? c) ¿Afecta la elección del paradigma en el proceso de desarrollo de un proyecto?
II. **Determinar la validez del experimento propuesto para que los resultados sean representativos dentro del contexto de la investigación.**	a) ¿Un experimento sobre estudiantes de pregrado es la mejor opción para este tipo de investigación? b) ¿El experimento cumple con las características de validez?
III. **Diseñar un experimento sobre el desarrollo de una aplicación web, para comparar la efectividad y la eficiencia en la ejecución de cambios entre los paradigmas de programación: orientado a objetos y procedimental**	a) ¿Es posible aislar el experimento de manera que los resultados sean influenciados únicamente por la elección del paradigma de programación? b) ¿Las métricas establecidas dentro del experimento se pueden medir con exactitud?
IV. **Realizar el experimento diseñado, dentro de los parámetros establecidos para obtener datos sobre la ejecución del proyecto y experiencia de los sujetos de estudio (estudiantes de pregrado ESPE).**	a) ¿Cuáles son las principales amenazas a la validez que se pueden dar al realizar el experimento? b) ¿Es posible generar contingencias adecuadas para estas amenazas a la validez?
V. **Evaluar e interpretar los resultados del experimento, comparando** efectividad **y la eficiencia en la ejecución de cambios para obtener información sobre la mantenibilidad del código producido por los paradigmas estudiados.**	a) ¿Existe una relación entre el paradigma de programación y la eficiencia en el mantenimiento? b) ¿Existe una relación entre el paradigma de programación y la efectividad en el mantenimiento?

1.6 Definición de la investigación

De acuerdo con (Genero, Cruz-Lemus, & Piattini, 2014), un experimento en Ingeniería de Software es una investigación empírica, donde es necesario seguir un proceso experimental. El cual estará incorporado en la siguiente metodología de trabajo.

a) Definición del alcance

El alcance del experimento está definido por el objetivo de la presente investigación.

Definición de la funcionalidad del objeto experimental:

- Una aplicación web para registrar ordenes de pizzas.
- La aplicación permitirá construir pizzas a partir de la selección de varios ingredientes y un tamaño.
- La aplicación permitirá registrar la orden de una pizza.

b) Planificación

a. Contexto

i. Objetos experimentales

1. Backend (POO) en Java (Spring)
2. Backend (PROC) en Python (Flask)
3. Frontend

ii. Sujetos

Alumnos de la carrera Ingeniería de Software cursando la asignatura de Programación Orientada a Objetos y de la carrera de Ingeniería de Sistemas cursando las asignaturas de Programación Avanzada e Inglés Técnico, de la Universidad de las Fuerzas Armadas ESPE.

b. Selección de sujetos

La selección de los sujetos para cada parte del experimento se hará de manera aleatoria.

c. Variables

i. Variable independiente: Elección de paradigma de programación.

ii. Variables dependientes

1. Efectividad de la mantenibilidad ($Mefec$): % de tareas de mantenimiento realizadas de manera correcta
2. Eficiencia de la mantenibilidad ($Mefic$): velocidad en la ejecución de tareas de mantenimiento correctas.

d. Hipótesis

Se formularon las siguientes hipótesis nulas y alternativas, relacionadas con las tres variables dependientes propuestas, detalladas en la *Tabla 2*.

Tabla 2
Hipótesis estadísticas

Hipótesis Nula	Hipótesis alternativas
$\boldsymbol{H_{0,1}: Mefec_{poo} = Mefec_{proc}}$	$H_{1,1,1}: Mefec_{poo} \neq Mefec_{proc}$ $H_{1,1,2}: Mefec_{poo} > Mefec_{proc}$ $H_{1,1,3}: Mefec_{poo} < Mefec_{proc}$
$\boldsymbol{H_{0,2}: Mefic_{poo} = Mefic_{proc}}$	$H_{1,2,1}: Mefic_{poo} \neq Mefic_{proc}$ $H_{1,2,2}: Mefic_{poo} > Mefic_{proc}$ $H_{1,2,3}: Mefic_{poo} < Mefic_{proc}$

c) Operación

Donde se realizará la preparación del experimento, la cual consiste en una primera prueba con un grupo de control, para mejorar aspectos del diseño, posterior a eso se realizará la ejecución del experimento y la validación de los datos obtenidos

d) Análisis e interpretación

Donde se procesará la información en el paquete estadístico diferentes SPSS para obtener las conclusiones del experimento realizado y presentar los resultados.

1.7 Hipótesis de investigación

Se ha planteado la siguiente hipótesis de trabajo:

- **Hipótesis de investigación:** La elección del paradigma de programación (Procedimental u Orientado a Objetos) influye en la efectividad y eficiencia de la implementación de cambios dentro de un proyecto de software.

CAPÍTULO II

ESTADO DEL ARTE Y MARCO TEÓRICO

2.1 Estado del Arte

La revisión de literatura acerca de las principales causas de fracaso en proyectos de software, además de trabajos relacionados con las comparativas entre los paradigmas de desarrollo mencionados, se realizó basado en las guías de revisión sistemática de literatura propuestas por (B. Kitchenham & Charters, 2007). Se considera las actividades descritas a continuación:

2.1.1 Planteamiento de la revisión de literatura

En base al problema de investigación planteado fue posible tener un contexto para realizar la búsqueda de estudios científicos que aborden los temas de fracaso en un proyecto de software y la influencia de paradigmas de programación sobre estos, definir un objetivo de búsqueda y plantear preguntas de investigación.

2.1.2 Conformación del grupo de control y extracción de palabras claves relevantes para la investigación

Para la revisión de literatura se han definido aquellos artículos considerados relevantes para la investigación, aquellos que tomen aspectos generales acerca de los paradigmas de programación, su éxito en la construcción de proyectos software y las principales causas para su fracaso.

El grupo de control seleccionado que se indica en la ***Tabla 3*** se obtuvo luego de una serie de validaciones sobre los estudios propuestos, considerando la actualidad de los estudios tomando como referencia la última década.

Tabla 3
Grupo de Control

Título	Cita	Palabras clave
An Empirical Study on Code Comprehension: Data Context Interaction Compared to Classical Object Oriented	(Valdecantos, Tarrit, Mirakhorli, & Coplien, 2017)	Software, measurement, programming paradigms, documentation, java, measurement
Empirical Study on the Correlation between Software Structural Modifications and Its Fault-proneness	(Wang et al., 2017)	Software bugs, software complex networks, structural modifications, fault prediction
Analysis Failure Factors for Small & Medium Software Projects Based on PLS Method	(Lu et al., 2010)	Project success criteria, software projects, PLS

Luego de un análisis de los estudios que conforman el Grupo de Control se seleccionaron las palabras más relevantes: software projects, failure, causes, software bugs, project success criteria, measurement, programming paradigms, comparison.

2.1.3 Construcción de la cadena de búsqueda

Con las palabras clave que fueron obtenidas de los artículos científicos del grupo de control, y luego de varias pruebas con distintas combinaciones de las mismas, se conformó la siguiente cadena de búsqueda: (SOFTWARE PROJECTS) AND ((FAILURE) OR (SOFTWARE BUGS) OR (PROJECT SUCCESS CRITERIA) OR (CAUSES)) AND ((COMPARISON) OR (EXPERIMENTAL STUDY)) AND (PROGRAMMING PARADIGMS) AND

(MEASUREMENT) AND (SOFTWARE); enfocada a las principales causas del fracaso de los proyectos de software y a comparativas, estudios o experimentos entre paradigmas de programación.

2.1.4 Selección de estudios

La cadena de búsqueda construida en el punto anterior fue aplicada a la base digital IEEE Xplore, donde se obtuvieron alrededor de 2.000 artículos relacionados, sobre los cuales se aplicaron filtros para abstraer un número más manejable de documentos. Los filtros fueron:

- Estudios del tipo: technical report, conference paper & journal paper.
- Vigencia: estudios realizados a partir del año 2010. Año elegido debido al rápido avance de la tecnología, siendo necesaria la actualidad de los estudios.

En base a estos filtros, la base digital presentó alrededor de 75 artículos científicos clasificados como relevantes, mismos que fueron revisados por los investigadores para así elegir 6 estudios primarios que constituyen la base para realizar el estado del arte, estos estudios se muestran en la ***Tabla 4***.

Tabla 4
Tabla de estudios primarios seleccionados

Código	Título	Cita
EP1	Determining the Root Causes of Various Software Bugs Through Software Metrics	(V Gupta, Ganeshan, & Singhal, 2015)
EP2	Analysis Failure Factors for Small & Medium Software Projects Based on PLS Method	(Lu et al., 2010)
EP3	Empirical Study on the Correlation between Software Structural Modifications and Its Fault-proneness	(Wang et al., 2017)
EP4	An Empirical Study on Code Comprehension: Data Context Interaction Compared to Classical Object Oriented	(Valdecantos et al., 2017)
EP5	A Roadmap for Software Maintainability Measurement	(Saraiva, 2013)
EP6	Exploring the Educational Benefits of Introducing Aspect-Oriented Programming into a Programming Course	(Boticki, Katic, & Martin, 2013)

2.1.5 Elaboración del estado del arte

EP1 (V Gupta et al., 2015): Determining the Root Causes of Various Software Bugs Through Software Metrics

Gupta et al. abordan la problemática actual que existe en la Ingeniería de Software particularmente sobre el crecimiento de fallas encontradas en las aplicaciones, pese a que se han realizado innumerables estudios, manuales y metodologías para evitarlo.

En años pasados se pensaba que los problemas surgían de los malos procesos en diseño, básicamente un mal producto software era resultado de malos requisitos; sin embargo, estos problemas no han disminuido radicalmente luego de la evolución que ha tenido la Ingeniería de Software. Por ello, en este estudio sus investigadores han aislado métricas con factores que pueden contribuir a esta problemática.

Gupta et al. proporcionan las posibles razones de los errores más comunes y cómo evitarlos, sin garantizar que estos sean los únicos motivos o su origen no sea de otros ámbitos no considerados por el estudio, como por ejemplo la etapa de desarrollo.

EP2 (Lu et al., 2010): Analysis Failure Factors for Small & Medium Software Projects Based on PLS Method

Este grupo de investigadores indica que la tasa de éxito de proyectos de TI se mantiene por debajo del 30%. Proponen una metodología para abstraer cinco factores principales que contribuyen al fracaso de los productos software, algunos ya estudiados por la Ingeniería de Software como: los clientes, el gerente de proyecto, la metodología de desarrollo entre otros.

Consideran a la unidad y nivel de comunicación del equipo del proyecto como factor clave para su éxito, atribuyendo la responsabilidad al gerente del proyecto siendo quien toma las decisiones acerca de la metodología y tecnologías a utilizarse, si estas decisiones no van acorde a las capacidades del equipo, el fracaso es más probable.

EP3 (Wang et al., 2017): Empirical Study on the Correlation between Software Structural Modifications and Its Fault-proneness

Wang et al. mencionan como el software, pese a los cuidados con que se lo maneje (metodológicamente hablando), está lejos de no presentar defectos. Aunque se apliquen muchas técnicas rigurosas de garantía de calidad en todo el ciclo de vida de desarrollo de software (SDLC) se podrían inyectar nuevos errores involuntarios durante el proceso de desarrollo.

El estudio tuvo el objetivo de evaluar el grado de correlación entre los cambios estructurales de software y los errores, con un resultado favorable, indica que los errores están estrechamente relacionados con las modificaciones estructurales de las funciones del sistema. El estudio se centró en software estructurado pero los autores mencionan la posibilidad de que la conclusión se repita en más tipos de software.

EP4 (Valdecantos et al., 2017): An Empirical Study on Code Comprehension: Data Context Interaction Compared to Classical Object Oriented

Valdecantos et al. proponen que un lenguaje de programación, junto con el paradigma de programación que soporta, es un factor importante que afecta profundamente la forma en que los programadores comprenden el código; para lo cual llevan a cabo un experimento controlado para evaluar la comprensión del código escrito, enfrentando el paradigma de interacción de contexto de

datos (DCI), en lenguaje Trygve, con el paradigma orientado a objetos (POO), en lenguaje Java, comúnmente utilizado en la industria.

Algunas de las métricas utilizadas fueron: tiempo de permanencia en el código y comprensión para la localización, corrección y mejora de una referencia; enfocadas a la mantenibilidad de un producto software. Los sujetos experimentales para el estudio lo conformaron tanto profesionales como estudiantes con al menos un año de experiencia en programación orientada a objetos, el estudio consistió en que los participantes leyeran código de programas escritos en lenguaje java y trygve, así recopilando información sobre el tiempo que tardaban en entender el código.

El estudio determina que un software de gran escala orientado a objetos presenta un código poco comprensible por la complejidad y gran cantidad de referencias, afectando el proceso de mantenibilidad del mismo.

EP5 (Saraiva, 2013): A Roadmap for Software Maintainability Measurement

Saraiva menciona como el paradigma de programación orientado a objetos ha tomado gran protagonismo en la industria, las investigaciones se han dedicado a mejorar la calidad del software que se adhieren a este paradigma. Por lo cual a través de los años se ha propuesto un alto número de métricas para medir el mantenimiento de software orientado a objetos, sin existir una estandarización o un catálogo para resumir toda la información sobre estas métricas.

El estudio reunió alrededor de 570 métricas aplicables sobre la (Programación Orientada a Objetos) POO tanto en el mundo académico como en la industria, de los cuales se concluye con un

catálogo de las mejores 36 métricas aplicables a experimentos enfocados a la calidad del software orientado a objetos.

Últimamente muchos estudios se enfocan en la mantenibilidad del software, por lo que cada investigador conglomera las métricas que le ayuden a su investigación, llegando inclusive a proponer nuevas, generando una inconsistencia que causa conflicto en la industria del desarrollo en lo que verdaderamente se debe medir para asegurar la calidad del software.

EP6 (Boticki et al., 2013): Exploring the Educational Benefits of Introducing Aspect-Oriented Programming into a Programming Course

Boticki et al. elaboran un estudio experimental para medir la productividad de un grupo de estudiantes durante el aprendizaje y desarrollo de software enfocado a dos paradigmas de programación diferentes: programación orientada a aspectos y orientada a objetos. Los resultados del experimento fueron que los estudiantes tuvieron más dificultades con la (Programación Orientada a Objetos) POO, demorando más tiempo en comprender el programa de prueba y desarrollar las funcionalidades adicionales. Con respecto a la (Programación Orientada a Aspectos) AOP el tiempo de aprendizaje, comprensión y desarrollo fue considerablemente bajo concluyendo que este paradigma es un suplemento viable a la programación orientada a objetos.

Características del estado del arte

La tasa de fracaso de los proyectos software se ha mantenido en los últimos años sobre el 70%, la mayoría de estudios relacionan este problema con el proceso de planificación debido a que una gran cantidad de proyectos no superan esta etapa.

Con el nivel de madurez actual que tiene la Ingeniería de Software, culpar de los errores y el fracaso a los procesos de planificación y diseño no es del todo válido.

Algunos de los estudios mencionados proponen que el origen de estas falencias puede encontrarse en los procesos de desarrollo o de integración, no necesariamente en las metodologías utilizadas, sino que posiblemente en las herramientas o tecnologías empleadas. Por otro lado, muy pocos estudios se han realizado enfocados a los paradigmas de programación y su influencia en el éxito de un proyecto software, de los existentes casi su totalidad toma como referencia a la POO dada su importancia actual en la industria, sin embargo, cabe la posibilidad de que otros paradigmas se presentan como una mejor opción.

Estos estudios abarcan esta posibilidad de manera general y no profundizan para demostrar si el proceso de desarrollo y la elección del paradigma de programación influyen en el éxito, mantenibilidad o adecuación funcional de los proyectos software, abriendo la posibilidad para proponer un estudio que cubra el tema.

2.2 Marco Teórico

Para la presente investigación es necesario establecer una serie de conceptos y definiciones, que permitan tener un fundamento teórico que servirá como base para el desarrollo del experimento. Se tratan los temas referentes a las variables dependientes e independientes de la hipótesis de investigación, las cuales son: paradigmas de programación y mantenibilidad, relacionada a la calidad de un proyecto de software.

2.2.1 Paradigmas de Programación

En los primeros años de la industria de software, el trabajo de diseñar programas que resolvieran problemas era una labor complicada, por lo cual siempre ha existido la necesidad de generar nuevos mecanismos de abstracción, es decir, el proceso por el cual se intenta remover todo detalle innecesario respecto a un concepto (Cornell University, n.d.), con la finalidad de simplificar el diseño del software (Simmonds, 2012a).

Estos mecanismos de abstracción o escuelas de pensamiento se convirtieron en lo que actualmente conocemos como Paradigmas de Programación, los cuales son: "una forma aceptada para resolver un problema por medio de una computadora" (Instituto Tecnológico de Celaya, n.d.).

Por otra parte, en lo referente a la clasificación de los paradigmas de programación, se ha establecido que es bastante diversa dependiendo del autor y la época en la que esta haya sido realizada (Krishnamurthi & Fisler, 2019).

En base a la clasificación propuesta por (Laine, Shestakov, Litvinova, & Vuorimaa, 2011) en la ***Figura 1*** se establece claramente que existen dos escuelas de pensamiento principales de las cuales se derivan otros paradigmas de programación: Declarativa e Imperativa.

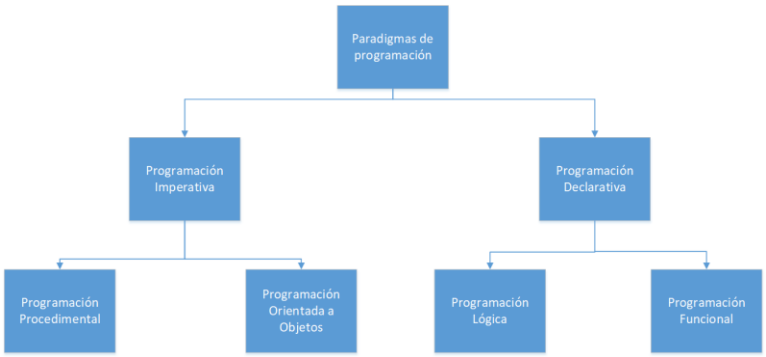

Figura 1. Jerarquía de paradigmas de programación

En base a estos macro conceptos han nacido diferentes enfoques especializados que influyen a los diferentes lenguajes de programación que se tiene en la actualidad (Krishnamurthi & Fisler, 2019). Según (Van Roy, 2009) cada lenguaje de programación entiende uno o más paradigmas de programación y los conceptos que estos conllevan, en consecuencia, actualmente existen más lenguajes que paradigmas. Siguiendo el modelo de la clasificación de los paradigmas de programación, es necesario definir las dos escuelas principales.

2.2.1.1 Programación Declarativa

Es un paradigma que presenta una aproximación a la resolución de problemas diferente a la programación imperativa (ver sección 2.2.1.2), dentro de este paradigma un programa es una teoría en un espacio lógico apropiado. En términos más simples, la programación declarativa provee un nivel de abstracción mayor, lo que quiere decir que los programadores se concentran en declarar lo que se debe calcular, sin necesariamente especificar como debe ser calculado (Torgersson, 1996).

2.2.1.2 Programación Imperativa

En contraposición a la programación declarativa, el paradigma imperativo presenta otra aproximación a la resolución de problemas, dando un menor nivel de abstracción y mayores libertades al programador.

Es considerado como una de las escuelas de pensamiento más importantes ya que, como se puede ver en la clasificación expuesta por (Laine et al., 2011), engloba a los estilos de programación Orientado a Objetos y Procedimental, siendo en general los paradigmas dominantes si se toma en cuenta los resultados de la encuesta anual para desarrolladores realizada por Stack Overflow,

(2019), expuesta en la ***Figura 2***, donde es evidente que lenguajes mayormente imperativos como Python, Java, C#, PHP, C++ y C tienen los mayores porcentajes de uso en la industria.

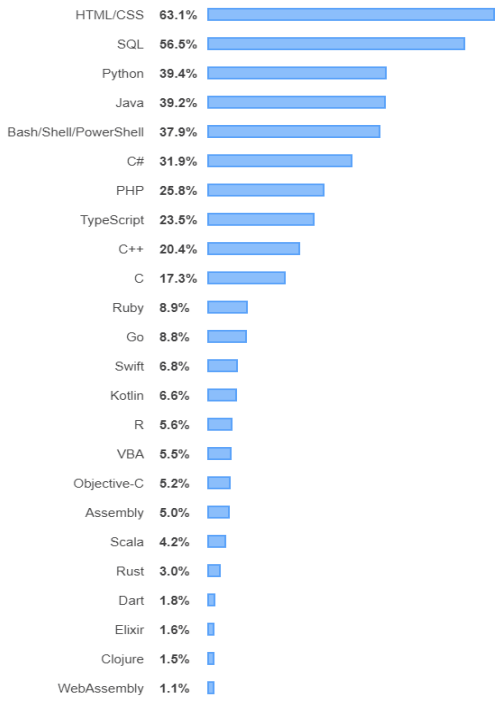

Figura 2. Lenguajes de programación más usados
Fuente: (Stack Overflow, 2019)

La programación imperativa toma como base a la arquitectura de Von Neumann, se refleja en los siguientes conceptos básicos sobre los cuales se han diseñado los lenguajes imperativos (Tošić, Milena Vujošević, 2008)

- Estados: representados por variables.
- Orden secuencial: refleja el uso de un CPU secuencial.
- Asignaciones: paso de información entre CPU y memoria.

Este paradigma utiliza los tres conceptos básicos para generar secuencias de direcciones que realizan una acción, es decir que cada instrucción en un lenguaje imperativo está diseñada para

mutar estados/variables, mediante asignaciones de manera ordenada. (Tošić, Milena Vujošević, 2008)

Además, se sostiene que una de las mayores ventajas de este paradigma es el hecho de su facilidad para representar conceptos cotidianos de una manera algorítmica, lo cual le ha dado el atractivo necesario para ser usado en todo tipo de proyectos de software (Frame & Coffey, 2014).

Finalmente, ya que esta escuela de pensamiento es bastante amplia y sus conceptos bastante generales, es necesario revisar a los principales paradigmas de programación basados en ella.

2.2.1.2.1 Programación Procedimental

El Paradigma Procedimental nace de la programación imperativa con el objetivo de disminuir la complejidad del código generado por paradigmas no estructurados, los cuales favorecían el uso de sentencias como ***GO TO***, que generaban una gran cantidad de saltos dentro del código, haciendo que operaciones de mantenimiento y actualización se volvieran complicadas (Tošić, Milena Vujošević, 2008).

La programación procedimental organiza el código basado en procedimientos, que no son más que funciones pequeñas y reutilizables, estas a su vez son organizadas en módulos los cuales agrupan funciones relacionadas con un mismo objetivo. Este paradigma se enfoca en construir un sistema basado desde el punto de vista del flujo de datos y cuáles son las operaciones o procedimientos que se deben realizar sobre estos (Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999).

(Asagba & Ogheneovo, 2008) establecen que el objetivo de programar organizando nuestro código en módulos de funciones es dividir la complejidad de un sistema grande en unidades más

pequeñas, reutilizando código, estableciendo divisiones claras entre secciones del programa y facilitar la adición de nuevas funcionalidades.

Dentro de este paradigma no existe una protección clara a la modificación de estados globales debido al manera de estados compartidos, por lo que inherentemente se permite la mutación de variables de forma insegura fuera del contexto de un módulo. Sin embargo, se puede evitar este problema manteniendo conceptos claros respecto al manejo de estados y eliminando el uso de variables globales.

Adicionalmente, para realizar un desarrollo procedimental existen varios lenguajes de programación que manejan este paradigma, sin embargo, para mantener un contexto similar en términos de antigüedad y características se ha tomado a Python como la herramienta a ser utilizada dentro del experimento.

Python

Es un lenguaje de programación multipropósito concebido al final de la década de 1980, la filosofía para su creación se centra en la legibilidad del código. Adicionalmente, según (Klein, 2016) Python posee ciertas características importantes a mencionar:

- Tipado dinámico: una variable en Python puede tomar valores de distinto tipo en distinto momento.
- Sintaxis limpia: al escribir código se favorece una sintaxis idiomática y legible, remplazando símbolos por palabras reservadas y usando abstracciones de mayor nivel.

- Resolución dinámica de nombres: Python asocia los nombres de funciones y variables en tiempo de ejecución
- Ejecución de código interactivo: permite la ejecución de comandos que devuelven un resultado inmediatamente, manteniendo las ejecuciones previas en memoria.

Python es multiparadigma, soportando diferentes estilos de creación de código, como: procedimental, estructurado, orientado a objetos y funcional; de manera que existe libertad al momento de diseñar un sistema. Sin embargo, Python es particularmente bueno implementando el paradigma procedimental (Mueller, 2015). Por su flexibilidad y sintaxis clara Python es el lenguaje con crecimiento más rápido en la industria (Stack Overflow, 2019).

Debido al enfoque web del experimento propuesto en el presente trabajo, es necesario elegir una tecnología que permita realizar este tipo de desarrollo sin modificar el estilo de codificación, es decir, un framework independiente como Flask.

Flask

Es un micro-framework para desarrollo de aplicaciones web con Python altamente flexible. Dentro del contexto de la documentación de Flask, el uso del prefijo micro, no hace referencia a la totalidad de funcionalidades del framework, más bien, se refiere a su simplicidad de uso y su filosofía “sin opinión”, es decir que no toma decisiones por el desarrollador, lo que en este contexto significa la libertad de estructurar el código usando cualquiera de los paradigmas de programación soportados por Python (Pallets Team, 2010).

Además, la extensibilidad de Flask viene dada por el soporte a diferentes dependencias o librerías desarrolladas por terceros, como el motor de plantillas o la capa de persistencia, lo que

permite a los desarrolladores adaptar el framework a las necesidades del proyecto (Pallets Team, 2010).

Finalmente, Flask no está diseñado para crear grandes aplicaciones monolíticas, de hecho, su objetivo es proveer una gran facilidad de uso para implementar de manera rápida y eficiente aplicaciones web tradicionales, esto juega de buena manera si consideramos que la tendencia de la industria tiende hacia realizar aplicaciones distribuidas basadas en componentes o servicios cada vez más pequeños (Di Francesco, Malavolta, & Lago, 2017).

2.2.1.2.2 Programación Orientada a Objetos

La Programación Orientada a Objetos es un tipo de programación imperativa, que nace como una posible respuesta a los diferentes problemas en la expresión del diseño de sistemas (Frederick P. Brooks, 1986), este paradigma tiene como concepto central al Objeto, el cual es una estructura de datos capaz de almacenar sub-rutinas o métodos además de variables o atributos. Un Objeto puede ser declarado mediante la creación de un patrón conocido como Clase, la cual esencialmente es un tipo de dato (Asagba & Ogheneovo, 2008).

Otra característica importante de este paradigma de programación es el hecho de que cada objeto que es creado encapsula su propio estado, es decir que los datos y procesos que se realizan en su interior no son visibles hacia otros procesos u otros objetos (Urdhwareshe, 2016).

La Programación Orientada a Objetos presenta diferentes propiedades las cuales definen en gran medida como se estructura un sistema y el proceso que se debe seguir para su desarrollo.

Abstracción

La abstracción de datos consiste en simplificar conceptos complejos del mundo real y convertirlos en clases que se puedan utilizar para resolver de manera adecuada un problema (Tošić, Milena Vujošević, 2008). Este apartado es uno de los principales fuertes de la POO, ya que en general muchos problemas pueden ser representados mediante relaciones entre objetos.

Encapsulamiento

El principio de encapsulamiento está directamente relacionado a la propiedad de los objetos de mantener su estado oculto o privado, forzando a un contexto externo a acceder a este mediante métodos propios del objeto (Asagba & Ogheneovo, 2008).

Herencia

La herencia es una propiedad asociada a las clases, en la cual una clase puede servir de modelo o padre para varias clases hijas las cuales heredaran todos los atributos y métodos dentro de la clase padre, esto ayuda a reutilizar código y agrupar objetos de acuerdo a propiedades generales (Asagba & Ogheneovo, 2008).

Por otra parte, la POO permite resolver el problema del estado compartido, mediante la división y encapsulamiento de cada estado dentro de un objeto, de esta manera se evitan ciertos problemas que surgen a raíz de la mutación de variables globales en la programación imperativa tradicional. Además, es importante notar que el concepto de POO es bastante extenso, de manera que para la presente investigación se ha elegido dar una descripción superficial de las definiciones que serán utilizadas para diseñar el código del experimento.

Finalmente, es necesario conocer las herramientas que permitirán un desarrollo Orientado a Objetos dentro del experimento: el lenguaje de programación Java y el framework Spring Boot.

Java

Es un lenguaje de programación de propósito general, multiplataforma, Orientado a Objetos (Oracle, 2018). Para comprender como Java se ha convertido en uno de los mayores exponentes de la POO es necesario revisar un poco de su historia.

A finales de la década de 1970 y comienzos de la década de 1980, se establece un nuevo concepto de programación, altamente influenciado por la aparición de C, y con esto un nuevo enfoque para la creación de lenguajes de programación basados en programación imperativa.

Esta vez motivado por la creciente complejidad del software, existe otro giro en el proceso de desarrollo de software. lo que llevo a la búsqueda de nuevos métodos de estructurar el código, la creación de C++ es uno de los resultados, este lenguaje heredaba bastante de su predecesor C, pero además incluía dentro de si el soporte para el paradigma orientado a objetos (Schildt, 2014).

Con estas bases nace Java en 1995, heredando varias características de C y C++ (como el estilo de sintaxis y la Orientación a Objetos), pero a su vez trayendo un concepto importante: la compatibilidad entre plataformas diferentes y entre otras cosas resolviendo los problemas de portabilidad y seguridad mediante el uso de bytecode no ejecutable, el cual es un set de instrucciones altamente optimizadas para ser ejecutadas en un ambiente especifico de java, la JVM (Java Virtual Machine) (Oracle, 2018).

Adicionalmente, debido a este diseño, Java mantiene una estrecha relación con la POO, siendo esta una parte integral del lenguaje. De hecho, esta relación es tan fuerte que cualquier programa desarrollado en Java es hasta algún punto Orientado a Objetos (Schildt, 2014).

Dentro del contexto del desarrollo web, existen diferentes frameworks y tecnologías que toman a Java como su principal lenguaje de programación, siendo una de estas JEE (Java Enterprise Edition), sin embargo, para el contexto del experimento se ha tomado una herramienta más simple y moderna como Spring Boot.

Spring Boot

Es un framework moderno para desarrollo de aplicaciones web basadas en Java, según (Pivotal Software, 2019) presenta varias características como:

- Bajo tiempo de instalación y desarrollo.
- Construcción de APIs REST de manera sencilla.
- Gran soporte para extensiones y herramientas para desarrolladores.

Spring es un framework que mantiene una fuerte opinión sobre cómo se debe desarrollar una aplicación, principalmente para facilitar al desarrollador el hecho de ponerse en marcha (Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, Michael Simons, Vedran Pavić, Jay Bryant, 2018).

2.2.2 Calidad de un Proyecto de Software

Es una extensa y compleja mezcla de factores, los mismos que varían de acuerdo al tipo de producto software o los requerimientos del cliente, los factores se pueden medir tanto directamente

como indirectamente. (Pressman, 2010) relaciona la calidad del software con tres puntos importantes: un proceso eficaz de software, un producto útil, agregar valor para el desarrollador y para sus stakeholders.

Un proceso eficaz de software constituye un apoyo para evitar que un proyecto incurra en errores y subsecuentemente caiga en el caos. Utilizando la ingeniería de software, un equipo de desarrollo puede analizar el problema y plantear una solución concreta. El desarrollar un producto útil implica que se entrega al cliente lo que desea, satisfaciendo sus requerimientos, asegurando que los activos entregados sean confiables y libre de errores.

Un software de alta calidad entrega valor a la entidad que lo desarrolla y a los usuarios finales. Al no representar un gran esfuerzo de mantenimiento ni tener gran cantidad de errores que corregir, los ingenieros de software pueden dedicar tiempo y esfuerzos a nuevos proyectos y no a corregir falencias.

En el desarrollo de software se puede definir a la calidad, como un concepto bastante complejo y multidimensional desde varias perspectivas, las dos principales: calidad del producto y calidad del proceso. Sin embargo, (Barbara Kitchenham & Pfleeger, 1996) sostienen que una mejora de los procesos no asegura la calidad del producto, ya que un modelo de procesos estandarizado puede garantizar uniformidad en la salida de los productos, pero de no ser el adecuado estos productos no cumplirían con la calidad esperada.

Según (Valenciano López, 2015), otras perspectivas son:

- **Perspectiva trascendental:** se reconoce la calidad, pero no se la describe, definiciones subjetivas se realizan mas no cuantificables.

- **Perspectiva del usuario:** se tiene en cuenta varias características, como: fiabilidad, rendimiento o eficiencia. La calidad se encuentra en estrecha relación con el cumplimiento de las necesidades y expectativas del cliente.
- **Perspectiva basada en aspectos económicos:** la calidad es medida de acuerdo a costos, precio, productividad, etc.

(McCall, Richards, & Walters, 1977) y Hewlett-Packard han propuesto diferentes factores que influyen en la calidad del software, de entre los cuales los más importantes son: Fiabilidad, Eficiencia, Rendimiento, Usabilidad, Mantenibilidad. Estos factores se pueden utilizar para construir métricas de calidad para cada una de las etapas del ciclo de vida de un proyecto software (M. Sicilia, 2012).

Por otro lado, la calidad del producto software se ha definido de manera más formal por la familia de normas ISO/IEC 25000, en donde se establece un marco de trabajo común para la evaluación de la calidad, siendo la principal base para el área de calidad de la Ingeniería de Software. (Rodríguez, Pedreira, & Fernández, 2015)

Dicho marco de trabajo hace uso de estándares internacionales y es denominado SQuaRE (Requisitos y Evaluación de Calidad de Productos de Software) (ISO, 1999), su objetivo es dirigir el proceso de desarrollo de productos software en cumplimiento con la especificación y evaluación de requisitos de calidad. En la ***Tabla 5*** se detalla cómo SQuaRE se compone:

Tabla 5
Modelo de calidad ISO/IEC 25000

ISO/IEC 2500n	División de gestión de calidad	Se definen todos los modelos comunes, términos y referencias.
ISO/IEC 2501n	División del modelo de calidad	Modelo de calidad detallado, características para la calidad interna, externa y en uso.
ISO/IEC 2502n	División de mediciones de calidad	Modelo de referencia de calidad del producto de software. Aplicaciones de métricas para la calidad interna, externa y en uso.
ISO/IEC 2503n	División de requisitos de calidad	Especificación de requisitos de calidad. Se guía en la norma ISO/IEC 15288
ISO/IEC 2504n	División de evaluación de la calidad	Requisitos, recomendaciones y guías para la evaluación de un producto.
ISO/IEC 25050 - 25099	Estándares de extensión SQuaRE	Requisitos para la calidad de productos software *"Off-The-Self"*

Concretamente, es en la norma ISO/IEC 25010 donde se definen algunos modelos de calidad y se determinan características o factores que se pueden evaluar referente a la calidad del software, como: adecuación funcional, rendimiento, compatibilidad, usabilidad, fiabilidad, seguridad, portabilidad y mantenibilidad, detallado en la ***Figura 3***. La norma ISO/IEC 25040 establece el proceso de evaluación de calidad del software, mientras que la norma ISO/IEC 25020 define las métricas de calidad.

Figura 3. Modelo de calidad de software ISO/IEC 25010
Fuente: (Rodríguez et al., 2015)

La adecuación funcional hace referencia a la capacidad que presenta un producto de software para contar con funcionalidades que cumplan con las necesidades del cliente, en las condiciones específicas de uso estimadas. (ISO, 2019) la subdivide en: completitud, corrección y pertinencia que hacen referencia al grado de cumplimiento en que las funcionalidades cubren los objetivos del usuario, la capacidad del producto para cumplir sus objetivos con la precisión esperada, y proporcionar un conjunto de funcionalidades apropiadas de acuerdo a lo requerido por el usuario, ni menos ni más.

El rendimiento, o también conocido como Eficiencia de Desempeño, es una característica representativa al desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones, como el caso del tiempo de respuesta o la capacidad de un elemento de software para cumplir con su requerimiento al límite máximo.

La compatibilidad se define entre dos o más sistemas o componentes, al compartir el mismo entorno de hardware o de software pueden intercambiar información y cumplir con funciones requeridas. La coexistencia e interoperabilidad es necesaria para que un software comparta sus recursos e información con otro sistema, en un entorno común, sin ponerse en riesgo el uno al otro.

La usabilidad se refiere a la capacidad del software para ser entendido, aprendido, usado y resultar atractivo para el usuario en determinadas condiciones. También hace referencia a la capacidad de uso, de aprendizaje del producto por parte del usuario, su nivel de protección o recuperación de errores provocados por el usuario, su interfaz y accesibilidad.

La fiabilidad del producto software se denota por su nivel de madurez, disponibilidad, tolerancia a fallos y capacidad de recuperación. Lo que quiere decir que el sistema cumpla con su

objetivo en condiciones normales, esté operativo y accesible cuando se lo requiera, continúe con su operación pese a fallos que se presenten en hardware o software recuperando datos afectados y restableciendo un estado deseado.

La seguridad representa la capacidad de mantener bajo protección la información del sistema, de tal manera que los datos de sí mismo y de los usuarios involucrados no puedan ser leídos o modificados por personas no autorizadas. La seguridad hace referencia a la confidencialidad, integridad, no repudio, responsabilidad y autenticidad.

La portabilidad es la capacidad del producto software para ser transferido de forma efectiva y eficiente de un entorno a otro, presentando adaptabilidad, facilidad para ser instalado y reemplazado por otro producto software con el mismo propósito y en el mismo entorno.

Por último, la mantenibilidad es la capacidad del producto software para ser modificado de manera efectiva y eficiente, ya sea por necesidades evolutivas, correctivas o perfectivas, este tema será abordado a más profundidad en la siguiente sección.

2.2.2.1 Mantenibilidad de un proyecto software

IEEE, (1990) define a la mantenibilidad como: “La facilidad con la que un sistema o componente de software puede modificarse para corregir fallos, mejorar el rendimiento y otros atributos, o adaptarse a un entorno modificado” (p. 46). Por lo que se puede decir que un software mantenible es ampliable y corregible de manera sencilla, algunas de las características que presenta son: facilidad para añadir nuevas funciones sin introducir errores durante el proceso, facilidad para corregir errores evitando la generación nuevos defectos y mejorar la usabilidad (Crouch & Lead, 2019) .

La mantenibilidad se encuentra estrechamente relacionada con el mantenimiento de software, siendo una característica de calidad de software afín a la facilidad de mantenimiento; a mayor mantenibilidad menor costos (Sicilia, 2012).

Tanto en las primeras etapas del SDLC como durante las fases de mantenimiento debe tomarse como prioridad la mantenibilidad, para así disminuir desde un inicio las necesidades de mantenimiento o posibles efectos colaterales ocasionados por algún error o falla. Por lo cual, la mantenibilidad se rige bajo cuatro necesidades que aparecen durante el ciclo de vida de un producto software: evolutivas, correctivas, preventivas y adaptivas (Valenciano López, 2015).

El mantenimiento de software evolutivo, también conocido como perfectivo, hace referencia a la incorporación en el producto software de nuevos requerimientos o funcionalidades, mejorando o adecuando los existentes requisitos. (Martínez, 2013) menciona que estas adaptaciones de nuevos procedimientos sirven para mejorar la operatividad del software, pero implican nuevos procesos de desarrollo a un alto costo.

El mantenimiento adaptativo hace referencia a cambios o modificaciones necesarios en el software producto de cambios en el sistema operativo, hardware, arquitectura (relacionado a las tendencias de mercado, como el cambio de cliente-servidor a web), entre otros. Este mantenimiento puede ir desde un retoque de funcionalidades o configuraciones del software hasta una completa reescritura del código, esto debido a la tendencia mundial de los últimos años a actualizar el hardware y sistemas operativos en cortos periodos de tiempo (Universidad de Valencia, 2015).

El mantenimiento correctivo corresponde a la corrección de fallos y defectos presentes en los productos software. Estos defectos pueden ser leves a los que les corresponde una corrección

rápida y fácil, pero también llegar a correcciones que requieran la reconstrucción de toda la integridad de los datos (Valenciano López, 2015).

Por último, el mantenimiento preventivo son los controles que se hacen sobre el producto software para evitar errores, usualmente causados por los mismos usuarios de forma inconsciente o consciente. A este mantenimiento también se lo conoce como restrictivo, porque al usuario se lo guía en lo que puede o no hacer a todo momento, reduciendo el margen para el surgimiento de acciones no contempladas en el producto software. (Martínez, 2013) concluye que este mantenimiento es esencial, ya que tratado adecuadamente puede prevenir mantenimientos correctivos a futuro, reduciendo costos y aumentando experiencia de usuario.

(M.-A. Sicilia, 2008) menciona la existencia de factores que afectan de manera directa a la mantenibilidad, si alguno de estos factores no se llega a cumplir satisfactoriamente la mantenibilidad se ve afectada, tres de los factores más representativos son:

- **Documentación:** el tiempo para que un equipo de mantenimiento entienda el diseño del software incrementa de manera considerable si la documentación o las especificaciones de diseño están disponibles de manera clara y concisa.
- **Proceso de desarrollo:** las técnicas de mantenimiento deben ser lo menos intrusivas al software desarrollado, para lo cual es necesario que el mantenimiento forme parte integral del proceso de desarrollo, incorporada intrínsecamente en los productos software.
- **Comprensión de programas:** La comprensión humana es uno de los principales causantes de mantenimientos extensos y costosos, ya sea porque la información

disponible es incomprensible o insuficiente, o que se mal interpreta el programa o sistema.

2.2.2.2 Métricas de mantenibilidad

Las métricas de mantenibilidad miden atributos tales como el comportamiento del usuario encargado del mantenimiento cuando el software se mantiene o modifica durante las pruebas o mantenimientos (Valenciano López, 2015).

La ISO/IEC 25000, específicamente la división 25010, establece a la mantenibilidad con cuatro sub dimensiones, cada una de las cuales presenta sus propias métricas, como se detalla en la ***Tabla 6***.

Tabla 6
Métricas de mantenibilidad

Sub dimensión	Detalle	Métricas
Analizabilidad	Mide los recursos necesarios para diagnosticar deficiencias o partes a modificar.	• Complejidad ciclomática • Código repetido • Densidad comentarios • Densidad de defectos de la capacidad para ser analizado
Modificabilidad	Mide el esfuerzo de la persona encargada del mantenimiento y el usuario cuando se intenta implementar una modificación.	• Densidad de ciclos • Documentación publicada • Nombres de las variables • Densidad de defectos de la modificabilidad
Capacidad de ser probado	Mide el esfuerzo de la persona encargada del mantenimiento midiendo su comportamiento cuando trata de probar el software modificado.	• Densidad de pruebas • Tamaño del programa
Reusabilidad	Mide la capacidad que tiene el programa para poder reusarse para nuevos propósitos	• Estructura del programa • Deuda técnica • Densidad de defectos de Reusabilidad

Fuente: (Valenciano López, 2015)

Para el presente experimento se utilizarán dos métricas definidas por los autores que tienen como base las métricas de mantenibilidad descritas en el experimento usado como ejemplo por (Genero et al., 2014), mismas que se detallan a continuación:

- **Efectividad de la mantenibilidad ($Mefec$):** Representa qué tan efectivos son los sujetos experimentales al ejecutar de forma correcta las tareas de mantenimiento. La fórmula consiste en la suma ponderada (Calderón Pascual, 2016) del resultado de cada una de las tareas de mantenimiento T_i por el factor de corrección $\left(\frac{Cu_i}{Pu_i}\right)$, y dividido para el número total de tareas.

  Dando como resultado un valor decimal entre 0 y 1 que representa el porcentaje de tareas que el sujeto es capaz de completar correctamente. Se calcula mediante la siguiente fórmula:

$$mefec = \frac{\sum_{i=1}^{n} T_i \left(\frac{Cu_i}{Pu_i}\right)}{n}$$

  Donde:

  - $n =$ Número total de tareas de mantenimiento propuestas.

    $T_i =$ Tarea completada (1) o no completada (0).
  - $Cu_i =$ Pruebas unitarias correctas por tarea.
  - $Pu_i =$ Pruebas unitarias propuestas por tarea.

- **Eficiencia de la mantenibilidad ($Mefic$):** Representa cuán rápidos son los sujetos experimentales al ejecutar de forma correcta las tareas de mantenimiento. La fórmula consiste en la suma ponderada (Calderón Pascual, 2016) del resultado de

cada una de las tareas de mantenimiento T_i por el factor de corrección $\left(\frac{Cu_i}{Pu_i}\right)$, y dividido para el tiempo total de ejecución de las tareas.

Dando como resultado un valor decimal mayor o igual a 0, medido en tareas por hora, que representa la velocidad con la que el sujeto es capaz de completar correctamente las tareas. Se calcula mediante la siguiente fórmula:

$$mefic = \frac{\sum_{i=1}^{n} T_i \left(\frac{Cu_i}{Pu_i}\right)}{t}$$

Donde:

- $t =$ Tiempo total utilizado para resolver las tareas de mantenimiento, medido en horas.
- $T_i =$ Tarea completada (1) o no completada (0).
- $Cu_i =$ Pruebas unitarias correctas por tarea.
- $Pu_i =$ Pruebas unitarias propuestas por tarea.

2.2.3 Experimentos en Ingeniería de Software

En ingeniería de software los experimentos son considerados como una investigación empírica, donde su propósito es la manipulación de una variable independiente para posteriormente analizar cuál es su efecto en una variable dependiente (Genero et al., 2014).

2.2.3.1 Proceso experimental

Para el desarrollo de cualquier experimento es necesario mantener un proceso consistente, que permita definir claramente su alcance, objetivo y desarrollo. En el caso del presente trabajo de

investigación, se ha tomado como base el proceso experimental detallado en el libro “Métodos de investigación en ingeniería del software” (Genero et al., 2014), como se detalla en la ***Figura 4***.

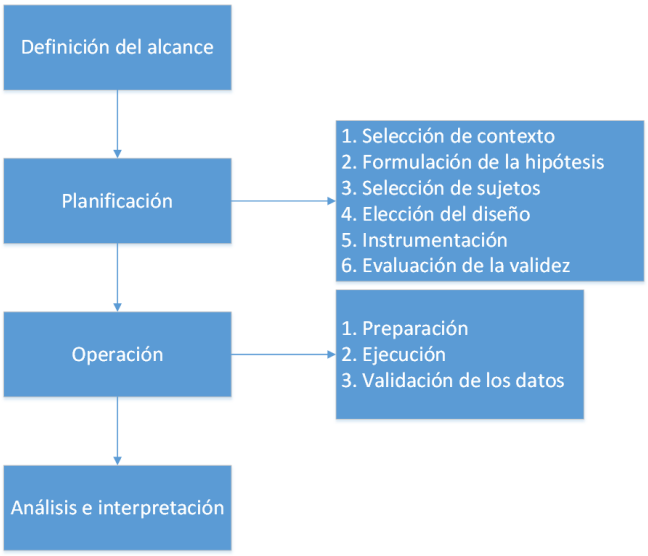

Figura 4. Proceso Experimental
Fuente: (Genero et al., 2014)

2.2.3.1.1 Definición del alcance

Dentro de esta etapa se precisa definir el objetivo general del experimento, mediante el cual se establece el alcance que va a tener la investigación.

Existen varias plantillas para la definición de objetivos, y debido a la necesidad de evaluar los resultados del trabajo mediante métrica y datos se hará uso de la plantilla *Goal-Question-Metric* (Basili, Caldiera, & Rombach, 1994).

2.2.3.1.2 Planificación

En la fase de planificación se tiene una serie de tareas que ayudarán a mantener una noción clara de cómo se llevará a cabo el experimento, las tareas son:

Selección del contexto

Se determina el ambiente sobre el cual será realizado el experimento, de esta manera se puede decidir sobre qué situaciones los resultados del experimento son válidos. Generalmente el contexto de un experimento está determinado por cuatro dimensiones, según (Genero et al., 2014) estas son:

- Online vs Offline
- Profesionales vs Estudiantes
- Problemas “de juguete” o Proyectos reales
- Específico o General

Formulación de la hipótesis

Dentro de un experimento se debe definir lo que se quiere probar, esto puede ser expresado mediante una hipótesis. Generalmente se parte de una o varias hipótesis nulas, las cuales son las suposiciones a rechazar mediante la ejecución del experimento.

Selección de variables

Existen varios tipos de variables que se deben tomar en cuenta durante un experimento, las cuales son:

- **Variables independientes:** se modifican para estudiar el efecto que producen esos cambios.
- **Variables dependientes:** se observan para determinar los cambios producidos por las variables independientes.

- **Variables controladas:** variables independientes que pueden ser controladas de manera fija
- **Variables enmascaradas:** variables que no pueden ser controladas y también pueden producir cambios en las variables dependientes.
- **Variables aleatorias:** variables no controladas que pueden tratarse como errores aleatorios.

Según (Genero et al., 2014) la selección de variables también involucra establecer su escala de medición y los valores que estas puedan tomar. Tomando en cuenta cómo minimizar el efecto que las variables no controladas tienen durante la ejecución del experimento.

Elección del diseño

La elección del diseño experimental es un paso crucial durante la elaboración de un experimento ya que este es la base para la replicación y confirmación del experimento. Dentro de esta etapa se tiene varias técnicas de control que permiten minimizar el error experimental (Genero et al., 2014):

- **Aleatorización**: se refiere a la distribución aleatoria de los tratamientos y los sujetos que van a realizarlos, igualmente a la selección aleatoria de sujetos dentro de una muestra.
- **Bloqueo**: permite eliminar el efecto no deseado de un factor dentro del experimento, de esta manera se evita sesgos al momento de procesar los datos obtenidos.
- **Balanceo o equilibrado**: útil al momento de la asignación de tratamientos, se cumple cuando a cada tratamiento se le asigna un número igual de sujetos.

Para encontrar el diseño experimental apropiado es necesario tomar en cuenta dos criterios: la cantidad de variables independientes y el número de tratamientos que se van a asignar a cada sujeto. En la ***Tabla 7*** se establecen diferentes tipos de diseños experimentales, basados en los criterios antes mencionados.

Tabla 7
Diseños experimentales

Tipo	Diseño experimental
Experimentos simples	• Inter – sujetos (Un solo tratamiento) • Intra – sujetos (Todos los tratamientos)
Experimentos factoriales	• Completos • Fraccionales • Parciales

Fuente: (Genero et al., 2014)

Selección de sujetos

La selección de sujetos está atada a una de las dimensiones descritas anteriormente en la selección del contexto. Un sujeto es el individuo sobre el cual se ejecuta el experimento, de tal manera que la selección de sujetos tiene un efecto pronunciado en los resultados obtenidos durante la ejecución (Genero et al., 2014).

Instrumentación

La instrumentación pretende establecer una metodología para la ejecución de un experimento, sin afectar al grado de control que se mantiene sobre este.

La validez de un experimento depende mucho de cómo este haya sido instrumentado, siendo que los resultados obtenidos deben ser los mismos independientemente de cómo se haya instrumentado. Esto quiere decir que si los instrumentos afectan el resultado del experimento este no será válido (Jedlitschka & Tn, 2005).

Evaluación de la validez

Antes de realizar cualquier experimento es importante estimar qué tan validos serán los resultados, es decir, que se debe considerar los posibles factores que puedan ser amenazas a la validez para planificar como mitigarlas.

Existen tres tipos de amenazas a la validez, descritas por (Shadish, W., Cook, T., Campbell, 2005):

- **A la validez interna:** afectan el grado de confianza que se tiene en la relación causa-efecto que mantienen las variables independiente y dependiente.

  Algunos factores que pueden influir en la validez interna pueden ser: la selección y agrupación de sujetos, su trato durante el experimento, los materiales utilizados y cualquier evento extraño que pueda suceder durante la ejecución.
- **A la validez externa:** afectan al grado en el cual los resultados pueden ser generalizados teniendo en cuenta la población seleccionada para el experimento. La validez externa de un experimento se ve afectada por: el diseño experimental, la selección de sujetos, el entorno y temporalidad del experimento.
- **A la validez de constructo:** la principal amenaza a la validez de constructo es la ausencia de pruebas teóricas que afirmen que las variables dependientes e independientes miden realmente los conceptos detallados en el experimento.
- **A la validez de la conclusión:** se determina mediante qué tan estadísticamente significativos fueron los resultados del experimento, además de la fiabilidad de las medidas

Es necesario recalcar que en un experimento es difícil tomar en cuenta todas las amenazas a la validez, por lo cual es necesario detallar cuales fueron mitigadas, cuáles no, y por qué (Genero et al., 2014).

2.2.3.1.3 Operación

Dentro de esta fase se lleva a cabo el experimento y se recogen los datos que van a ser analizados. En esta fase se realizan las siguientes tareas:

Preparación

Dentro de esta fase se realizan todas las actividades necesarias para la ejecución del experimento, es decir que principalmente se trabaja con los sujetos experimentales, de manera que estos estén preparados para realizar los tratamientos especificados.

Existen varias consideraciones éticas que se deben tomar en cuenta, ya que por lo general un experimento en ingeniería de software involucra seres humanos:

- Contar con el consentimiento de los sujetos.
- El rendimiento de cada sujeto debe ser confidencial.
- Se debe ofrecer un incentivo por realizar el experimento.
- Es necesario comunicar todos los aspectos del experimento, siempre y cuando estos no introduzcan sesgos en los resultados.

Además, en la fase de preparación se debe entrenar a los sujetos en el tema de estudio para que sean capaces de realizar los tratamientos que les serán asignados. También es recomendable

pedir a los sujetos que llenen una encuesta sobre datos personales, experiencia y conocimientos previos. Así se tendrá un análisis más granular de los resultados (Genero et al., 2014).

Ejecución

Durante la ejecución del experimento es preferible reunir a todos los sujetos en un mismo lugar, de esta manera los resultados serán más sencillos de recoger y se puede resolver cualquier duda que tengan los sujetos experimentales. Para que esta etapa sea exitosa es importante comunicar a los sujetos de manera detallada las tareas a realizar, las restricciones que tienen para desarrollarlas y el objetivo del experimento.

Validación de los datos

Después de la recolección de los datos es necesario tomar en cualquier anomalía detectada al momento de la ejecución del experimento, de esta manera en la etapa de procesamiento se podrán filtrar estos datos de ser necesario.

2.2.3.1.4 Análisis e interpretación

Para obtener los resultados del experimento es necesario realizar un análisis estadístico (cuantitativo) mediante: estadísticos descriptivos, reducción de datos y contraste de hipótesis.

En la ***Tabla 8*** se puede evidenciar los test estadísticos más adecuados para realizar el contraste de hipótesis, en función del tipo de diseño experimental seleccionado y la distribución de los datos. Se utilizarán test paramétricos en caso de que la distribución de los datos sea normal, caso contrario se utilizarán no paramétricos. Una vez realizado el análisis cuantitativo se pueden interpretar los resultados y presentar las conclusiones del experimento.

Tabla 8
Test estadísticos según el diseño experimental

Tipo de diseño	Test paramétricos	Test no paramétricos
Un factor, un tratamiento		• Test binomial • Chi 2
Un factor, dos tratamientos	• Test T • Test F • Test T emparejado	• Mann-Whitney • Chi 2 • Wilcoxon
Un factor, más de dos tratamientos	• ANOVA	• Kruskal-Wallis • Chi 2
Más de un factor	• ANOVA	

Fuente: (Genero et al., 2014)

2.2.4 Test estadísticos

Son utilizados para ayudar a la toma de decisión de si una suposición sobre una población se puede confirmar o rechazar, mediante la observación de una muestra. El test estadístico consiste en plantear una hipótesis nula sobre la población y aplicar el test adecuado, en contraste con la hipótesis, sobre la muestra observada (Ferrán Aranaz, 2001).

La probabilidad de encontrar diferencias significativas o equivalentes entre los resultados observados y los teóricos del estudio debe ser muy pequeña, por lo que se debe fijar un nivel de significancia tal que los sucesos con probabilidad menor lleven a rechazar la hipótesis nula. El nivel de significancia, por lo general, toma valores de 0.1, 0.05 o 0.01.

A dicha probabilidad se le denomina p-valor, por lo que, si el p-valor es menor que el nivel de significancia escogido, la diferencia entre la observación de la muestra y lo esperado por la hipótesis nula será estadísticamente significativa rechazando la hipótesis bajo estudio.

Previo a ejecutar un test estadístico sobre una muestra, es necesario comprobar si los datos siguen una distribución normal o no, para así, aplicar el test estadístico apropiado. Kolmogorov-Smirnov es un test de comprobación de normalidad para una muestra (Ferrán Aranaz, 2001).

Kolmogorov-Smirnov es una prueba de bondad de ajuste; es decir, comprueba que la muestra se origine de una población en donde la distribución de sus datos (F) es una distribución teórica normal (FO), originando la siguiente hipótesis nula a contrastar: $H_O: F = F_O$.

Si el p-valor resultante, también conocido como Z, es menor que el nivel de significancia escogido arbitrariamente se rechaza la hipótesis nula, es decir, los datos no siguen una distribución normal. En el contexto del presente estudio se utilizará el test no paramétrico de Wilcoxon cuando los datos de la muestra no sigan una distribución normal, caso contrario se utilizará el test paramétrico T de Student.

CAPÍTULO III

DISEÑO Y PLANIFICACIÓN DEL EXPERIMENTO

3.1 Definición del alcance

El presente experimento tiene como objetivo analizar la influencia del uso de los paradigmas de programación procedimental y orientado a objetos en la mantenibilidad de un proyecto software, el mismo se realiza en el contexto de estudiantes de pregrado de la carrera de Software y de la carrera de Ingeniería en Sistemas e Informática de la Universidad de las Fuerzas Armadas ESPE.

3.2 Selección del contexto

3.2.1 Objetos experimentales

Para el presente contexto, se han seleccionado tres objetos experimentales: una interface de usuario web (frontend) que no será modificada durante el experimento, y dos APIs REST (backends) con los mismos diez servicios web cada uno, programados usando dos paradigmas de programación diferentes:

- Frontend: Interfaz de usuario desarrollada utilizando HTML, CSS y JavaScript, por medio de la cual los sujetos interactúan con los backends.
- $\boldsymbol{B_{proc}}$ **:** Backend desarrollado utilizando el paradigma de programación procedimental (PROC), el lenguaje de programación Python y Flask.

- $\boldsymbol{B_{poo}}$**:** Backend desarrollado utilizando el paradigma de programación orientado a objetos (POO), el lenguaje de programación Java y Spring Boot.

Cada backend, en conjunto con el frontend forma una aplicación web encargada de la gestión de una pizzería con los siguientes requerimientos funcionales:

- Crear, actualizar y leer ingredientes.
- Crear y leer tamaños de pizza.
- Crear, leer y calcular el precio de una orden.

Para mayor detalle de los requerimientos funcionales ver el ***Anexo A***.

En ambos casos, cada backend cumple con la misma funcionalidad. Serán programados siguiendo el patrón Modelo-Vista-Controlador (MVC), utilizando el editor de código Visual Studio Code, manteniendo la misma complejidad ciclomática e introduciendo los mismos errores a ser solucionados por los sujetos. En la ***Tabla 9*** se detallan características adicionales de cada backend. (Ramirez, Reyes, & Gil, 2014)

Tabla 9
Métricas generales de cada backend

Métrica	B_{proc}	B_{poo}
SLOC (Líneas de código significativas)	509	612
CCOM (Complejidad ciclomática)	A	A
NFILES (Número de archivos)	10	45

3.2.2 Sujetos experimentales

Se cuenta con la participación de noventa estudiantes de la Universidad de las Fuerzas Armadas ESPE con la siguiente distribución:

- 32 estudiantes de tercer semestre de Software de la asignatura Programación Orientada a Objetos.
- 40 estudiantes de sexto semestre de Ingeniería en Sistemas e Informática de la asignatura Programación Avanzada.
- 18 estudiantes de octavo semestre de Ingeniería en Sistemas e Informática de la asignatura Inglés Técnico.

Los grupos de sujetos experimentales han sido seleccionados porque cuentan con los suficientes conocimientos de: programación, algoritmos, resolución de problemas e ingeniería de software para ejecutar de manera adecuada las tareas de mantenimiento propuestas en el experimento.

La participación de los sujetos en el experimento es voluntaria. Para evitar temor a la evaluación que pudiera ocasionar amenazas a la validez, se utilizan como base las medidas sugeridas por Genero et al., (2014) las cuales consisten en comunicar a los estudiantes que su desempeño en el experimento no será calificado y únicamente, en las evaluaciones de la asignatura se incluirán temas relacionados al experimento.

Por otro lado, para que todos los sujetos compartan una base de conocimientos común, realizó una capacitación acerca de ambos paradigmas de programación y uso de las herramientas necesarias para ejecutar el experimento, tales como: Paradigma Orientado a Objetos, Paradigma Procedimental, Python, Flask y Java, Spring Boot, buenas prácticas de programación, bases de datos, HTTP, REST y JSON.

En el contexto de este experimento se considera apropiado trabajar con estudiantes (Host, Regnell, & Wohlin, 2000) debido a que:

- La ejecución de las tareas propuestas no requiere de experiencia profesional.
- Cada grupo de estudiantes tiene un nivel de conocimientos homogéneo.
- Existe una mayor facilidad para monitorear el experimento.
- La gran cantidad de sujetos experimentales disponibles.

Finalmente, durante sus cinco años de estudio en la carrera de ingeniería en sistemas e informática, los autores constataron que el currículo impartido (Departamento de Ciencias de la Computación, 2019) estaba fuertemente orientado al lenguaje de programación Java y a la programación orientada a objetos. Razón por lo cual es posible que los sujetos de semestres superiores tengan mayor experiencia con el paradigma y lenguaje de programación mencionados.

3.3 Selección de variables

3.3.1 Variable independiente

Elección del paradigma de programación, la cual es una variable nominal que toma dos valores: ***PROC*** (Procedimental) y ***POO*** (Orientado a Objetos).

3.3.2 Variable dependiente

Mantenibilidad de un proyecto de software medida respecto a los paradigmas de programación, utilizando las siguientes métricas, para más detalle sobre las métricas utilizadas referirse a la sección 2.2.1:

Efectividad del Mantenimiento (*Mefic*): Esta métrica está relacionada con la capacidad de un sujeto de ejecutar correctamente un conjunto de tareas de mantenimiento. Es calculada con la siguiente formula:

$$mefec = \frac{\sum_{i=1}^{n} T_i \left(\frac{Cu_i}{Pu_i}\right)}{n}$$

Eficiencia del Mantenimiento (*Mefec*): Esta métrica está relacionada con el tiempo que le toma a un sujeto terminar correctamente un conjunto de tareas de mantenimiento. Es calculada con la siguiente formula:

$$mefic = \frac{\sum_{i=1}^{n} T_i \left(\frac{Cu_i}{Pu_i}\right)}{t}$$

3.4 Formulación de hipótesis

En la **Tabla 10** se describen las hipótesis nulas y alternativas relacionadas con la variables dependiente e independiente, el objetivo del análisis estadístico será rechazar las hipótesis nulas y posiblemente aceptar alguna de las hipótesis alternativas.

Tabla 10
Hipótesis nulas y alternativas del experimento

Hipótesis Nula	Hipótesis alternativas
$\mathbf{H_{0,1}: Mefec_{poo} = Mefec_{proc}}$	$H_{1,1,1}: Mefec_{poo} \neq Mefec_{proc}$ $H_{1,1,2}: Mefec_{poo} > Mefec_{proc}$ $H_{1,1,3}: Mefec_{poo} < Mefec_{proc}$
$\mathbf{H_{0,2}: Mefic_{poo} = Mefic_{proc}}$	$H_{1,2,1}: Mefic_{poo} \neq Mefic_{proc}$ $H_{1,2,2}: Mefic_{poo} > Mefic_{proc}$ $H_{1,2,3}: Mefic_{poo} < Mefic_{proc}$

3.5 Elección del diseño

Para este experimento, de acuerdo con los valores que puede tomar la variable independiente, se han definido los siguientes tratamientos:

- Tratamiento I: Ejecutar para $\boldsymbol{B_{proc}}$ las tareas de mantenimiento adaptivo y correctivo especificadas en el instructivo de mantenimiento.
- Tratamiento II: Ejecutar para $\boldsymbol{B_{poo}}$ las tareas de mantenimiento adaptivo y correctivo especificadas en el instructivo de mantenimiento.

Para los grupos de Programación Orientada a Objetos y Programación Avanzada (72 sujetos), se utilizará un diseño intra-sujetos descrito por Otero & Dolado, (2000), con las siguientes características: cada grupo de sujetos es separado en dos subgrupos con igual cantidad de sujetos seleccionados al azar, a cada subgrupo se le asigna un orden diferente para la ejecución de los tratamientos, resultando en que cada sujeto experimental realizará ambos tratamientos.

El experimento duró dos días. Al inicio el subgrupo I desarrolló el tratamiento I y el subgrupo II, el tratamiento II. El segundo día se intercambiaron los tratamientos, por consiguiente, el subgrupo I desarrolló el tratamiento II y el subgrupo II, el tratamiento I.

Es importante mencionar que se han tomado medidas sugeridas por Wohlin, (2014) para minimizar el efecto de aprendizaje que podrían experimentar los sujetos una vez ya desarrolladas las tareas de mantenimiento. Las tareas y los paradigmas se asignaron en orden aleatorio.

Para el grupo de Inglés Técnico (18 sujetos), se usó un diseño balanceado inter-sujetos como en Otero & Dolado, (2000), con las siguientes características: el grupo es dividido

aleatoriamente en dos subgrupos de igual cantidad de sujetos, cada subgrupo con un único tratamiento.

El orden de ejecución de las tareas de mantenimiento será diferente para cada sujeto, mitigando de esta manera el efecto que este orden pueda tener en su desempeño.

En la ***Tabla 11***, se puede observar la distribución de los sujetos y tratamientos para el experimento, tomando en cuenta los diferentes diseños experimentales.

Tabla 11
Distribución de sujetos y tratamientos para el experimento

Día 1	Día 2	Día 3	Día 4
	POO – S1 – T1	POO – S1 – T2	IT – S1 – T1
	POO – S2 – T2	POO – S2 – T1	IT – S2 – T2
PA – S1 – T1	PA – S1 – T2		
PA – S2 – T2	PA – S2 – T1		

Donde:

- **PA:** Curso de Programación Avanzada
- **POO:** Curso de Programación Orientada a Objetos
- **IT:** Curso de Inglés Técnico
- **S1 y S2:** Grupo o sección de sujetos seleccionados al azar.
- **T1:** Paradigma de Programación Procedimental
- **T2:** Paradigma de Programación Orientado a Objetos.

3.6 Instrumentación

La valoración de las métricas de mantenibilidad y las tareas de mantenimiento se desarrolló con los siguientes instrumentos:

- Un backend en dos versiones, utilizando los paradigmas de programación: Programación Orientada a Objetos (Java y Spring) y Programación Procedimental (Python y Flask).
- Un frontend desarrollado en JavaScript, HTML y CSS cuya función es ayudar a los sujetos a probar el sistema en conjunto.

Para cada versión de backend se escribió un paquete de pruebas unitarias, las cuales permitieron comprobar si los mantenimientos realizados por los sujetos fueron exitosos y calcular el porcentaje de corrección necesario para las métricas del experimento.

Se considera un instructivo de mantenimiento, detallando la duración, objetivo, definición, requerimientos del sistema e instrucciones para el desarrollo de las siguientes tareas:

- Corregir la funcionalidad para leer ingredientes
- Implementar la funcionalidad para obtener la lista de tamaños de pizza disponibles
- Corregir la funcionalidad para el registro de los datos del cliente para una orden
- Corregir la funcionalidad para obtener información de una orden almacenada en el sistema
- Corregir el cálculo del precio total de una orden

Cada tarea de mantenimiento tiene como único objetivo la modificación del código del backend. Fueron detalladas de la siguiente manera:

- Título de la tarea
- Tipo de mantenimiento
- Requerimiento
- Descripción del error a corregir
- Pasos para reproducir el error mediante el frontend y la URL de los servicios web
- Salida esperada (correcta)
- Salida actual (incorrecta)
- Duración estimada para la ejecución del mantenimiento

Para cada grupo de estudiantes (POO, PA e IT) se configuró los siguientes repositorios GitHub: PROC-OOP-Experiment-OP[1], PROC-OOP-Experiment-PA[2], y PROC-OOP-Experiment-IT[3]. Dentro de los cuales cada sujeto manejara su rama personal (*branch*) con los instructivos de mantenimiento y el código fuente del frontend y ambas versiones de backend, lo que permitió revisar de manera eficiente los cambios realizados durante la ejecución del experimento.

Finalmente, como parte de la instrumentación, se diseñaron tres encuestas:

- Previo al entrenamiento, con módulos demográfico y de conocimiento previos.
- Post entrenamiento
- Post experimento

[1] https://github.com/Wason1797/PROC-OOP-Experiment-OOP
[2] https://github.com/Wason1797/PROC-OOP-Experiment-PA
[3] https://github.com/Wason1797/PROC-OOP-Experiment-IT

Las preguntas realizadas fueron de respuesta simple y en el caso de medir niveles de acuerdo, se utilizó una escala de Likert de cinco puntos. Esta información será útil para interpretar y explicar los resultados obtenidos.

El modelo de instructivo de mantenimiento y los enlaces a los repositorios de GitHub se encuentran en el ***Anexo B***, la encuesta pre entrenamiento en el ***Anexo C***, la encuesta post entrenamiento en el ***Anexo D***, y la encuesta post experimento se encuentra en el ***Anexo E***.

3.7 Operación

3.7.1 Preparación

Como parte de la preparación para el desarrollo del experimento se llevaron a cabo las siguientes actividades:

Se distribuye a los sujetos como se muestra en la

a) ***Tabla 13***, que corresponde con los horarios de clase de cada asignatura.

- **PA:** Curso de Programación Avanzada
- **POO:** Curso de Programación Orientada a Objetos
- **IT:** Curso de Inglés Técnico
- **S1 y S2:** Grupo o sección de sujetos seleccionados al azar.
- **T1:** Entrenamiento en el paradigma de programación procedimental
- **T2:** Entrenamiento en el paradigma de programación orientado a objetos.

El tiempo que se utiliza para cada entrenamiento está de acuerdo al nivel de conocimientos generales de cada grupo: cinco días para los sujetos de menor nivel, dos días para los sujetos de nivel intermedio y un día para los sujetos de nivel avanzado (cada día representa 2 horas-reloj de clase).

b) Se solicita a los sujetos que realicen la encuesta de conocimientos previos.

c) Se desarrolla un entrenamiento dos semanas antes de la ejecución del experimento, en el cual los sujetos fueron capacitados en cada paradigma de programación, conceptos generales y herramientas necesarias.

d) Concluido el entrenamiento se solicita que los sujetos realicen la encuesta post entrenamiento.

e) Se llevó a cabo un estudio piloto con estudiantes de nivel intermedio de conocimientos, los cuales no participaron en el experimento. Esto sirvió para corregir el material de entrenamiento y definir el tiempo total del experimento en dos horas.

Tabla 12
Distribución de sujetos y tratamientos para el entrenamiento, semana 1

Día 1	Día 2	Día 3	Día 4	Día 5
POO – S1 – T1	POO – S1 – T1	POO – S1 – T1	POO – S1 – T1	POO – S1 – T1
POO – S2 – T2	POO – S2 – T2	POO – S2 – T2	POO – S2 – T2	POO – S2 – T2
PA – S1 – T1		PA – S1 – T1		IT – S1 – T2
PA – S2 – T2		PA – S2 – T2		IT – S2 – T2

Tabla 13
Distribución de sujetos y tratamientos para el entrenamiento, semana 2

Día 1	Día 2	Día 3	Día 4	Día 5
POO – S1 – T2	POO – S1 – T2	POO – S1 – T2	POO – S1 – T2	POO – S1 – T2
POO – S2 – T1	POO – S2 – T1	POO – S2 – T1	POO – S2 – T1	POO – S2 – T1
PA – S1 – T2		PA – S1 – T2		IT – S1 – T1
PA – S2 – T1		PA – S2 – T1		IT – S2 – T1

3.7.2 Ejecución

a) Se realizó una nueva distribución de los sujetos según el esquema descrito en la sección 3.5 y la ***Tabla 11***.

b) Se distribuyó el material necesario para realizar las tareas de mantenimiento (repositorio GitHub con el manual y código fuente).

c) Se dio las indicaciones sobre la ejecución del experimento:

1. Durante el experimento no se permitirá la comunicación entre los sujetos.
2. Cualquier duda debe ser consultada con los experimentadores.
3. Cuando los experimentadores lo indiquen los sujetos podrán ejecutar la aplicación web.
4. Una vez todos los sujetos hayan ejecutado la aplicación, se registra la hora de inicio y podrán empezar a desarrollar las tareas de mantenimiento.
5. Las tareas de mantenimiento deberán ser realizadas en el orden detallado en su manual.
6. Cada vez que un sujeto complete una tarea deberá indicarlo a los experimentadores para su registro en la rúbrica.

d) Una vez concluido el tiempo estimado para el experimento, los sujetos deberán subir sus cambios en el código a en su respectiva rama del repositorio GitHub.

Para evitar posibles sesgos en los resultados no se comunicó a los sujetos las hipótesis bajo estudio, solo se les dijo que el presente experimento era parte de una investigación sobre el uso de los frameworks Spring y Flask en el ámbito académico.

3.7.3 Validación de los datos

Una vez concluido el experimento, se recolectó los datos de la rúbrica y los componentes necesarios para calcular cada una de las métricas detalladas en la sección 3.3.2. Las tareas incorrectas no se puntúan negativamente.

En caso de detectar valores atípicos estos serán excluidos. Por ejemplo, si un sujeto que debía realizar dos tratamientos solo ejecutaba uno sus datos fueron descartados.

CAPÍTULO IV

ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

En este capítulo se presentan los resultados del análisis de los datos obtenidos en el experimento. Estos resultados se concentran en la eficiencia y la efectividad de los sujetos al ejecutar las tareas de mantenimiento, ya sea usando el paradigma orientado a objetos o el paradigma procedimental. Se analizaron semejanzas y/o diferencias entre las métricas de mantenibilidad propuestas en la sección 3.3.2, para determinar si las hipótesis nulas propuestas en el diseño del experimento pueden ser rechazadas.

Para el análisis de los resultados antes mencionados se sigue el siguiente procedimiento:

- Realizar un estudio de los estadísticos descriptivos para las métricas de la variable dependiente: efectividad del mantenimiento y eficiencia del mantenimiento.
- Comprobar el tipo de distribución de los datos por medio del test de normalidad de Kolmogorov-Smirnov para decidir qué tipo de test estadístico utilizar: paramétrico o no paramétrico.
- Evaluar la hipótesis nula mediante test estadísticos apropiados definidos en el punto anterior.
- Comparar los resultados obtenidos de las encuestas para evaluar el impacto del entrenamiento y la dificultad percibida por los sujetos al realizar las tareas de mantenimiento usando ambos paradigmas.

Para todos los test de hipótesis se consideró un nivel de significancia del 95% (Cobo et al., 2007). Los resultados que se presentarán durante este capítulo fueron obtenidos mediante el paquete estadístico SPSS, el programa utilizado y la base de datos se detallan en el ***Anexo F*.**

4.1 Análisis de los estadísticos descriptivos

Para obtener información adicional acerca del comportamiento de los sujetos se realizó el análisis de los estadísticos descriptivos de manera: general, por edad, por género, por asignatura y por orden en que fueron aplicados los tratamientos.

Análisis General

En la ***Tabla 14*** se muestran los estadísticos descriptivos generales para las dos métricas de mantenibilidad, aplicadas a los dos paradigmas de programación. Donde se observa que la eficiencia de los sujetos usando el paradigma procedimental (3.92 tareas correctas de mantenimiento por hora) es mayor que la eficiencia al usar el paradigma orientado a objetos (2.82 tareas/hora). Es decir, en promedio, los sujetos realizan correctamente una tarea más por hora usando el paradigma procedimental.

Tabla 14
Estadísticos descriptivos generales

	Casos Válidos	Valor Mínimo	Valor Máximo	Media	Desviación Estándar
Efectividad del mantenimiento en el paradigma procedimental	81	0.00	1.00	***0.8228***	0.27444
Efectividad del mantenimiento en el paradigma orientado a objetos	81	0.00	1.00	0.7395	0.29397
Eficiencia del mantenimiento en el paradigma procedimental (tareas/hora)	81	0.00	12.86	***3.9191***	2.51813
Eficiencia del mantenimiento en el paradigma orientado a objetos (tareas/hora)	81	0.00	10.00	2.8247	2.15826
N Válidos por lista	72				

Análisis por Género

En la ***Tabla 15*** se encuentran los estadísticos descriptivos agrupados por el género de los sujetos: hombres y mujeres. Al analizar los datos, es posible notar que los resultados obtenidos por las mujeres en ambos paradigmas de programación son mejores que los resultados obtenidos por los hombres.

Tabla 15
Estadísticos descriptivos por género

Género del estudiante		Efectividad del Mantenimiento en el Paradigma Orientado a Objetos	Efectividad del Mantenimiento en el Paradigma Procedimental	Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)	Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora)
Hombre	Media	0.7261	0.8141	2.7415	3.8384
	Cantidad	69	71	69	71
	Desviación estándar	0.29087	0.28438	2.13229	2.51098
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	10.00	12.86
	Varianza	0.085	0.081	4.547	6.305
Mujer	Media	0.8167	0.8850	3.3033	4.4925
	Cantidad	12	10	12	10
	Desviación estándar	0.31286	0.18864	2.34034	2.62844
	Mínimo	0.20	0.40	0.38	0.62
	Máximo	1.00	1.00	8.33	10.56
	Varianza	0.098	.036	5.477	6.909
Total	Media	0.7395	.8228	2.8247	3.9191
	Cantidad	81	81	81	81
	Desviación estándar	0.29397	0.27444	2.15826	2.51813
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	10.00	12.86
	Varianza	0.086	0.075	4.658	6.341

Respecto a la diferencia entre paradigmas de programación (***Figura 5*** y ***Figura 6***), se puede observar que tanto hombres como mujeres presentan mejores resultados en el paradigma procedimental, tanto en la efectividad como en la eficiencia del mantenimiento.

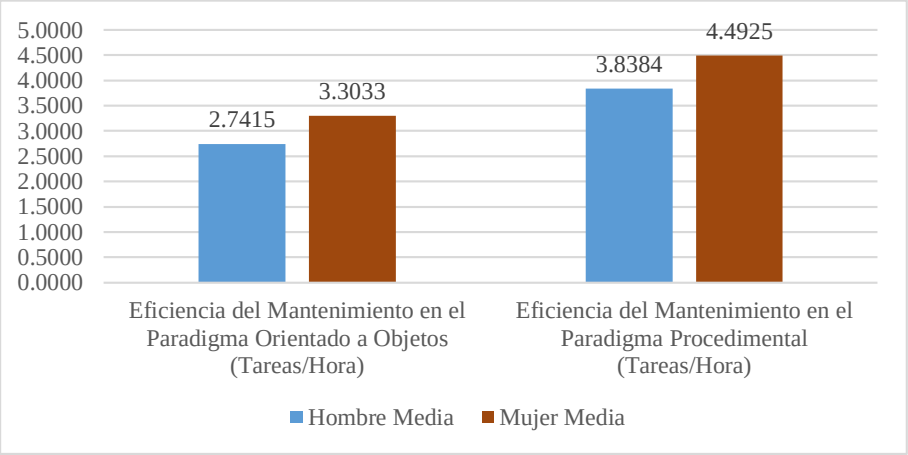

Figura 5. Medias agrupadas por género para eficiencia del mantenimiento

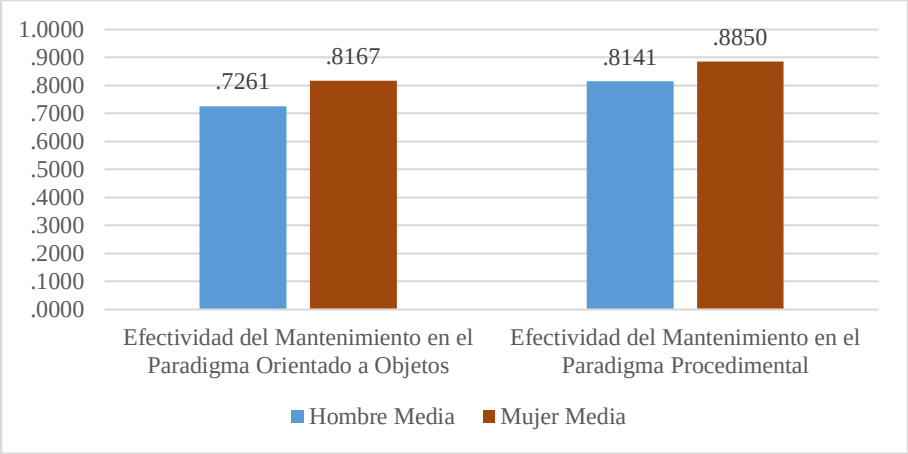

Figura 6. Medias agrupadas por género para efectividad del mantenimiento

Análisis por Edad

En la ***Tabla 16***, se encuentran los estadísticos descriptivos distribuidos por los tres grupos de edad con las frecuencias más altas dentro de la muestra: de 18 a 21 años, de 22 a 24 años y de 25 años en adelante. El grupo de sujetos con las medias más elevadas para ambos paradigmas de programación es el comprendido entre 22 y 24 años, seguido por el grupo entre 18 y 21 años.

Tabla 16
Estadísticos descriptivos por grupo de edad

Grupos de Edad		Efectividad del Mantenimiento en el Paradigma Orientado a Objetos	Efectividad del Mantenimiento en el Paradigma Procedimental	Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)	Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora)
Entre 18 y 21	Media	0.7152	0.8182	2.7060	3.7739
	Cantidad	33	33	33	33
	Desviación estándar	0.27400	0.26717	2.28889	2.96031
	Mínimo	0.20	0.00	0.35	0.00
	Máximo	1.00	1.00	10.00	12.86
Entre 22 y 24	Media	0.8000	.8234	2.9901	4.3005
	Cantidad	32	32	32	32
	Desviación estándar	0.27357	0.29430	1.81876	2.41993
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	8.33	10.56
25 años y más	Media	0.6500	0.8091	2.3168	3.4018
	Cantidad	10	11	10	11
	Desviación estándar	0.39791	0.30481	1.78751	1.71726
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	4.84	5.88
Total	Media	0.7427	0.8191	2.7753	3.9417
	Cantidad	75	76	75	76
	Desviación estándar	0.29325	0.28047	2.02246	2.58125
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	10.00	12.86
	Varianza	0.086	0.079	4.090	6.663

Se puede notar, en la **Figura 7** y ***Figura 8***, que en los tres grupos de edad, el paradigma procedimental presenta medias más altas que el paradigma orientado a objetos.

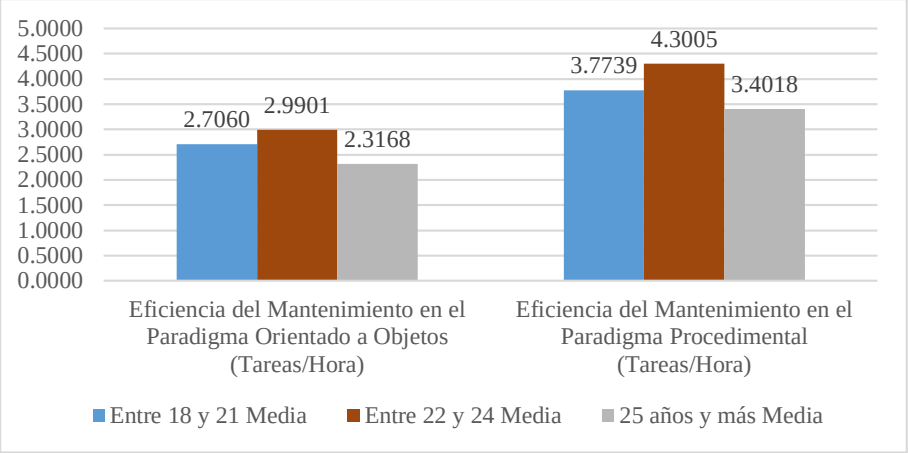

Figura 7. Medias agrupadas por grupo de edad para eficiencia del mantenimiento

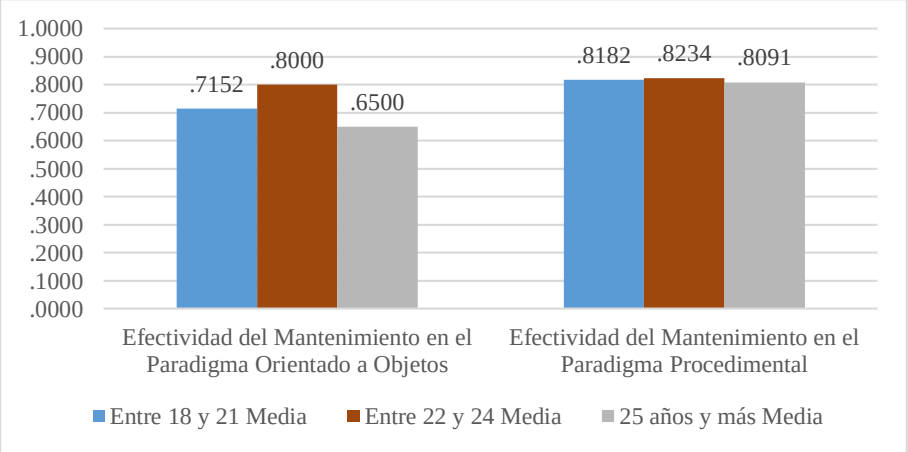

Figura 8. Medias agrupadas por grupo de edad para efectividad del mantenimiento

Análisis por Asignatura

En la ***Tabla 17*** se encuentran los estadísticos descriptivos distribuidos por las asignaturas: Programación Orientada a Objetos, Programación Avanzada e Inglés Técnico.

Tabla 17
Estadísticos descriptivos por asignatura

Materia que está cursando actualmente		Efectividad del Mantenimiento en el Paradigma Orientado a Objetos	Efectividad del Mantenimiento en el Paradigma Procedimental	Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)	Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora)
Programación Orientada a Objetos	Media	0.6500	0.7406	1.9161	2.9668
	Cantidad	32	32	32	32
	Desviación estándar	0.25400	0.30012	1.38193	2.53576
	Mínimo	0.20	0.00	0.35	0.00
	Máximo	1.00	1.00	6.98	12.86
	Varianza	0.065	0.090	1.910	6.430
Programación Avanzada	Media	0.8225	0.8788	3.4356	4.6378
	Cantidad	40	40	40	40
	Desviación estándar	0.30507	0.25290	2.11157	2.39712
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	10.00	12.00
	Varianza	0.093	0.064	4.459	5.746
Inglés Técnico	Media	0.6889	0.8667	3.3406	4.1112
	Cantidad	9	9	9	9
	Desviación estándar	0.30185	0.21794	3.46075	2.05648
	Mínimo	0.20	0.40	0.41	1.04
	Máximo	1.00	1.00	9.68	7.69
	Varianza	0.091	0.048	11.977	4.229
Total	Media	0.7395	0.8228	2.8247	3.9191
	Cantidad	81	81	81	81
	Desviación estándar	0.29397	0.27444	2.15826	2.51813
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	10.00	12.86
	Varianza	0.086	0.075	4.658	6.341

En la **Figura 9** y ***Figura 10*** se puede observar que las medias de efectividad y eficiencia del mantenimiento para el paradigma procedimental son superiores a las del paradigma orientado a objetos en todos los grupos, sin embargo, el grupo de Programación Avanzada presenta una menor diferencia entre las efectividades para ambos paradigmas.

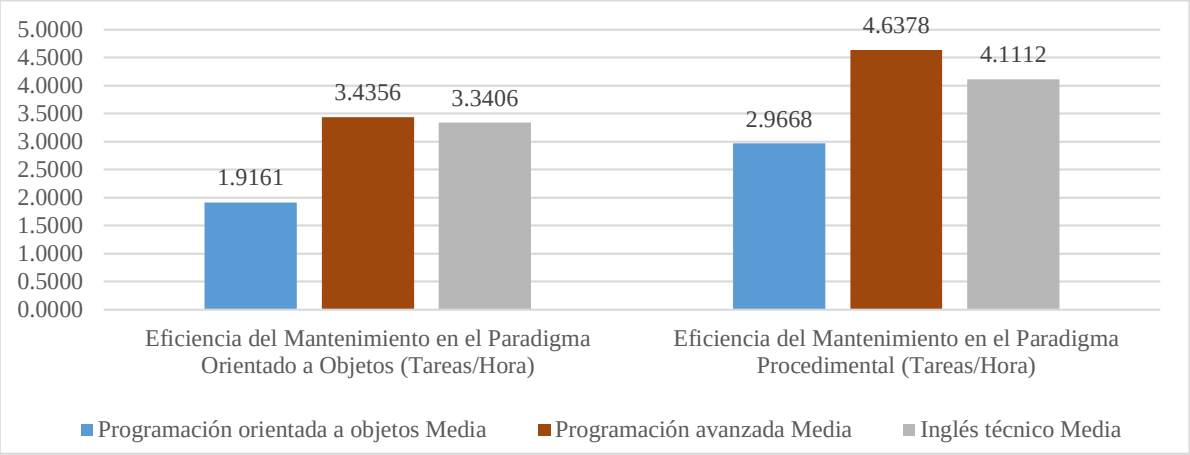

Figura 9. Medias agrupadas por asignatura para eficiencia del mantenimiento

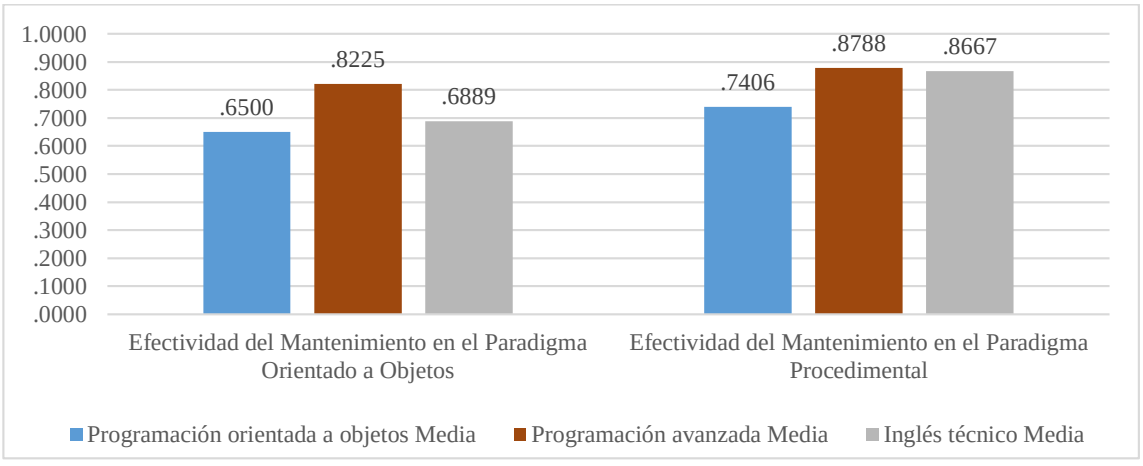

Figura 10. Medias agrupadas por asignatura para efectividad del mantenimiento

Análisis por Orden de Tratamiento

En la ***Tabla 18*** se encuentran los estadísticos descriptivos por orden de desarrollo del mantenimiento, primero procedimental y luego orientado a objetos o viceversa. Esta agrupación se realizó para comprobar el efecto del aprendizaje que pueda experimentar el sujeto al realizar los dos tratamientos.

En la **Figura 11** y ***Figura 12*** se puede notar el efecto del aprendizaje para los sujetos que realizaron primero el tratamiento orientado a objetos, teniendo un crecimiento en las medias de eficiencia y efectividad de su segundo tratamiento. Por otro lado, los sujetos que realizaron primero el tratamiento procedimental tuvieron un ligero decrecimiento en las medias para el siguiente tratamiento. Sin embargo, podemos notar que para los sujetos que desarrollaron el mantenimiento con el paradigma orientado a objetos como segundo tratamiento, con aprendizaje, sus promedios de efectividad 0.80 y eficiencia 3.28 son menores a los del paradigma procedimental 0.83 y 3.4.

Finalmente, las medias de efectividad 0.68 y eficiencia 2.4 de los sujetos que realizaron el experimento sin aprendizaje en el paradigma orientado a objetos, son menores a las que obtuvieron los sujetos que realizaron el experimento con el paradigma procedimental bajo las mismas condiciones 0.83 y 3.4.

Tabla 18
Estadísticos descriptivos por orden de mantenimiento

Orden en el que se desarrollaron las tareas		Efectividad del Mantenimiento en el Paradigma Orientado a Objetos	Efectividad del Mantenimiento en el Paradigma Procedimental	Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)	Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora)
Procedimental - Objetos	Media	0.8086	0.8341	3.2818	3.4569
	Cantidad	35	44	35	44
	Desviación estándar	0.24299	0.27168	1.97910	1.83038
	Mínimo	0.20	0.00	0.36	0.00
	Máximo	1.00	1.00	10.00	7.69
	Varianza	0.059	0.074	3.917	3.350
Objetos - Procedimental	Media	0.6870	0.8095	2.4770	4.4688
	Cantidad	46	37	46	37
	Desviación estándar	0.32014	0.28083	2.24401	3.08507
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	9.68	12.86
	Varianza	0.102	0.079	5.036	9.518
Total	Media	0.7395	0.8228	2.8247	3.9191
	Cantidad	81	81	81	81
	Desviación estándar	0.29397	0.27444	2.15826	2.51813
	Mínimo	0.00	0.00	0.00	0.00
	Máximo	1.00	1.00	10.00	12.86
	Varianza	0.086	0.075	4.658	6.341

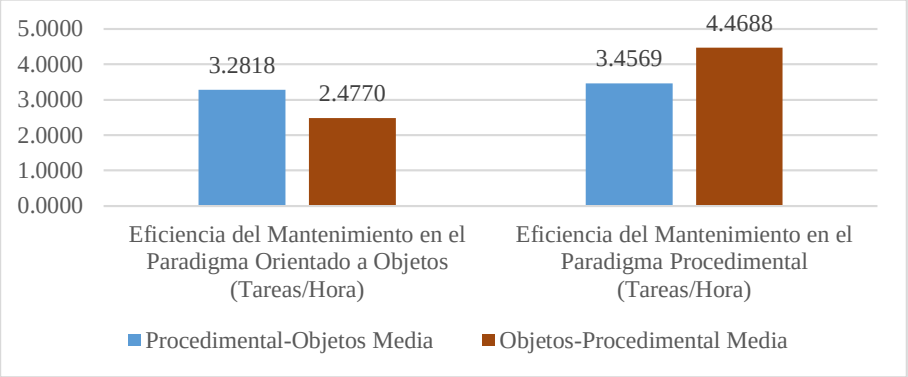

Figura 11. Medias agrupadas por orden de mantenimiento para eficiencia del mantenimiento

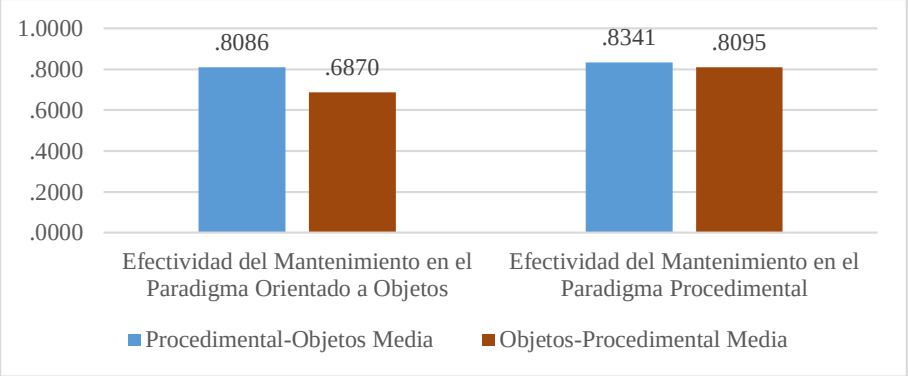

Figura 12. Medias agrupadas por orden de mantenimiento para efectividad del mantenimiento

El análisis de los estadísticos descriptivos muestra que, para cada caso, las medias de efectividad y eficiencia del mantenimiento para el paradigma procedimental son mayores que las del paradigma orientado a objetos, sin importar la asignatura, el género, la edad o el orden del mantenimiento, lo que indica la posibilidad de que la elección del paradigma de programación tenga influencia en la mantenibilidad de un proyecto de software, bajo el contexto especifico de este experimento.

4.2 Comprobación del tipo de distribución de los datos

Con el fin de determinar qué tipo de test estadístico aplicar sobre los datos obtenidos del experimento, se realizó una prueba de normalidad de Kolmogorov-Smirnov sobre las cuatro variables: efectividad y eficiencia del mantenimiento utilizando el paradigma procedimental y efectividad y eficiencia usando el paradigma orientado a objetos. Los resultados de la prueba de normalidad se presentan en la ***Tabla 19***.

Tabla 19

Prueba de normalidad de una muestra de Kolmogorov-Smirnov

		Efectividad del Mantenimiento en el Paradigma Procedimental	Efectividad del Mantenimiento en el Paradigma Orientado a Objetos	Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora)	Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)
Cantidad		81	81	81	81
Parámetros normales	Media	0.8228	0.7395	3.9191	2.8247
	Desviación estándar	0.27444	0.29397	2.51813	2.15826
Diferencias extremas	Absoluto	0.272	0.261	0.124	0.101
	Positivo	0.259	0.188	0.124	0.093
	Negativo	-0.272	-0.261	-0.060	-0.101
Kolmogorov-Smirnov Z		2.444	2.345	1.116	0.907
Nivel de significancia		0.000	0.000	0.166	0.383

Para las medidas de efectividad en ambos paradigmas se obtuvieron niveles de significancia menores a 0.05, esto permite determinar que los datos no siguen una distribución normal. Sin embargo, para ambas medidas de eficiencia se obtuvieron niveles de significancia mayores a 0.05, por consiguiente, estos datos siguen una distribución normal.

De acuerdo al tipo de distribución, se decidió utilizar los siguientes test estadísticos para contrastar las hipótesis:

- Para las medidas de efectividad del mantenimiento, y en vista que no siguieron una distribución normal, se utilizó el test no paramétrico de Wilcoxon.

- Para las medidas de eficiencia del mantenimiento, las cuales, sí siguieron una distribución normal, se utilizó la prueba de medias T de Student para muestras relacionadas.

4.3 Pruebas de hipótesis

4.3.1 Pruebas de hipótesis relacionadas con la eficiencia del mantenimiento

De acuerdo a los resultados obtenidos mediante la prueba T de Student para las medidas de eficiencia del mantenimiento, detalladas en la ***Tabla 20*** y ***Tabla 21***, es posible rechazar la hipótesis nula $H_{0,2}: \text{Mefic}_{\text{poo}} = \text{Mefic}_{\text{proc}}$, debido a que para un intervalo de confianza del 95% se obtuvo un p-valor menor a 0.05, además de aceptar la hipótesis alternativa $H_{1,2,3}: \text{Mefic}_{\text{poo}} < \text{Mefic}_{\text{proc}}$ ya que el valor del estadístico de contraste ***t*** es mayor que 0.

Tabla 20
Estadísticas de muestras emparejadas

	Media	Cantidad	Desviación estándar	Error estándar medio
Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora)	3.8951	72	2.58128	0.30421
Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)	2.7603	72	1.96497	0.23157

Tabla 21
Pruebas de muestras emparejadas

	Diferencias emparejadas					t	df	p-valor
	Media	Desviación estándar	Error estándar medio	Diferencia Inferior	Diferencia Superior			
Eficiencia del Mantenimiento en el Paradigma Procedimental (Tareas/Hora) - Eficiencia del Mantenimiento en el Paradigma Orientado a Objetos (Tareas/Hora)	1.13486	2.21360	.26088	.61469	1.65503	4.350	71	.000

4.3.2 Pruebas de hipótesis relacionadas con la efectividad del mantenimiento

En la ***Tabla 22*** se detallan los resultados de la prueba de Wilcoxon para las medidas de efectividad del mantenimiento donde se evidencia la posibilidad de rechazar la hipótesis nula $H_{0,1}: \text{Mefec}_{\text{poo}} = \text{Mefec}_{\text{proc}}$, para un intervalo de confianza del 95% ya que el nivel de significancia es menor que 0.05.

Tabla 22
Test estadístico de Wilcoxon para la efectividad del mantenimiento

	Efectividad del Mantenimiento en el Paradigma Orientado a Objetos - Efectividad del Mantenimiento en el Paradigma Procedimental
Z	-2.656[b]
Nivel de significancia	0.008

4.4 Resultados de las encuestas

4.4.1 Análisis de las encuestas pre y post experimento

Con el propósito de determinar cuál ha sido el impacto del entrenamiento previo al experimento, se ha calculado el valor promedio de los resultados de las preguntas tipo Likert de 5 puntos de las encuestas utilizando la misma distribución de los sujetos en grupos de la sección 4.1.

Se puede observar en la ***Figura 13*** que después del entrenamiento todos los grupos de estudio experimentaron crecimiento en sus niveles de conocimiento, indicando un impacto positivo del entrenamiento en los sujetos experimentales.

Además, con el propósito de evidenciar el cambio en el conocimiento de los sujetos experimentales, se realizó el cálculo de la diferencia entre los valores obtenidos antes y después del entrenamiento, detallado en la ***Figura 14***.

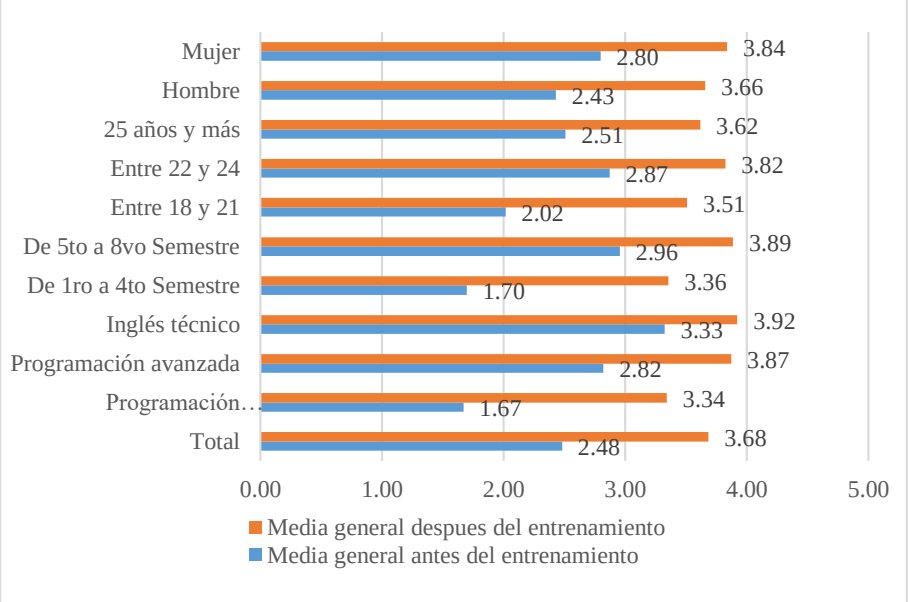

Figura 13. Media general de conocimientos sobre paradigmas de programación

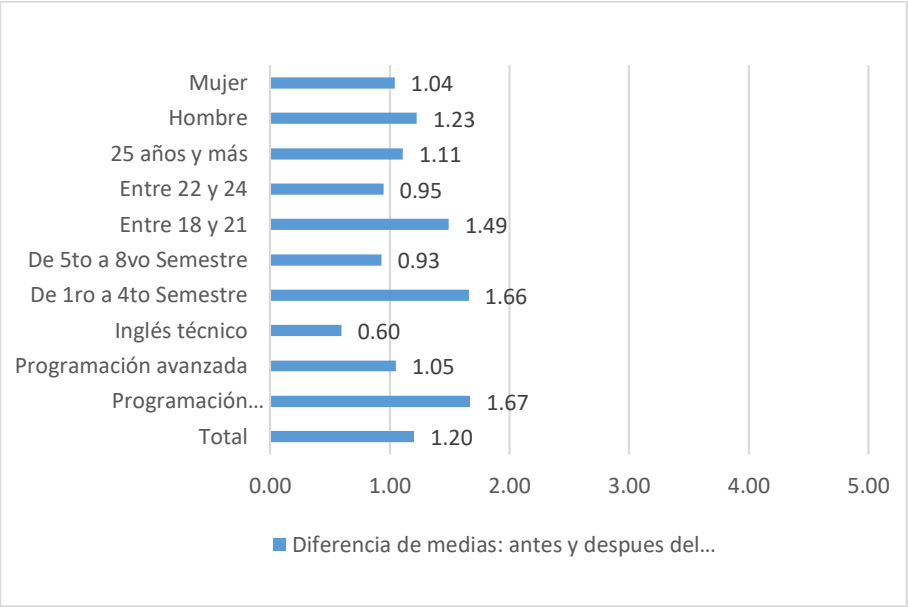

Figura 14. Diferencia de medias generales de conocimiento antes y después del entrenamiento

De los datos presentes en la ***Figura 13*** y ***Figura 14*** se puede observar que luego del entrenamiento, el grupo de hombres presentó una mayor diferencia de aprendizaje (1.23 puntos de crecimiento) respecto a las mujeres que es de 1.04, sin embargo, el grupo de mujeres tiene medias de conocimiento más altas que el grupo de hombres antes y después del entrenamiento, 2.80 y 3.84 respectivamente, (2.43 y 3.66).

Analizando los grupos de edad, los sujetos entre 18 y 21 años son los que presentan una mayor diferencia de aprendizaje: 1.49 puntos de crecimiento, seguidos por el grupo de mayores de 25 años con 1.11, finalmente los sujetos entre 22 y 24 años presentan una diferencia de crecimiento de 0.95.

Los sujetos de la asignatura de Programación Orientada a Objetos son los que presentan una mayor diferencia de conocimientos: 1.67 puntos, en segundo lugar, se encuentran los sujetos

de la asignatura de Programación Avanzada con una diferencia positiva luego del entrenamiento de 1.05. Finalmente, los sujetos de Ingles Técnico presentan un crecimiento de 0.60, sin embargo, este grupo cuenta con una mayor media de conocimientos previo al entrenamiento, de 3.33.

4.4.2 Análisis de la encuesta post experimento

El análisis de la encuesta post experimento se realiza con el propósito de identificar posibles diferencias entre la dificultad de las tareas de mantenimiento para ambos paradigmas y validar si existió algún sesgo durante la ejecución del experimento que pudiera afectar a la validez de las conclusiones.

Se realiza una prueba de hipótesis comparando los resultados obtenidos para ambos paradigmas de programación, donde se busca aceptar o negar la hipótesis nula para cada pregunta, $H_o: Proc = Obj$ (No existe diferencia entre los resultados del paradigma procedimental y el paradigma orientado a objetos.)

Previo a seleccionar el test estadístico se realizó la prueba de normalidad de Kolmogorov-Smirnov para determinar si los datos presentaban una distribución normal. Los resultados detallados en el ***Anexo G*** permiten concluir que los datos no presentan dicha distribución, por consiguiente, se utilizó el test no paramétrico de Wilcoxon para muestras relacionadas. Los resultados de las nueve preguntas se encuentran agrupados por asignatura, y su rango de calificación es entre 1 y 5.

Pregunta 1: Acerca del nivel de entendimiento del código del backend entregado

En esta pregunta, el valor de 1 representa mínimo entendimiento del código por parte del encuestado, mientras que el valor de 5 indica que el sujeto tuvo el mayor nivel de entendimiento del código del backend.

En la ***Figura 15*** se puede observar que los sujetos de las asignaturas de Inglés Técnico y Programación Avanzada presentaron un mayor nivel de entendimiento para el paradigma procedimental, no obstante, los sujetos de Programación Orientada a Objetos presentaron un nivel de entendimiento idéntico para ambos paradigmas.

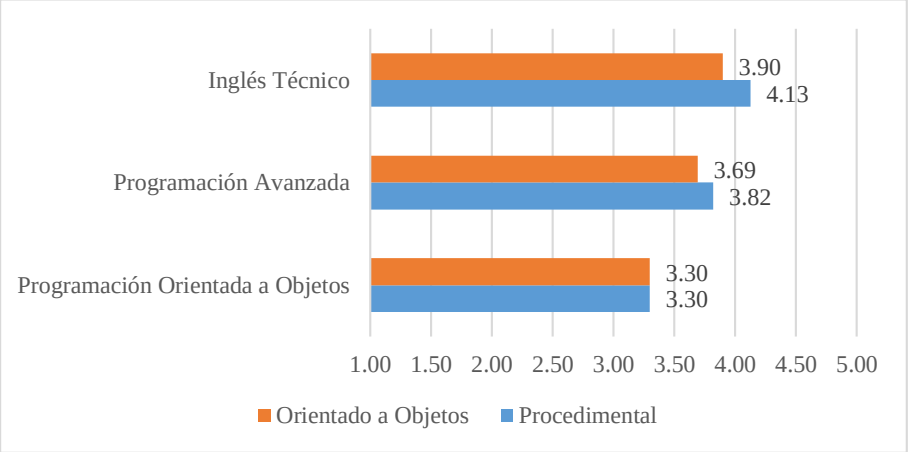

Figura 15. Nivel de entendimiento del código del backend entregado

Pregunta 2: Acerca de la dificultad para encontrar defectos relacionados a las tareas de mantenimiento

En esta pregunta el valor de 1 representa que el sujeto tuvo gran dificultad para encontrar los defectos descritos en las tareas de mantenimiento, mientras que el valor de 5 indica que el sujeto no presentó ninguna dificultad.

En la ***Figura 16*** se observa que los sujetos de la asignatura de Inglés Técnico presentaron menor dificultad para encontrar defectos relacionados con las tareas de mantenimiento al momento de realizar el experimento en el paradigma orientado a objetos, por otro lado, los grupos de las asignaturas de Programación Avanzada y Programación Orientada a Objetos presentaron un menor nivel de dificultad con el paradigma procedimental.

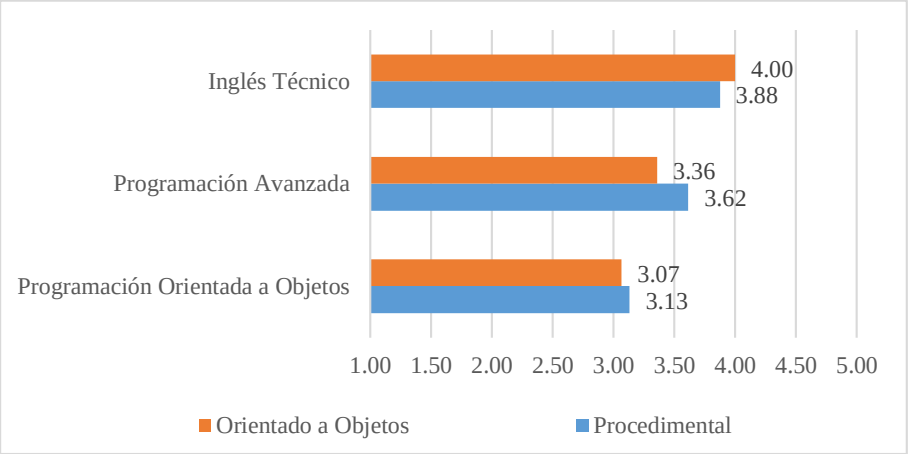

Figura 16. Dificultad para encontrar defectos relacionados a las tareas de mantenimiento

Pregunta 3: Acerca de la percepción de que tan descriptivos fue el código

En esta pregunta el valor de 1 representa que los elementos del código no fueron nada descriptivos e incomprensibles, mientras que el valor de 5 indica que el código fue sencillo de entender por parte del sujeto.

En la ***Figura 17*** se detalla que los sujetos de la asignatura de Inglés Técnico encontraron los elementos del código como: nombre de variables, funciones, clases y archivos, del backend orientado a objetos más descriptivos que los elementos del backend procedimental, mientras que, los sujetos de las asignaturas de Programación Orientada a Objetos y Programación Avanzada encontraron los elementos del backend procedimental más descriptivos que los elementos del backend orientado a objetos.

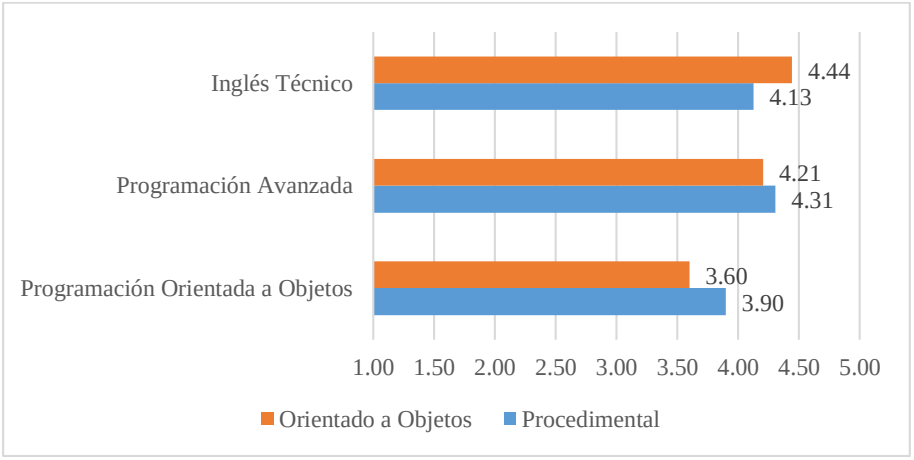

Figura 17. Percepción de que tan descriptivo fue el código

Pregunta 4: Acerca de la percepción de la complejidad en utilizar las características del framework

En esta pregunta el valor 1 representa que el sujeto tuvo gran dificultad para utilizar y comprender el framework de desarrollo, mientras que el valor 5 indica que para el sujeto le resultó sencillo el manejo del framework.

En la ***Figura 18*** se puede observar que, de manera general, los sujetos experimentales consideran que las características del framework Spring Boot fueron más sencillas de utilizar durante la ejecución del experimento que las características de Flask.

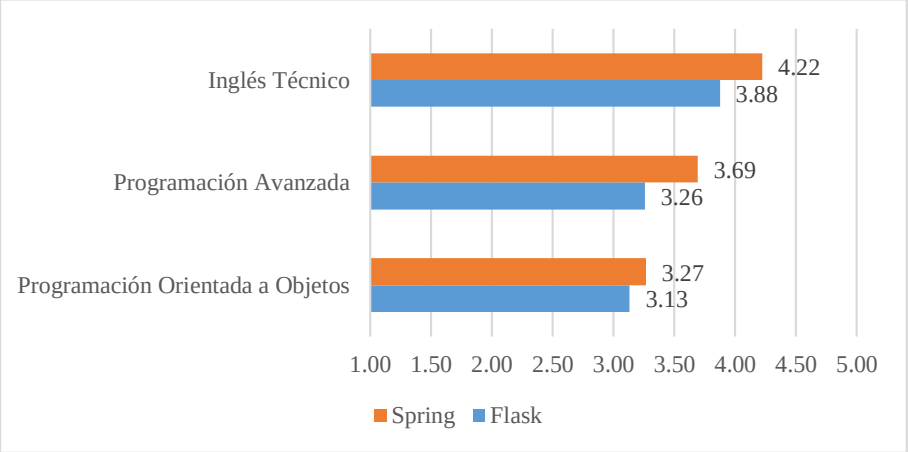

Figura 18. Percepción de la complejidad para utilizar las características del framework

Para las siguientes preguntas de la encuesta, el valor 1 representa que al sujeto le resultó completamente difícil realizar la tarea de mantenimiento, mientras que el valor 5 indica que el sujeto no tuvo dificultades en entender, localizar y desarrollar la tarea de mantenimiento.

Pregunta 5: Acerca de la percepción de la dificultad de la tarea de mantenimiento MC_I_01 Ingrediente vacío

En la ***Figura 19*** se muestra que los sujetos de las asignaturas de Inglés Técnico y Programación Orientada a Objetos tuvieron menores dificultades con la tarea de mantenimiento MC_I_01 utilizando el paradigma de programación procedimental. Por otro lado, los estudiantes de la asignatura de Programación Avanzada tuvieron menor dificultad con el paradigma orientado a objetos.

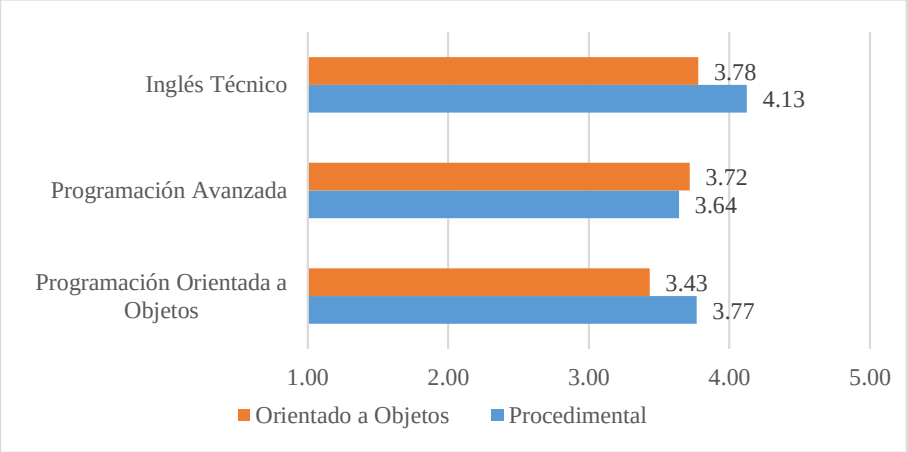

Figura 19. Percepción de dificultad de la tarea MC_I_01

Pregunta 6: Acerca de la percepción de la dificultad de la tarea de mantenimiento MC_O_01 Orden vacía

En la ***Figura 20*** se puede observar que, de manera general, los tres grupos de sujetos experimentales percibieron que el desarrollo de la tarea MC_O_01 fue más sencilla utilizando el paradigma procedimental.

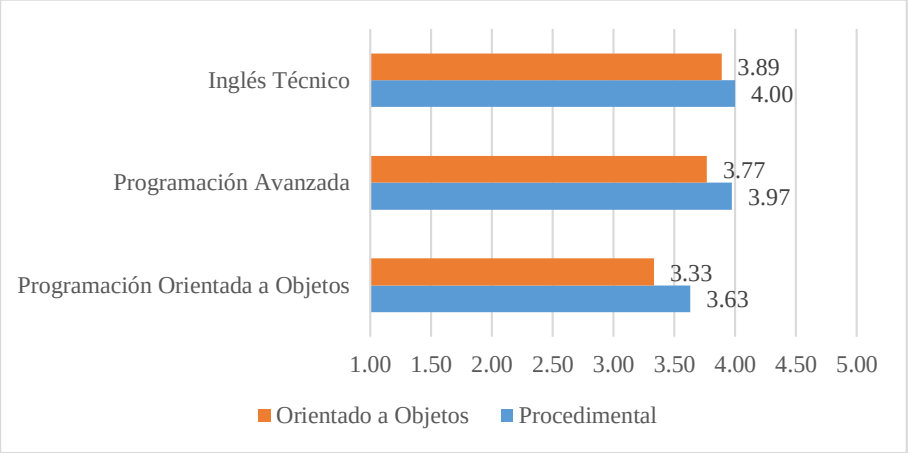

Figura 20. Percepción de dificultad de la tarea MC_O_01

Pregunta 7: Acerca de la percepción de la dificultad de la tarea de mantenimiento MC_O_02 Orden sin datos del cliente

En la ***Figura 21*** se puede observar que los sujetos experimentales de la asignatura de Inglés Técnico presentaron menores dificultades en esta tarea de mantenimiento utilizando el paradigma orientado a objetos, mientras que, los grupos de las asignaturas de Programación Avanzada y Programación Orientada a Objetos percibieron menor dificultad al desarrollar esta tarea utilizando el paradigma procedimental.

Pregunta 8: Acerca de la percepción de la dificultad de la tarea de mantenimiento MA_S_01 Lista tamaño vacía

En la ***Figura 22*** se observa que los sujetos experimentales de Inglés Técnico reportan menor dificultad para el desarrollo de esta tarea utilizando el paradigma orientado a objetos, mientras que el resto reporta menor dificultad al desarrollar esta tarea de mantenimiento utilizando el paradigma procedimental.

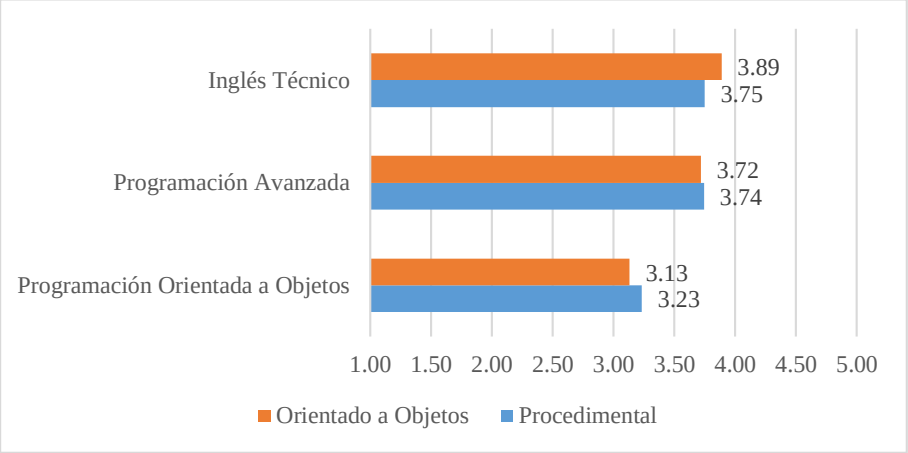

Figura 21. Percepción de dificultad de la tarea MC_O_02

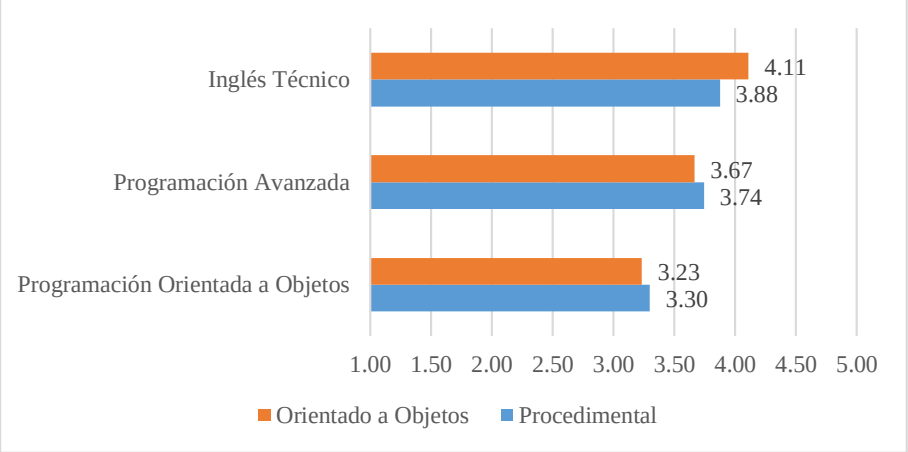

Figura 22. Percepción de dificultad de la tarea MA_S_01

Pregunta 9: Acerca de la percepción de la dificultad de la tarea de mantenimiento MC_O_03 Orden precio erróneo

En la ***Figura 23*** observamos que los sujetos experimentales de Inglés Técnico, experimentaron menores dificultades al desarrollar esta tarea de mantenimiento utilizando el paradigma orientado a objetos, mientras que el resto de sujetos, presento menores dificultades con el paradigma procedimental.

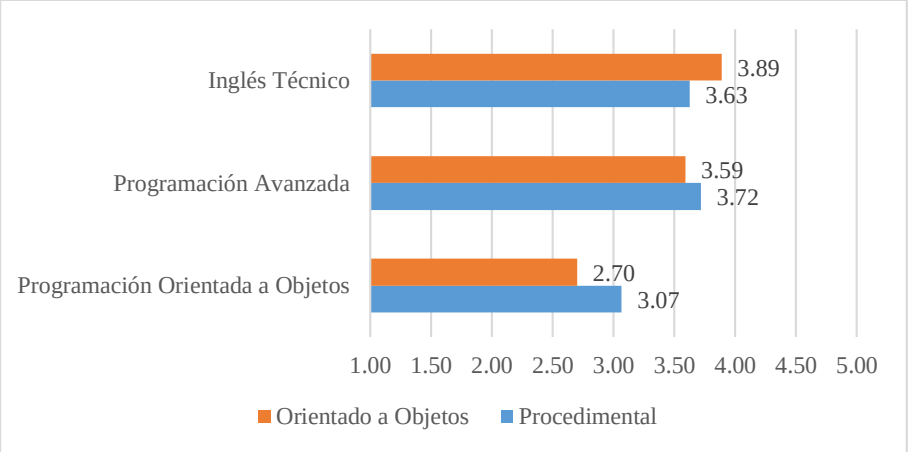

Figura 23. Percepción de dificultad de la tarea MC_O_03

Al desarrollar la prueba de hipótesis para cada pregunta de la encuesta post experimento, detalladas en el ***Anexo H***, se obtuvo un p-valor mayor a 0.05 en 5 de 9 casos, que para un nivel de confianza del 95% no permite rechazar la hipótesis nula H_o.

4.5 Amenazas a la validez

4.5.1 Validez externa

La validez está relacionada a la posibilidad de generalizar los resultados. Existen varias amenazas, mismas que se detallan a continuación:

- El uso de estudiantes como sujetos experimentales, que puedan no ser profesionales representativos de la industria de software. Sin embargo, las tareas de mantenimiento no requieren experiencia profesional.
- Los objetos experimentales podrían no ser representativos de un contexto real, por su simplicidad.

4.5.2 Validez interna

Las amenazas a la validez interna fueron mitigadas, dentro de lo posible, mediante el diseño del experimento. Se utilizó grupos de sujetos balanceados y seleccionados aleatoriamente. El cuestionario post experimento revela que no existió una diferencia en la dificultad de comprensión del código percibida por los sujetos entre los dos backends, ver sección 4.4.2. Además, permite confirmar que los sujetos percibieron igual de satisfactorio al entrenamiento en ambos paradigmas de programación (véase ***Anexo I***).

4.5.3 Validez de constructo

De acuerdo con (Nguyen-Duc, 2017), es posible medir la mantenibilidad de un proyecto software indirectamente, a través del rendimiento de los sujetos al realizar actividades de mantenimiento. Se utilizaron métricas estándares adaptadas al presente estudio para medir

eficiencia y efectividad de los sujetos al realizar las actividades de mantenimiento (Genero et al., 2014). Los cuestionarios presentan preguntas y escalas estándar.

4.5.4 Validez de la conclusión

Se utilizó los test estadísticos apropiados de acuerdo a la distribución de los datos para corroborar las conclusiones del experimento: T de Student para datos con distribución normal, y el test no paramétrico de Wilcoxon para datos que no siguen dicha distribución.

CAPÍTULO V

CONCLUSIONES, RECOMENDACIONES Y LÍNEAS DE TRABAJO FUTURO

5.1 Conclusiones

Se corrobora una de las líneas de investigación del trabajo de Nguyen-Duc, (2017), relacionado con la medición indirecta de la mantenibilidad de sistema. En la presente investigación fue posible encontrar diferencias en eficiencia y efectividad de los desarrolladores al realizar tareas de mantenimiento respecto al paradigma de programación utilizado.

En su mayoría, no se percibió diferencias estadísticamente significativas entre la dificultad de las tareas de mantenimiento, según los resultados del cuestionario post experimento, sección 4.4.2. Por lo cual, se rechaza la opción de sesgos originados por material de entrenamiento y los objetos experimentales (Backend procedimental y Backend orientado a objetos).

A pesar de que el entrenamiento previo al experimento fue impartido por dos instructores diferentes para cada paradigma, los sujetos no reportaron diferencias estadísticamente significativas en su nivel de satisfacción. De acuerdo con la prueba no paramétrica de Wilcoxon detallada en el ***Anexo H***.

Un objeto experimental sin un nivel de complejidad alto nos permitió que los sujetos tengan un buen nivel de comprensión del código y así evitar las amenazas a la validez relacionadas con la fatiga al momento de realizar el mantenimiento.

La efectividad del paradigma orientado a objetos fue menor que la del paradigma procedimental. Existe una diferencia estadísticamente significativa de 8.33 puntos porcentuales a favor del paradigma procedimental, demostrado a través de la prueba no paramétrica de Wilcoxon, que se utiliza para distribuciones diferentes de la normal.

La eficiencia del paradigma orientado a objetos fue menor que la del paradigma procedimental por aproximadamente una tarea/hora estadísticamente significativa, lo cual es demostrado mediante la prueba T de Student para muestras relacionadas, que se utiliza para distribuciones normales.

La efectividad y la eficiencia del paradigma orientado a objetos en promedio son mayores, cuando los sujetos desarrollan este tratamiento luego de haber utilizado el paradigma procedimental para realizar el mantenimiento respectivo (con aprendizaje). Sin embargo, estos valores de efectividad y eficiencia son menores, en 2.6 puntos porcentuales y 0.2 tareas/horas respectivamente, que el de los sujetos al realizar el mantenimiento con el paradigma procedimental en primer lugar (sin aprendizaje).

El nivel de efectividad del paradigma orientado a objetos de las mujeres es, en promedio, 9 puntos porcentuales mayor que el de los hombres. El nivel de eficiencia del paradigma orientado a objetos de las mujeres es, en promedio, alrededor de media tarea/hora más que el promedio de los hombres.

El nivel de efectividad del paradigma procedimental de las mujeres es, en promedio, 9 puntos porcentuales mayor que el de los hombres. El nivel de eficiencia del paradigma procedimental de las mujeres es, en promedio, 0.6 tareas/hora más que el de los hombres.

Todos los grupos de edad investigados presentan, en promedio, una mayor efectividad y eficiencia de mantenimiento utilizando el paradigma procedimental.

Los promedios más altos de efectividad y eficiencia, en ambos paradigmas de programación, de acuerdo a la asignatura, los obtuvieron los sujetos que se encuentran actualmente cursando Programación Avanzada (6to nivel de formación de pregrado en ingeniería de sistemas).

Las mujeres presentan un promedio de conocimiento percibido inicial y final, mayor al de los hombres, de acuerdo con los resultados del cuestionario pre entrenamiento y post entrenamiento. Sin embargo, el grupo de hombres presenta una mayor diferencia entre sus valores inicial y final, es decir los hombres percibieron un mayor aprendizaje.

Los estudiantes de Programación Orientada a Objetos (tercer semestre) en promedio duplican su conocimiento percibido después del entrenamiento, llegando a tener un valor casi igual al conocimiento percibido inicial de los estudiantes de Inglés Técnico (octavo semestre).

5.2 Recomendaciones

Antes de desarrollar cualquier tipo de entrenamiento, es recomendable obtener una medida base del conocimiento percibido de los sujetos, para comprobar si el entrenamiento fue efectivo denotando aprendizaje y satisfacción, adicionalmente obtener una visión global de sus conocimientos previos.

Al desarrollar un entrenamiento en temas diferentes e impartirlo diferentes instructores, que podría ocasionar un sesgo en el experimento dependiendo del nivel de beneficio de los sujetos en cada tema, se recomienda que los instructores trabajen en conjunto para el desarrollo del material y el contenido del entrenamiento.

Se recomienda incorporar un estudio de los diferentes paradigmas de programación en los sílabos de las asignaturas que pertenecen a las áreas de conocimiento: Programación e Ingeniería de Software. En especial el paradigma procedimental que, de acuerdo al contexto de la presente investigación, permite al futuro profesional desarrollar los procesos del SDLC con mayor eficiencia y efectividad.

En la actualidad, los primeros cursos de programación están enfocados al paradigma puramente estructurado (no sujeto de estudio en la presente investigación), que podría derivar inmediatamente en el paradigma procedimental, para luego continuar con el paradigma orientado a objetos.

5.3 Líneas de trabajo futuro

Es necesario replicar el experimento realizado tomando medidas que reduzcan las amenazas a la validez externa, es decir:

- Realizar el experimento en un contexto empresarial.
- Trabajar con sujetos experimentales que posean una mayor experiencia profesional.
- Utilizar objetos experimentales con un mayor nivel de complejidad, desarrollados en diferentes herramientas a las empleadas en la presente investigación.
- Utilizar una muestra de sujetos experimentales de mayor tamaño.

Finalmente, se recomienda realizar comparaciones entre un conjunto más grande de paradigmas de programación, tomando en cuenta otras escuelas de pensamiento como la programación declarativa, para así obtener información más amplia sobre el efecto de diferentes paradigmas en un proyecto de desarrollo de software.

BIBLIOGRAFÍA

Ahmad, A., & Talha, M. (2002). A measurement based comparative evaluation of effectiveness of object-oriented versus conventional procedural programming techniques and languages. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2002-Janua*, 517–526. https://doi.org/10.1109/APSEC.2002.1183072

Asagba, P., & Ogheneovo, E. (2008). A Comparative Analysis of Structured and Object-Oriented Programming Methods. *Journal of Applied Sciences and Environmental Management*, *11*(4). https://doi.org/10.4314/jasem.v11i4.55190

Banker, R. D. (2014). *Software complexity and maintainability*. (June). https://doi.org/10.1145/75034.75056

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. *Encyclopedia of Software Engineering*, *2*, 528–532. https://doi.org/10.1.1.104.8626

Boticki, I., Katic, M., & Martin, S. (2013). Exploring the educational benefits of introducing aspect-oriented programming into a programming course. *IEEE Transactions on Education*, *56*(2), 217–226. https://doi.org/10.1109/TE.2012.2209119

Calderón Pascual, M. V. (2016). *Medidas de Tendencia Central*. 23.

Chow, T., & Cao, D. B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, *81*(6), 961–971. https://doi.org/10.1016/j.jss.2007.08.020

Cobo, E., Muñoz, P., González, J. A., Bigorra, J., Corchero, C., Miras, F., … Videla, S. (2007). Prueba de significación y contraste de hipótesis. *Bioestadística Para No Estadísticos*, 157–

192. https://doi.org/10.1016/b978-84-458-1782-7.50007-9

Cornell University. (n.d.). Abstraction. Retrieved from CS211 course website: http://www.cs.cornell.edu/courses/cs211/2006sp/Lectures/L08-Abstraction/08_abstraction.html

Crouch, B. S., & Lead, S. T. (2019). *Developing maintainable software*. 1. Retrieved from https://software.ac.uk/resources/guides/developing-maintainable-software

Departamento de Ciencias de la Computación. (2019). Malla Curricular Ingeniería en Sistemas e Informática.

Di Francesco, P., Malavolta, I., & Lago, P. (2017). Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 21–30. https://doi.org/10.1109/ICSA.2017.24

Ferrán Aranaz, M. (2001). *SPSS para Windows : análisis estadístico*. 421.

Frame, S., & Coffey, J. W. (2014). *A Comparison of Functional and Imperative Programming Techniques for Mathematical Software Development*. *12*(2), 49–53.

Frederick P. Brooks, J. (1986). No Silver Bullet —Essence and Accident in Software Engineering. *University of North Carolina at Chapel Hill*. https://doi.org/10.13225/j.cnki.jccs.2017.1345

Genero, M., Cruz-Lemus, J., & Piattini, M. (2014). *Métodos de investigación en ingeniería del software*.

Gupta, V., Ganeshan, N., & Singhal, T. K. (2015). Determining the root causes of various software

bugs throughsoftware metrics. *2015 International Conference on Computing for Sustainable Global Development, INDIACom 2015*, 1211–1215.

Henry, S. M., & Humphrey, M. (2002). *A controlled experiment to evaluate maintainability of object-oriented software*. 258–265. https://doi.org/10.1109/icsm.1990.131370

Ieee. (1990). IEEE Standard Glossary of Software Engineering Terminology. *Office*, *121990*(1), 1. https://doi.org/10.1109/IEEESTD.1990.101064

Instituto Tecnológico de Celaya. (n.d.). *Paradigmas de programación*. Retrieved from http://www.iqcelaya.itc.mx/~vicente/Programacion/Paradigmas.pdf

ISO. (2019). ISO 25000.

Jedlitschka, A., & Tn, A. (2005). Reporting Guidelines for Controlled Experiments in Software Engineering Dietmar Pfahl. *Evaluation*, 95–104.

Kikuno, T. (2005). *Why Do Software Projects Fail ? Reasons and a Solution Using a Bayesian Classifier to Predict Potential Risk*. (1), 3.

Kitchenham, B., & Charters, S. (2007). issue: EBSE 2007-001. *Technical Report*, *2*(3).

Kitchenham, Barbara, & Pfleeger, L. S. (1996). Software quality: The Elusive Target. *IEEE Software*, 12–21. https://doi.org/http://doi.ieeecomputersociety.org/10.1109/52.476281

Klein, B. (2016). History of Python. *Python Course*, 1–3. Retrieved from https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/10471/461.pdf?sequence=3%0Ahttp://www.python-course.eu/python3_history_and_philosophy.php

Krishnamurthi, S., & Fisler, K. (2019). Programming Paradigms and Beyond. *The Cambridge*

Handbook of Computing Education Research, 377–413. https://doi.org/10.1017/9781108654555.014

Laine, M., Shestakov, D., Litvinova, E., & Vuorimaa, P. (2011). Toward unified web application development. *IT Professional*, *13*(5), 30–36. https://doi.org/10.1109/MITP.2011.55

Lehtinen, T. O. A., Mäntylä, M. V., Vanhanen, J., Itkonen, J., & Lassenius, C. (2014). Perceived causes of software project failures - An analysis of their relationships. *Information and Software Technology*, *56*(6), 623–643. https://doi.org/10.1016/j.infsof.2014.01.015

Lu, X., Liu, H., & Ye, W. (2010). *Analysis Failure Factors for Small & Medium Software Projects Based on PLS Method*.

Lynch, J. (2018). *Report Project Resolution Benchmark for IBEX Financial Corp*.

Maleki, S., Fu, C., Banotra, A., & Zong, Z. (2017). *This paper appears in the 2017 IGSC Workshop on Sustainability in Multi/Many-Core Systems Understanding the Impact of Object Oriented Programming and Design Patterns on Energy Efficiency*. 0–5.

Martin Host, Bjorn Regnell, C. W. (2000). Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessmen. *Empirical Software Engineering: An International Journal*, *5*(3).

Martínez, S. (2013). Por qué pagar un mantenimiento software de ERP.

McCall, J., Richards, P., & Walters, G. (1977). *Factors un software quality*. Roma.

Mueller, J. (2015). Embracing the Four Python Programming Styles. Retrieved from https://blog.newrelic.com/engineering/python-programming-styles/

Nguyen-Duc, A. (2017). The Impact of Software Complexity on Cost and Quality - A Comparative Analysis Between Open Source and Proprietary Software. *International Journal of Software Engineering & Applications*, *8*(2), 17–31. https://doi.org/10.5121/ijsea.2017.8202

Oracle. (2018). *Java Programming Language*. Retrieved from https://docs.oracle.com/javase/7/docs/technotes/guides/language/index.html

Otero, M. C., & Dolado, J. J. (2000). Diseño Experimental en Ingeniería del Software. In *Medición para la Gestión en la Ingeniería del Software* (pp. 51–72).

Pallets Team. (2010). *Foreword - Flask Documentation*. Retrieved from http://flask.pocoo.org/docs/1.0/foreword/

Phillip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch, Andy Wilkinson, Marcel Overdijk, Christian Dupuis, Sébastien Deleuze, Michael Simons, Vedran Pavić, Jay Bryant, M. B. (2018). Introducing Spring Boot. Retrieved from https://docs.spring.io/spring-boot/docs/current/reference/html/

Pivotal Software. (2019). Spring. Retrieved from https://spring.io/

Pressman, R. S. (2010). *Ingeniería del software*.

Ramirez, J., Reyes, C., & Gil, G. (2014). *Métricas de Código fuente y Evolución de F / OSS : un estudio exploratorio*. (September 2013).

Rodríguez, M., Pedreira, Ó., & Fernández, C. M. (2015). Certificación de la Mantenibilidad del Producto Software: Un Caso Práctico. *Revista Latinoamericana de Ingenieria de Software*, *3*(3), 127. https://doi.org/10.18294/relais.2015.127-134

Ruiz, J. Z. (2004). ¿ Por Qué Fracasan los Proyectos de Software ? Un Enfoque Organizacional. *Congreso Nacional de Software Libre*, (2), 20–42. https://doi.org/10.13140/RG.2.1.4741.3206

Saraiva, J. (2013). A roadmap for software maintainability measurement. *Proceedings - International Conference on Software Engineering*, 1453–1455. https://doi.org/10.1109/ICSE.2013.6606742

Schildt. (2014). *Java_ The Complete Reference_ Comprehensive Coverage of the Java Language*.

Shadish, W., Cook, T., Campbell, T. (2005). Experiments and generalized causal inference. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*, *100*(470), 1–81. https://doi.org/10.1198/jasa.2005.s22

Sicilia, M.-A. (2008). *Aspectos que influyen en la Mantenibilidad*. *12*(1990), 1–2.

Sicilia, M. (2012). *Métricas del Mantenimiento de Software*. Texas.

Simmonds, D. M. (2012). *Paradigm Evolution*. (June), 93–95.

Sommerville, I. (2009). *Sommerville-Software Engineering*.

Stack Overflow. (2019). Stack Overflow Developer Survey 2019. Retrieved from https://insights.stackoverflow.com/survey/2019/#technology

Torgersson, O. (1996). A note on declarative programming paradigms and the future of definitional programming. *Das Winteroete*, *96*(1996), 13.

Tošić, Milena Vujošević, J. & D. (2008). The role of programming paradigms in the first programming courses. *The teaching of mathematics 2008, Vol. XI, 2, Pp. 63–83*, *XI*, 63–83.

Universidad de Valencia. (2015). *Mantenimiento* (p. 12). p. 12. Universidad de Valencia.

Urdhwareshe, A. (2016). *Object-Oriented Programming and its Concepts*. *26*(1), 1–6.

Valdecantos, H. A., Tarrit, K., Mirakhorli, M., & Coplien, J. O. (2017). An Empirical Study on Code Comprehension: Data Context Interaction Compared to Classical Object Oriented. *IEEE International Conference on Program Comprehension*, 275–285. https://doi.org/10.1109/ICPC.2017.23

Valenciano López, J. (2015). *Auditoría mantenibilidad aplicaciones según la ISO/IEC 25000*.

Van Roy, P. (2009). Programming Paradigms for Dummies: What Every Programmer Should Know. *New Computational Paradigms for Computer Music*, 9–47. Retrieved from http://www.dmi.unict.it/~barba/PROG-LANG/PROGRAMMI-TESTI/READING-MATERIAL/VanRoyChapter.pdf

Wang, F., Ai, J., & Wang, J. (2017). Empirical Study on the Correlation between Software Structural Modifications and Its Fault-Proneness. *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017*, 634–635. https://doi.org/10.1109/QRS-C.2017.125

Whitten, J. L., Bentley, L. D., & Dittman, K. C. (2004). *Systems Analysis and Design Methods*.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore, C. L. (1999). Comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, *11*(3), 255–282. https://doi.org/10.1016/S0953-5438(98)00029-0

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/2601248.2601268