UNIVERSIDAD TÉCNICA DE BABAHOYO FACULTAD DE ADMINISTRACIÓN, FINANZAS E INFORMÁTICA

PROCESO DE TITULACIÓN

PERIODO DICIEMBRE 2022 - MAYO 2023

EXAMEN COMPLEXIVO DE GRADO O DE FIN DE CARRERA

PRUEBA PRÁCTICA

PREVIO A LA OBTENCIÓN DEL TITULO DE INGENIERÍA EN SISTEMAS

TEMA:

ESTUDIO COMPARATIVO ENTRE LA PROGRAMACIÓN ORIENTADA A OBJETOS Y LA PROGRAMACIÓN ORIENTADA A ASPECTOS.

EGRESADO:

DARWIN PATRICIO MOROCHO TENELEMA

TUTOR:

ING. CARLOS ALFREDO CEVALLOS MONAR

AÑO:

2023

INTRODUCCIÓN

La programación es un campo en constante evolución, y uno de los debates más interesantes dentro de ella es el de la Programación Orientada a Objetos (POO) versus la Programación Orientada a Aspectos (POA). Ambos enfoques tienen sus ventajas y desventajas, y elegir entre ellos puede marcar una gran diferencia en el desarrollo de software.

El propósito de esta investigación es realizar un estudio comparativo entre la POO y la POA, con el objetivo de explorar las fortalezas y debilidades de cada enfoque y determinar cuál es más adecuado para diferentes tipos de proyectos de software. Para ello, se analizarán las características, ventajas y desventajas de cada enfoque, se explicará cómo se implementan en el código y se compararán sus resultados en términos de rendimiento, escalabilidad, mantenibilidad y otras métricas relevantes.

Para llevar a cabo este caso de estudio, se utilizarán diferentes tipos y métodos de investigación. Para conceptualizar las principales características de ambos paradigmas de programación se emplearon el tipo de investigación bibliográfica, para analizar, estudiar y experimentar las características principales y diseñar una comparación técnica se empleó el método comparativo.

La investigación se basará en los conocimientos adquiridos sobre programación, específicamente en torno a la POO y la POA. Además, se integrarán los conocimientos adquiridos a través de la revisión de la literatura y el análisis de casos de estudio.

Al final de esta investigación, se presentará una síntesis de los resultados obtenidos, se compararán los pros y los contras de cada enfoque y se ofrecerán recomendaciones sobre cuál es más adecuado para diferentes tipos de proyectos. También

se presentarán las preguntas de reflexión que surgieron a lo largo de la investigación, con el objetivo de fomentar la reflexión y el debate sobre el tema.

El objetivo general de esta investigación es realizar un estudio comparativo entre la Programación Orientada a Objetos (POO) y la Programación Orientada a Aspectos (POA), con el fin de determinar cuál es más adecuada para diferentes tipos de proyectos de software, mediante el análisis y comparación de sus características, ventajas y desventajas, su implementación en el código y su rendimiento en términos de escalabilidad, mantenibilidad y otras métricas relevantes.

La investigación del estudio comparativo entre la POO y la POA tiene una clara relación con la línea de investigación "Sistemas de información y comunicación, emprendimiento e innovación" y la sub línea específica de "Redes y tecnologías inteligentes de software y hardware", ya que puede contribuir a la creación de soluciones de software más inteligentes y eficientes y al desarrollo de emprendimientos y proyectos de innovación exitosos.

DESARROLLO

La programación es un área fundamental en el desarrollo de soluciones de software. La Programación Orientada a Objeto (POO) ha sido uno de los paradigmas más utilizados en la programación, pero en los últimos años la Programación Orientada a Aspectos (POA), ha surgido como una alternativa interesante que permite separar la lógica de negocio de los aspectos transversales de un sistema, facilitando la reutilización y modularidad del código.

Ante este panorama, surge la necesidad de realizar una investigación comparativa entre ambas metodologías para determinar cuál es más adecuada para diferentes tipos de proyectos de software. En este sentido, esta investigación es factible, ya que existen múltiples proyectos en la industria que utilizan ambos paradigmas, por lo que se pueden obtener muestras representativas de aplicaciones implementadas en ambos enfoques para realizar una comparación rigurosa y objetiva.

Además, los resultados de esta investigación pueden ser de gran utilidad para la industria del software y para los profesionales del área de la programación, ya que les permitirán tomar decisiones informadas sobre cuál enfoque es el más adecuado para sus proyectos específicos. Esto, a su vez, podría tener un impacto positivo en la calidad y eficiencia del software desarrollado, lo que podría conducir a una mejora en la competitividad de las empresas y una mayor satisfacción de los usuarios finales.

El objetivo general de esta investigación es realizar un estudio comparativo entre la Programación Orientada a Objetos (POO) y la Programación Orientada a Aspectos (POA), con el fin de determinar cuál es más adecuada para diferentes tipos de proyectos de software, mediante el análisis y comparación de sus características, ventajas y

desventajas, su implementación en el código y su rendimiento en términos de escalabilidad, mantenibilidad y otras métricas relevantes.

Para poder realizar una comparación entre la Programación Orientada a Objetos (POO) y Programación Orientada a Aspectos (POA), es necesario tener un conocimiento previo sobre los conceptos bibliográficos sobre los paradigmas de programación y sus elementos, características, ventajas y desventajas, mismos que se detallan a continuación.

Programación Orientada a Objetos

Según Booch (2020), la POO es un paradigma de programación que se enfoca en la creación de objetos que tienen atributos y métodos (funciones) asociados a ellos. En la POO, los objetos son las unidades fundamentales de programación, y se utilizan para modelar y resolver problemas complejos. La POO se basa en cuatro conceptos principales: encapsulamiento, abstracción, herencia y polimorfismo. Este paradigma de programación es ampliamente utilizado en la actualidad en diferentes lenguajes de programación, como Java, Python y C++.

La programación orientada a objetos propone una perspectiva diferente para resolver problemas complejos en comparación con la programación estructurada. A diferencia de la programación procedimental que se centra en los algoritmos, la POO se enfoca en los datos. En lugar de tratar de adaptar un problema al enfoque procedimental de un lenguaje, la POO busca adaptar el lenguaje al problema.

Esto se logra mediante la creación de formatos de datos que se ajusten a las características fundamentales del problema. Los lenguajes orientados combinan tanto los datos como las funciones que actúan sobre ellos en una sola unidad o módulo, conocido como objeto. Si se desea modificar los datos de un objeto, se debe hacer a través de las

funciones miembro del objeto, y no es posible acceder a los datos desde ninguna otra función. Esto simplifica la tarea de escribir, depurar y mantener el programa.

Clase

Para Blanco (2020) “una clase es una abstracción que utilizamos para representar nuestra experiencia sensorial. El ser humano tiende a agrupar objetos o entidades con características similares en categorías o clases” (p. 19). Es decir que, aunque existan numerosos diseños de vasos, somos capaces de identificar un vaso inmediatamente cuando lo vemos, incluso si nunca antes hemos visto ese modelo específico de vaso.

Objeto

De acuerdo con Taylor (2019) “un objeto es un componente individual, mientras que una clase representa a una colección de objetos similares”(p. 49). Se puede entender a una clase como un modelo que se emplea para crear o definir objetos. A partir de una clase se pueden generar múltiples objetos que comparten las mismas características y métodos.

En la Programación Orientada a Objetos, un objeto se compone de un conjunto de datos y métodos. Los datos son lo que se ha denominado como atributos o características, mientras que los métodos son las acciones que el objeto puede llevar a cabo. En un sistema de POO, los datos y los métodos están estrechamente interconectados, formando una sola unidad conceptual y operacional.

Propiedades fundamentales de la POO

Hay varias características asociadas a la programación orientada a objetos. Aunque algunas de estas propiedades también pueden estar presentes en otros paradigmas

de programación, en su conjunto son distintivas de los lenguajes orientados a objetos. Estas características incluyen:

- Encapsulado de datos.
- Herencia.
- Abstracción (tipos abstractos de datos y clases).
- Ocultación de datos.
- Polimorfismo.

Encapsulación y ocultación de datos

El encapsulamiento es un concepto clave en la Programación Orientada a Objetos, según menciona Kickzales (2021) que consiste en "ocultar los detalles internos de un objeto y exponer solo los elementos necesarios para su uso" (p.81). Esto se logra mediante la definición de interfaces públicas, que son los métodos y atributos que pueden ser accedidos desde fuera del objeto, y la ocultación de los detalles internos detrás de interfaces privadas y protegidas, que solo pueden ser accedidos desde dentro del objeto o desde objetos relacionados.

El encapsulamiento permite proteger los datos y métodos internos de un objeto de modificaciones no autorizadas o de acceso no controlado desde el exterior, lo que aumenta la seguridad y la robustez del sistema. Además, facilita el mantenimiento y la evolución del código, ya que los cambios internos pueden ser realizados sin afectar a los usuarios de la interfaz pública del objeto.

Herencia

Según Echeverria (2019) afirma que "la herencia en Programación Orientada a Objetos permite la creación de nuevas clases basadas en clases existentes con el objetivo

de reutilizar el código, lo que da lugar a una jerarquía de clases en la aplicación" (p.3). Al derivar de una clase, una nueva clase hereda sus atributos y métodos, permitiendo la adición de nuevos atributos, métodos o la redefinición de los heredados.

De acuerdo con el párrafo anterior, la cualidad más significativa de un sistema de Programación Orientada a Objetos es su capacidad de ofrecer mayor eficacia y productividad, permitiéndonos ahorrar una gran cantidad de tiempo en programación y resolución de errores. Por esta razón, se cree que un lenguaje de programación no puede ser considerado como orientado a objetos si no incluye la herencia

Abstracción

Para Lamping (2019) "la abstracción es el proceso de identificar las características esenciales y relevantes de un objeto y separarlas de las características que no son importantes en ese contexto" (p. 36). Esto permite representar un objeto de manera más simple, abstracta y generalizada, sin tener que preocuparse por detalles irrelevantes.

Referente a lo mencionado por Lamping, la abstracción en POO es importante porque permite crear modelos simplificados de objetos y entidades del mundo real, lo que facilita la programación, el mantenimiento y la reutilización de código. Además, ayuda a los desarrolladores a concentrarse en lo que es importante para un determinado problema o situación, lo que puede aumentar la eficiencia y reducir la complejidad del código.

Polimorfismo

Maeda (2022) se refiere al polimorfismo como "la capacidad de una operación de tener el mismo nombre en múltiples clases, pero de tener diferentes implementaciones en cada una de ellas" (págs. 11-12). Por ejemplo, la acción de "abrir" puede ser realizada en diferentes clases, como "abrir una puerta", "abrir una ventana", "abrir un periódico", "abrir un archivo", "abrir una cuenta bancaria", "abrir un libro", entre otras. Aunque todas

estas acciones se llaman “abrir”, cada una de ellas tiene una implementación diferente. El polimorfismo permite que una operación sea interpretada por el objeto al que pertenece.

Programación Orientada a Aspectos

Según Navasal (2020) afirma que, “la programación orientada a aspectos (POA) posibilita a los programadores trabajar con un aspecto que está disperso por todo el sistema como una entidad separada, de una forma inteligente, eficiente e intuitiva” (p. 62). Esto significa que se puede escribir, visualizar y modificar un aspecto de manera independiente, lo que hace más fácil y efectivo el manejo de las responsabilidades transversales del programa.

La programación orientada a aspectos (POA) es un paradigma de programación que se enfoca en tratar las responsabilidades transversales de nuestros programas como módulos separados (llamados “aspectos”), con el fin de lograr una correcta separación de responsabilidades. Las responsabilidades transversales son aquellas que se repiten en varias partes de un programa, independientemente de si las secciones en las que aparecen tienen relación directa. Por ejemplo, un método que actualice lo que se muestra en la pantalla de un programa de dibujo puede ser llamado desde métodos encargados de dibujar, recortar, redimensionar, guardar, exportar, deshacer, entre otros.

Consideraciones generales de la POA

El propósito principal de la POA es posibilitar la creación de un programa mediante la descripción individual de cada concepto. Según Bohr (2021) menciona que el Lenguajes Orientados a Aspectos (LAO), es una clase de lenguajes especiales utilizados para apoyar este nuevo enfoque, que proporcionan herramientas y elementos constructivos para identificar aquellos elementos que se extienden a lo largo del sistema, denominados aspectos.

Se podría describir a los lenguajes orientados a aspectos como aquellos que permiten distinguir entre la definición de la funcionalidad esencial y la definición de los distintos aspectos. Estos lenguajes deberían cumplir con diversas características deseables, tales como:

- Cada aspecto debe ser fácilmente reconocible y distinguible.
- Los aspectos no deben perturbar los mecanismos utilizados para definir y mejorar la funcionalidad, como la herencia, ni interferir en su evolución.
- Cada aspecto debe estar contenido en sí mismo y ser independiente de los demás aspectos.
- Los diferentes aspectos no deben afectarse mutuamente ni producir conflictos.
- Los aspectos deben ser capaces de ser intercambiados de manera sencilla y sin causar inconvenientes.

Aspecto (aspect)

Un aspecto se refiere a una preocupación o interés particular dentro de un sistema de software que cruza varios módulos o componentes del mismo. Para Kickzales (2020) "los aspectos son elementos transversales a la estructura modular de un programa, y pueden estar relacionados con funcionalidades no funcionales como la seguridad, el registro de actividades, la validación de entrada de datos, el manejo de excepciones, entre otros" (p.75).

Es posible distinguir los aspectos de los otros elementos del sistema: cuando se va a implementar una funcionalidad, según Pascal Frade, esta puede ser presentada en una de las dos formas siguientes:

1. Como un componente: cuando puede ser contenido (encapsularse) fácilmente en un procedimiento generalizado, siendo bien localizado, accesible y fácil de componer.
2. Como un aspecto: cuando no puede ser fácilmente encapsulado en un procedimiento generalizado. Los aspectos suelen ser propiedades que sistemáticamente afectan el rendimiento o la semántica de los componentes, tales como la sincronización, el manejo de memoria, la distribución, entre otros. (Pascal F, 2022).

Después de distinguir los aspectos de los componentes, podemos definir un aspecto como un concepto que no puede ser encapsulado claramente y que se extiende por todo el código. En la programación orientada a aspectos, los aspectos son la unidad básica. Una definición más formal es que "un aspecto es una unidad modular que se extiende por la estructura de otras unidades funcionales". Los aspectos existen tanto en la etapa de diseño como en la de implementación, y se pueden describir como "unidades modulares de diseño que se mezclan en la estructura de otras partes del diseño" o "unidades modulares de programa que aparecen en otras unidades modulares del programa".

Punto de enlace (joinpoint)

En POA (Programación Orientada a Aspectos), los joinpoints son puntos específicos en el flujo de ejecución de un programa donde se pueden aplicar aspectos. Según Kenens (2021) define "un punto de enlace es un evento en el flujo de ejecución de un programa, como la ejecución de un método, la creación de un objeto, la manipulación de variables, la propagación de excepciones, entre otros" (p. 74).

Los puntos de enlaces son importantes en la POA porque permiten identificar eventos específicos en el código fuente que pueden ser interceptados por un aspecto y modificados. De esta manera, los aspectos pueden ser aplicados selectivamente a partes específicas del código, lo que facilita la modularidad y la reutilización del código.

Es importante destacar que los puntos de enlaces no están limitados a eventos estáticos, también pueden incluir eventos dinámicos durante la ejecución de un programa, como la invocación de métodos a través de interfaces remotas o llamadas a servicios web.

Punto de corte (pointcut)

Según Hernández (2022) "es el encargado de definir, mediante el uso de expresiones regulares (regex), los puntos exactos en el programa donde debe insertarse un aspecto" (págs. 28-30). Un punto de corte permite identificar eventos específicos en el flujo de ejecución de un programa que pueden ser interceptados por un aspecto y modificados. Estos eventos pueden incluir la invocación de un método, la creación de un objeto o la manipulación de variables.

En POA (Programación Orientada a Aspectos), un punto de corte es una expresión que define un conjunto de puntos de un programa donde se pueden aplicar aspectos. En otras palabras, es una especificación de los puntos específicos en el código fuente donde el código del aspecto debe ser ejecutado.

Para Quintero (2020) los punto de corte se definen utilizando expresiones regulares y combinadores lógicos que permiten al programador especificar patrones en el código fuente para identificar puntos específicos en el flujo de ejecución del programa. De esta manera, los aspectos pueden ser aplicados de manera selectiva a partes específicas del código, lo que facilita la modularidad y la reutilización del código.

Avisos (advice)

En POA (Programación Orientada a Aspectos), un "Advice" (Consejo, en español) es el código que se ejecuta en un punto de enlace (joinpoint) identificado por un punto de corte (pointcut). En otras palabras, el Advice es el código que implementa la funcionalidad del aspecto en un punto específico del flujo de ejecución de un programa.

Según Ghezzi (2021) menciona que "los Advice en POA se utilizan para encapsular la funcionalidad transversal de un programa y facilitar la modularidad y la reutilización del código" (págs. 46-50). Al separar la lógica del aspecto del código base del programa, los aspectos se pueden aplicar selectivamente a partes específicas del código, lo que permite una mayor flexibilidad y mantenibilidad del sistema. Existen varios tipos de Advice en POA, incluyendo:

- Before Advice: código que se ejecuta antes de que se ejecute el joinpoint identificado por el pointcut.
- After Advice: código que se ejecuta después de que se haya ejecutado el joinpoint identificado por el pointcut.
- Around Advice: código que se ejecuta tanto antes como después de la ejecución del joinpoint identificado por el pointcut. Este tipo de Advice puede modificar el flujo de ejecución del programa, permitiendo la ejecución condicional de código adicional antes o después del joinpoint. (Cugola M, 2022).

La comparación entre POO (Programación Orientada a Objetos) y POA (Programación Orientada a Aspectos) no está relacionada con una metodología específica de investigación, ya que se trata de una comparación técnica y conceptual. Sin embargo, para analizar las característica, ventajas y desventajas de cada paradigma desde una perspectiva teórica, en esta investigación se utilizó la metodología de revisión bibliográfica o documental, en la cual se analizarían diferentes estudios y artículos científicos que aborden el tema, para identificar las diferencias, similitudes y aplicaciones de cada enfoque.

Ventajas de la POO:

- Modularidad: el código puede ser dividido en módulos para facilitar el mantenimiento y la reutilización.
- Reutilización: la herencia y la composición permiten la reutilización de código.
- Flexibilidad: se pueden crear nuevas clases y objetos con facilidad.
- Abstracción: permite representar los conceptos del mundo real de manera abstracta en el software.
- Encapsulamiento: los datos y la funcionalidad están ocultos y protegidos del acceso no autorizado.
- Polimorfismo: permite crear diferentes objetos que pueden compartir el mismo nombre de método o atributo.

Desventajas de la POO:

- Complejidad: la POO puede ser difícil de aprender y de implementar correctamente.

- Rendimiento: los sistemas POO pueden tener problemas de rendimiento debido a la sobrecarga en la creación y manejo de objetos.
- Dificultades en el mantenimiento: los sistemas POO pueden ser difíciles de mantener y actualizar debido a la complejidad del código.
- Necesidad de planificación: es necesario realizar una planificación adecuada antes de empezar a escribir código para evitar la creación de código desorganizado y difícil de mantener.

Ventajas de la POA:

- Separación de preocupaciones: la POA permite separar la lógica principal del programa de los aspectos transversales, lo que permite una mejor organización del código.
- Flexibilidad: la POA permite cambiar el comportamiento de los programas sin cambiar el código principal.
- Modularidad: los aspectos se pueden separar en módulos para facilitar el mantenimiento y la reutilización.
- Mejora del rendimiento: los aspectos pueden ser aplicados selectivamente, lo que puede mejorar el rendimiento.

Desventajas de la POA:

- Complejidad: la POA puede ser difícil de entender y de aplicar correctamente.
- Cambio de mentalidad: la POA requiere un cambio de mentalidad en la forma de escribir código, lo que puede llevar tiempo para acostumbrarse.

- Depuración: la depuración de programas con aspectos puede ser difícil, ya que el comportamiento puede ser difícil de seguir.
- Necesidad de herramientas especiales: la POA requiere herramientas especiales, como AspectJ o Spring AOP, lo que puede limitar la adopción en algunos entornos.

Tabla 1. Características de POO vs POA.

CARACTERÍSTICAS	PROGRAMACIÓN ORIENTADA A OBJETOS (POO)	PROGRAMACIÓN ORIENTADA A ASPECTOS (POA)
Tipo de programación	Paradigma de programación	Paradigma de programación
Enfoque principal	Modelado de objetos	Separación de aspectos
Unidad de programación	Objeto	Aspecto
Concepto clave	Clases y objetos	Aspectos
Encapsulamiento	Se enfoca en la funcionalidad de un objeto	Se enfoca en la funcionalidad de un aspecto
Herencia y polimorfismo	Ampliamente utilizado	No es un enfoque principal
Modularidad y reutilización	Alcanzable a través de la herencia y composición	Alcanzable a través de los puntos de corte
Flexibilidad	No es tan flexible	Muy flexible
Implementación y mantenimiento	Compleja y costosa	Fácil de implementar y mantener
Escalabilidad y rendimiento	Difícil de escalar y puede tener problemas de rendimiento	Escalable y puede mejorar el rendimiento
Herramientas y lenguajes	Utiliza lenguajes de programación como Java, C# y Python	Utiliza herramientas como AspectJ y Spring AOP
Dificultades	Dificultad para manejar cambios en el tiempo de ejecución	Puede introducir cierta complejidad
Ejemplos de uso	Desarrollo de aplicaciones de escritorio, web y móviles	Mejora de la seguridad, el registro y la transacción en sistemas empresariales

Elaborado por el autor. Fuente: (Ghezzi L, 2021)

Como resultado podemos mencionar que, aunque puede parecer que la POA y la POO son paradigmas idénticos, pero en realidad no lo son. Si se profundiza en el análisis, se pueden observar las diferencias entre ambos enfoques.

La POO permite que los sistemas se organicen en objetos que trabajan juntos en colaboración. Su principal ventaja radica en su capacidad para modelar conceptos comunes con gran eficacia. Sin embargo, esta aproximación falla al tratar de modelar conceptos que se interconectan. La POA resuelve este problema tratando a estos conceptos como elementos principales y extrayéndolos horizontalmente del árbol de herencia vertical.

Tanto la POA como la POO tienen como objetivo crear implementaciones modulares con un acoplamiento mínimo. La distinción radica en que la POA se centra en los conceptos que se interconectan, mientras que la POO se enfoca en los conceptos comunes y los organiza en un árbol de herencia. Ver anexo 1

En la POA, la implementación de los conceptos es independiente, lo que la diferencia de las técnicas utilizadas en la POO. En la POA, el proceso de composición fluye desde los conceptos que se interconectan hacia el concepto principal, mientras que en la POO el flujo es en sentido contrario.

Es común tener una impresión equivocada al leer acerca de la POA, y creer que está limitada a utilizar la POO como base. Sin embargo, es importante aclarar que una implementación de POA puede utilizar cualquier paradigma de programación como base, y aun así mantener los beneficios propios de ese paradigma. La elección de utilizar POO como base se hace para obtener una mejor implementación de los conceptos comunes, y permitir que los conceptos individuales puedan utilizar técnicas orientadas a objetos.

CONCLUSIONES

- Tanto la Programación Orientada a Objetos como la Programación Orientada a Aspectos tienen ventajas y desventajas, y la elección entre ambas dependerá de las necesidades específicas del proyecto.
- La POO se enfoca en el modelado de objetos y la encapsulación de datos y funcionalidades, mientras que la POA se enfoca en la separación de aspectos transversales para mejorar la organización del código.
- Ambas paradigmas de programación ofrecen modularidad y reutilización de código, pero la POO lo logra a través de la herencia y composición, mientras que la POA lo logra a través de puntos de corte.
- Tanto la POO como la POA requieren una planificación adecuada y una comprensión profunda del paradigma para implementarlas correctamente y evitar problemas en el mantenimiento y actualización del código**.**
- Ambos paradigmas de programación pueden mejorar la organización del código y permitir la reutilización y el mantenimiento de éste a largo plazo, pero requieren una planificación y diseño adecuados desde el inicio para evitar problemas futuros.
- La implementación de la programación orientada a aspectos puede requerir la utilización de herramientas especializadas, lo que puede limitar su adopción en algunos entornos y proyectos.

RECOMENDACIÓNES

- Antes de elegir entre la programación orientada a objetos y la programación orientada a aspectos, es importante comprender los conceptos clave y las diferencias entre ambas para seleccionar la que mejor se adapte a las necesidades del proyecto.
- En proyectos grandes y complejos, se recomienda la utilización de ambas técnicas para obtener lo mejor de cada paradigma y lograr una organización eficiente del código.
- Al implementar la programación orientada a aspectos, es importante elegir herramientas que sean compatibles con el lenguaje de programación utilizado en el proyecto y que sean adecuadas para la tarea específica que se esté abordando.
- Asegurarse de que el equipo de desarrollo tenga una comprensión profunda del paradigma elegido y se capaciten si es necesario.
- En el caso de utilizar la POA, es importante contar con herramientas especializadas, como AspectJ o Spring AOP, y capacitarse en su uso.
- En ambos casos, es importante realizar una planificación adecuada para evitar la creación de código desorganizado y difícil de mantener.
- Realizar pruebas exhaustivas y depuración cuidadosa para garantizar que el código funcione correctamente y se pueda mantener a largo plazo.

BIBLIOGRAFÍA

Blanco Y. (2020). Introducción a Programación Orientada a Objetos. Andavira Editora. https://www.google.com.ec/books/edition/Introducci%C3%B3n_a_Programaci%C3%B3n_Orientada/eRqpEAAAQBAJ?hl=es&gbpv=1&dq=que+es+una+clase+en+programacion&printsec=frontcover

Buhr R. (2021). A possible design notation for aspect oriented programming in Proceedings of the Aspect-Oriented Programming. Cannes.

Cugola M. (2022). Introducción a la programación orientada a aspectos,. Scielo. https://victormingueza.wordpress.com/2022/06/12/introduccin-a-la-programacin-orientada-a-aspectos-aop/

Echeverria H. (2019). ¿Qué es la Herencia en programación orientada a objetos? Geek. https://ifgeekthen.nttdata.com/es/herencia-en-programacion-orientada-objetos

Ghezzi L. (2021). “Malaj: A Proposal to Eliminate Clashes Between Aspect-Oriented and Object-Oriented Programming”. Politecnico de Milano.

Grady Booch. (2020). Object-Oriented Analysis and Design with Applications. Series editor. https://www.google.com.ec/books/edition/Object_Oriented_Analysis_and_Design_with/8Rg5wxGj818C?hl=es&gbpv=1&dq=Grady+Booch&printsec=frontcover

Hernández D. (2022). “Visión General de la Programamción Orientada a Aspectos”. Universidad de Sevilla.

Hugunin J. (2020). Getting Started with AspectJ”, in Communications of the ACM (CACM). CACM.

Kenens R. (2021). “An AOP Case with Static and Dynamic Aspects”. Leuven.

Kickzales G. (2021). Aspect- OrientedProgramming", in Proceedings of the European Conference on Obj ect-Oriented Programming. Scielo Educacion.

Lamping J. (2019). "The role ofthe base in aspect oriented programming", in Proceedings of the European Conference on Obj ect-Oriented Programming. Austral Workshops.

Maeda L. (2022). Analysis of inheritance anomaly in objectoriented concurrent programming languajes. Person Edu.

Maldonado V. (2023). Características de los paradigmas POO y POA. EES eduneo. http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectihome/ai5announce.htm

Navasa A. (2020). Un Lenguaje de Descripción Arquiectónica Orientado a Aspectos. Universidad de Extremadura.

Pascal F. (2022). “An Aspect Language for robust programming”, in Proceedings of the European Conference on ObjectOriented Programming (ECOOP). España: Conference on ObjectOriented Programming. https://ecoop.org/#:~:text=ECOOP%202022%20will%20be%20held,Mezini%2C%20TU%20Darmstadt%2C%20Germany.

Quintero R. (2020). “Aspect-Oriented Programming: A Critical Analysis of a new programming paradigm”. Department of Computer Science.

Taylor G. (2019). Fundamentos de Lenguajes Orientados a Objetos. Universidad Informatica. https://www.mhe.es/universidad/informatica/8448150414/archivos/capitulo_23.pdf

Anexo 1: Comparación entre POO y POA

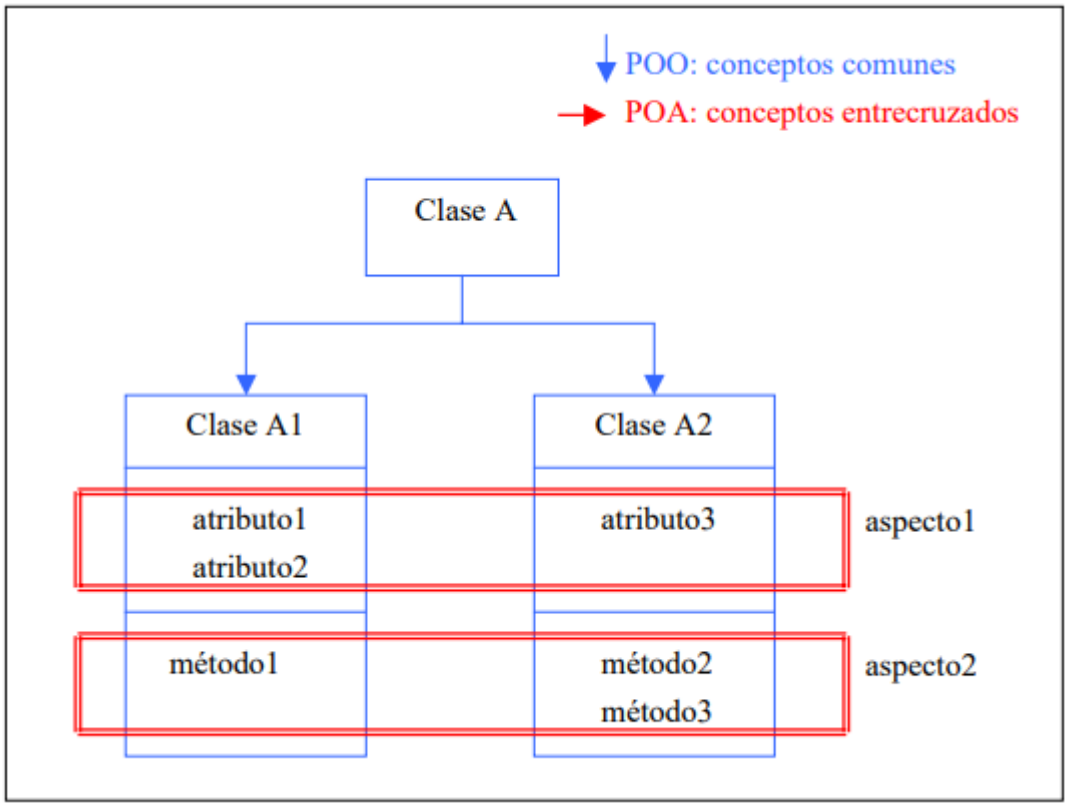

Ilustración 1. Comparación entre POO y POA. Fuente: (Ghezzi L, 2021)

Anexo 1: Reporte Antiplagio