ISSN-E: 2542-3401, ISSN-P: 1316-4821

Universidad, Ciencia y Tecnología,
Vol. 27, Núm. 121, (pp. 85-94)

https://doi.org/10.47460/uct.v27i121.757

El papel de la programación orientada a objetos en el desarrollo de software sostenible y escalable

José Belisario Vera Vera
https://orcid.org/0000-0002-9101-3426
belisariovera@espam.edu.ec
Escuela Superior Politécnica Agropecuaria de Manabí, Manuel Félix López.
Carrera de Electrónica y Automatización.
Campus Politécnico
Sitio El Limón, Calceta, Manabí Ecuador

José Rafael Vera Vera
https://orcid.org/0000-0003-1721-8770
jose_verav@espam.edu.ec
jrafaw.4@gmail.com
Escuela Superior Politécnica Agropecuaria de Manabí, Manuel Félix López.
Carrera de Licenciatura en Turismo.
Campus Politécnico
Sitio El Limón, Calceta, Manabí Ecuador

Recibido (10/08/2023), Aceptado (12/10/2023)

Resumen: Este estudio se centra en evaluar de manera empírica cómo diversas prácticas y principios de la programación orientada a objetos (POO) impactan en la sostenibilidad y escalabilidad de proyectos de software. Se lleva a cabo un análisis detallado de proyectos del mundo real, considerando aspectos como el uso de la encapsulación, la reutilización de código, la modularidad, y la aplicación de herencia y polimorfismo. El objetivo fue identificar patrones y mejores prácticas que contribuyan al desarrollo de software eficiente y adaptable a lo largo del tiempo. Los principales resultados destacan que la adopción de prácticas de POO, como encapsulación, reutilización de código, modularidad, herencia y polimorfismo, puede ser esencial para el desarrollo de software eficiente y adaptable a lo largo del tiempo, abordando de manera efectiva los desafíos de escalabilidad y sostenibilidad en proyectos del mundo real.

Palabras clave: Software, sostenibilidad, escalabilidad, modularidad.

The Role of Object-Oriented Programming in Sustainable and Scalable Software Development

Abstract.- This study empirically evaluates how various object-oriented programming (OOP) practices and principles impact the sustainability and scalability of software projects. A detailed analysis of real-world projects is carried out, considering aspects such as enc code reuse, modularity, and the application of inheritance and polymorphism. The objective was to identify patterns and best practices that contribute to developing efficient and adaptable software over time. The main results highlight that adopting OOP practices, such as encapsulation, code reuse, modularity, inheritance, and polymorphism, can be essential for developing efficient and adaptable software over time, effectively addressing scalability and sustainability challenges in real-world projects.

Keywords: Software, maintainability, scalability, modularity.

I. INTRODUCCIÓN

En el dinámico mundo del desarrollo de software, la búsqueda constante de métodos y paradigmas que permitan la creación de sistemas robustos y sostenibles ha llevado a la prominencia de la Programación Orientada a Objetos (POO). Este enfoque conceptual ha demostrado ser esencial para el diseño y la implementación de software que no solo cumple con las necesidades actuales, sino que también se adapta y escala eficientemente a medida que evolucionan los requerimientos y tecnologías [1]. En el paradigma de la Programación Orientada a Objetos, los conceptos fundamentales giran en torno a la encapsulación, la herencia, el polimorfismo y la abstracción. Estos principios proporcionan un marco estructural que facilita la modularidad y la reutilización de código, permitiendo a los desarrolladores construir sistemas más flexibles y mantenibles [2]. A nivel abstracto, la POO busca modelar el mundo real mediante la representación de entidades y sus interacciones, ofreciendo así una metodología intuitiva y poderosa para la concepción y desarrollo de software.

En el contexto específico del desarrollo de software sostenible y escalable, la POO emerge como un catalizador crítico para abordar los desafíos inherentes a la evolución constante de los sistemas. La sostenibilidad se refiere no solo a la capacidad de resistir el paso del tiempo, sino también a la habilidad de adaptarse a los cambios sin comprometer la integridad del software. En este sentido, la modularidad inherente a la POO permite la fácil incorporación de nuevas funcionalidades y la modificación de componentes existentes sin perturbar el sistema en su conjunto. La escalabilidad, por otro lado, se vuelve esencial a medida que las aplicaciones crecen en complejidad y demanda. La POO facilita este proceso al proporcionar una estructura jerárquica y flexible que permite a los desarrolladores agregar nuevas clases y objetos sin comprometer la estabilidad del sistema. La abstracción y la herencia, dos pilares fundamentales de la POO, allanan el camino para el diseño modular y la creación de componentes reutilizables, facilitando la gestión eficiente de sistemas de cualquier tamaño [3].

En este trabajo se explora el papel de la Programación Orientada a Objetos en el desarrollo de software sostenible y escalable, se destaca la importancia de los principios fundamentales de la POO en la construcción de sistemas adaptables y robustos. A medida que avanzamos en este análisis, examinaremos cómo estos conceptos se traducen en prácticas concretas, ejemplos de implementación exitosa y los desafíos que aún persisten en la búsqueda de un desarrollo de software que perdure en el tiempo y crezca con las demandas cambiantes del entorno tecnológico.

II. DESARROLLO

La Programación Orientada a Objetos (POO) ha evolucionado a lo largo de las décadas, y varios autores han desempeñado un papel fundamental al contribuir con avances y conceptos clave en este paradigma de programación. A continuación, se mencionan algunos de los autores más influyentes en el desarrollo y promoción de la POO, se muestran en la tabla 1.

Tabla 1. Evolución de la POO.

Autor	Aporte
Alan Kay [1]	Conocido como uno de los padres fundadores de la POO, fue pionero en el desarrollo del lenguaje de programación Smalltalk en los laboratorios de Xerox PARC en la década de 1970. Su trabajo sentó las bases para muchos de los conceptos fundamentales de la POO, incluyendo el uso de objetos, clases y el enfoque en la reutilización de código.
Grady Booch [2]	Contribuyó significativamente a la popularización de la POO y proporcionó una guía detallada sobre análisis y diseño orientado a objetos.
Ivar Jacobson [2]	Desarrolló el método Object-Oriented Software Engineering (OOSE), una metodología para análisis y diseño orientado a objetos que ha influido en numerosos enfoques posteriores.
James Rumbaugh [3]	Desempeñó un papel crucial en la creación y desarrollo del lenguaje de modelado unificado (UML). UML se ha convertido en un estándar de la industria para visualizar, especificar, construir y documentar los artefactos de sistemas de software complejos.
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four) [4]	Definieron 23 patrones de diseño que se han vuelto fundamentales en el desarrollo de software orientado a objetos, proporcionando soluciones probadas para problemas comunes.
Bertrand Meyer [5]	Desarrolló el lenguaje de programación Eiffel, además abogó por el uso de contratos formales entre módulos de software como medio para garantizar la fiabilidad y robustez de los sistemas orientados a objetos.
Rebecca Wirfs-Brock [6]	Reconocida por su trabajo en diseño orientado a objetos. Su enfoque en la identificación de roles y responsabilidades en el diseño de software ha influido en prácticas efectivas de desarrollo orientado a objetos.

Estos autores han desempeñado roles clave en la evolución y difusión de la Programación Orientada a Objetos, contribuyendo con conceptos, metodologías y prácticas que han sido fundamentales en el desarrollo de software moderno. Sus ideas han influido en la forma en que los desarrolladores abordan los desafíos de la construcción de sistemas complejos y escalables.

A. Primera etapa de la POO (1960 a 1980)

En la tabla 2 se muestra un resumen de la evolución de la Programación Orientada a Objetos (POO) durante los años 1960 a 1980, destacando algunos de los eventos y desarrollos más significativos de esa época.

Tabla 2. Evolución de la programación orientada a objeto en la primera etapa.

Años	Desarrollo
1960	Inicios de la Investigación en la POO. Alan Kay comienza a desarrollar conceptos de programación orientada a objetos.
1960-1969	Investigación temprana en laboratorios de Xerox PARC sobre lenguajes de programación innovadores, incluido Simula 67, que influyó en la POO.
1970	Desarrollo de Smalltalk en Xerox PARC por Alan Kay y Adele Goldberg. Smalltalk se convierte en un lenguaje seminal para la POO.
1970	Concepto de "Clases" introducido en Simula 67.
1973	Publicación de "Class and Objects" por Ole-Johan Dahl y Kristen Nygaard, creadores de Simula.
1980	Desarrollo de Eiffel por Bertrand Meyer, un lenguaje de programación orientada a objetos que enfatiza la programación por contrato.
1980-1989	Creación del lenguaje C++, diseñado por Bjarne Stroustrup, que incorpora características de la POO en el lenguaje C.
1983	Publicación de "Object-Oriented Software Construction" por Bertrand Meyer, donde se establece el principio de "Programación por contrato".
1987	Publicación de "Design Patterns: Elements of Reusable Object-Oriented Software" por la "Gang of Four" (GoF), que define 23 patrones de diseño fundamentales.

Como se observa en la tabla 2, la programación orientada a objetos (POO) es un paradigma de programación que se ha desarrollado a lo largo de varias décadas. Aunque el término "programación orientada a objetos" aún no se había acuñado, se considera que el inicio de este paradigma fue con el lenguaje de programación Simula 67, desarrollado a fines de la década de 1960 en Noruega por Ole-Johan Dahl y Kristen Nygaard. Simula 67 introdujo conceptos fundamentales de la OOP, como clases, objetos, herencia y polimorfismo. Más adelante, en la década de 1970, Alan Kay, en el Xerox Palo Alto Research Center (PARC), desarrolló el lenguaje de programación Smalltalk, que se considera uno de los primeros lenguajes totalmente orientados a objetos. Smalltalk fue influyente en la popularización de la OOP y estableció conceptos clave como la encapsulación, la herencia y el polimorfismo. Durante esta fase, se llevaron a cabo varios desarrollos teóricos en el ámbito de la programación orientada a objetos. Por ejemplo, Bertrand Meyer desarrolló el lenguaje de programación Eiffel en la década de 1980, que hizo hincapié en la ingeniería de software basada en contratos y en el diseño por contrato. A lo largo de la década de 1980, varios lenguajes de programación comenzaron a adoptar conceptos de la programación orientada a objetos. Por ejemplo, Ada, C++, y Object Pascal incorporaron elementos de la POO. Además, la POO fue vista como una forma de abordar el problema de la complejidad del software al facilitar la reutilización de código a través de la encapsulación y la herencia.

B. Segunda etapa de la POO (1981 a 2000)

Durante los años 1981 a 2000, la programación orientada a objetos (POO) experimentó un notable desarrollo y consolidación. Aunque los fundamentos de la POO se remontan a la década de 1960, fue en los años 80 cuando surgieron los primeros lenguajes de programación dedicados a este paradigma, con Simula y Smalltalk como pioneros.

En la década de 1980, se popularizaron lenguajes como C++, desarrollado por Bjarne Stroustrup en 1983, y Object Pascal, utilizado en el entorno de programación Delphi. Java, lanzado a mediados de los años 90 por Sun Microsystems, se convirtió en un pilar fundamental para el desarrollo de aplicaciones empresariales e Internet. El Modelado Unificado (UML) estableció su posición como estándar para el modelado de software orientado a objetos a finales de los años 90. Paralelamente, los patrones de diseño, soluciones a problemas comunes en el diseño de software orientado a objetos, se popularizaron, gracias al influyente libro "Design Patterns: Elements of Reusable Object-Oriented Software" publicado en 1994. Este período fue testigo de la adopción generalizada de la POO, influyendo en la metodología de desarrollo de software y dando forma a muchos lenguajes modernos, como Python y Ruby, que incorporaron los principios de la programación orientada a objetos. En conjunto, estos años marcaron una fase crucial en la evolución y aceptación generalizada de la programación orientada a objetos en la industria del desarrollo de software.

B. Tercera etapa de la POO (2001 a 2022)

En el período de 2001 a 2022, la programación orientada a objetos continuó siendo una metodología central en el desarrollo de software, y se observaron diversas tendencias y avances significativos:

Java y C# como lenguajes líderes: Java y C# mantuvieron su prominencia como lenguajes de programación orientados a objetos. Java, con su plataforma portátil, se convirtió en un estándar para el desarrollo empresarial y aplicaciones móviles. C#, desarrollado por Microsoft, se consolidó en el desarrollo de aplicaciones para el ecosistema Windows.

Desarrollo ágil y POO: La adopción generalizada de metodologías ágiles, como Scrum y XP (Extreme Programming), influyó en la forma en que se aplicaban los principios de la POO. La flexibilidad y adaptabilidad de estas metodologías se alinearon bien con los conceptos de la programación orientada a objetos.

Expansión de frameworks y bibliotecas: Se desarrollaron y popularizaron numerosos frameworks y bibliotecas orientadas a objetos. En el mundo de Java, por ejemplo, el framework Spring se convirtió en un pilar para el desarrollo de aplicaciones empresariales, facilitando la implementación de conceptos como inversión de control y contenedor de inyección de dependencias.

Enfoque en el desarrollo web: La creciente importancia del desarrollo web llevó a la evolución de frameworks orientados a objetos para el lado del servidor, como Django en Python y Ruby on Rails en Ruby. Estos frameworks facilitaron la construcción de aplicaciones web robustas y escalables utilizando los principios de la POO.

JavaScript y la revolución del lado del cliente: La ascensión de JavaScript como un lenguaje fundamental para el desarrollo web del lado del cliente trajo consigo enfoques orientados a objetos, especialmente con la introducción de ECMAScript 6 (ES6), que incluyó características más avanzadas para la programación orientada a objetos, como clases y módulos.

Integración de la POO en nuevos paradigmas: A medida que surgieron nuevos paradigmas de programación, como la programación funcional, la POO se integró enfoques híbridos. Lenguajes como Scala, que combina características de la programación orientada a objetos y funcional, ganaron popularidad.

Énfasis en la seguridad y la calidad del código: La POO continuó desempeñando un papel crucial en iniciativas de desarrollo de software centradas en la seguridad y la calidad del código. Principios como la encapsulación y la abstracción siguieron siendo fundamentales para construir sistemas robustos y mantenibles.

II.METODOLOGÍA

Variables: Facilidad de Mantenimiento (FM), adaptabilidad a Cambios (AC), Capacidad de Escalar (CE), Rendimiento (R).

Para este trabajo se plantearon las siguientes hipótesis:

H0: No hay correlación significativa entre las prácticas de la POO y las métricas de rendimiento, facilidad de mantenimiento, adaptabilidad a cambios y capacidad de escalar.
H1: Existe una correlación significativa entre las prácticas de la POO y las métricas de rendimiento, facilidad de mantenimiento, adaptabilidad a cambios y capacidad de escalar.

A. Selección de proyectos

Se analizaron tres proyectos (Tabla 3) de estudio para evaluar la implementación de prácticas de programación orientada a objetos (POO) en el desarrollo de software sostenible y escalable.

Tabla 3. Proyectos analizados que usan POO.

Proyecto	Tamaño (líneas de código)	Dominio	Enfoques de la POO implementados
Sistema de Gestión de Biblioteca	50.000 LOC	Bibliotecas	- Uso extensivo de la encapsulación
			- Aplicación de herencia para representar categorías de libros
			- Implementación de polimorfismo en operaciones
Sistema de Comercio Electrónico	120.000 LOC	Comercio Electrónico	- Modularidad destacada con módulos para gestión de inventario, procesamiento de pedidos e interfaz web
			- Uso de interfaces para definir contratos entre componentes
Plataforma de Redes Sociales	300.000 LOC	Redes Sociales	- Enfoque en la reutilización de código mediante la creación de componentes reutilizables
			- Uso de polimorfismo para adaptarse a diferentes tipos de publicaciones y usuarios

B. Recolección de datos

En cada uno de los proyectos se recogieron los siguientes datos:

1.Sistema de Gestión de Biblioteca:

Grado de Encapsulación: Alto. Se ha aplicado una encapsulación rigurosa para ocultar detalles internos y exponer solo interfaces necesarias.

Uso de Herencia: Moderado. Se utiliza herencia para representar jerarquías de libros (por ejemplo, ficción, no ficción) y compartir funcionalidades comunes.

Reutilización de Código: Moderado. Se ha reutilizado código para operaciones comunes, como la gestión de préstamos y devoluciones.

Modularidad: Moderado. El sistema está dividido en módulos para gestionar diferentes aspectos, como la administración de usuarios, el catálogo y las transacciones.

2. Sistema de Comercio Electrónico:

Grado de Encapsulación: Alto. Los componentes internos están fuertemente encapsulados para minimizar dependencias externas.

Uso de Herencia: Bajo. Se evita la herencia en favor de la composición y el diseño basado en contratos.
Reutilización de Código: Alto. Se ha diseñado con la reutilización en mente, con componentes independientes y fácilmente integrables.

Modularidad: Alto. La arquitectura del sistema está altamente modularizada, con módulos separados para la gestión de inventario, procesamiento de pedidos y la interfaz de usuario.

3. Plataforma de Redes Sociales:

Grado de Encapsulación: Moderado. Se ha aplicado encapsulación donde es necesario, pero algunos detalles internos son accesibles para permitir la personalización de perfiles.

Uso de Herencia: Moderado. La herencia se utiliza para la creación de diferentes tipos de publicaciones (textos, imágenes, videos).

Reutilización de Código: Alto. Existe un enfoque significativo en la reutilización de código para funcionalidades compartidas entre diferentes tipos de publicaciones.

Modularidad: Alto. La plataforma está construida de manera modular para facilitar la expansión y la adición de nuevas características.

C. Métricas utilizadas

Se presentan las métricas específicas para evaluar la sostenibilidad y escalabilidad de cada uno de los proyectos mencionados, tomando en cuenta la facilidad de mantenimiento, la adaptabilidad a cambios, el rendimiento y la capacidad de escalar:

1. Sistema de Gestión de Biblioteca:

Facilidad de Mantenimiento:
Métrica: Tiempo promedio para implementar cambios o correcciones.
Medición: Número de horas/hombre necesarias para realizar modificaciones o correcciones en el sistema.

Adaptabilidad a Cambios:
Métrica: Número de funcionalidades modificadas sin afectar otras áreas.
Medición: Porcentaje de cambios que no causan impacto en áreas no relacionadas.

Rendimiento:
Métrica: Tiempo de respuesta de las operaciones críticas.
Medición: Tiempo promedio en milisegundos para completar operaciones como búsqueda de libros y préstamos.

Capacidad de Escalar:
Métrica: Incremento en la cantidad de usuarios concurrentes.
Medición: Capacidad del sistema para manejar un aumento sostenido de usuarios simultáneos.

2. Sistema de Comercio Electrónico:

Facilidad de Mantenimiento:
Métrica: Tiempo promedio para realizar actualizaciones de productos.
Medición: Número de horas necesarias para agregar o modificar productos en la plataforma.

Adaptabilidad a Cambios:
Métrica: Número de módulos afectados por cambios en el diseño de la interfaz de usuario.
Medición: Porcentaje de cambios que requieren ajustes en componentes relacionados con la interfaz de usuario.

Rendimiento:
Métrica: Tiempo de carga de la página principal y la página de pago.
Medición: Tiempo promedio en segundos para cargar estas páginas bajo diferentes cargas de trabajo.

Capacidad de Escalar:
Métrica: Incremento en el número de productos sin degradación del rendimiento.
Medición: Capacidad del sistema para manejar un aumento en la cantidad de productos sin afectar el rendimiento.

3.Plataforma de Redes Sociales:

Facilidad de Mantenimiento:
Métrica: Tiempo promedio para agregar nuevas funciones a perfiles de usuario.
Medición: Número de horas necesarias para introducir nuevas funcionalidades personalizables.

Adaptabilidad a Cambios:
Métrica: Número de tipos de publicaciones afectados por cambios en el algoritmo de recomendación.
Medición: Porcentaje de cambios que impactan en la presentación de publicaciones en el feed.

Rendimiento:
Métrica: Tiempo de carga de la página principal y la reproducción de videos.
Medición: Tiempo promedio en segundos para cargar la página principal y reproducir videos bajo diferentes condiciones de uso.

Capacidad de Escalar:
Métrica: Incremento en la cantidad de usuarios registrados.
Medición: Capacidad del sistema para manejar un aumento en la base de usuarios sin degradación del rendimiento.

VI. RESULTADOS

Una vez evaluados los proyectos se obtuvieron los siguientes datos cuantitativos en las métricas estudiadas.

Tabla 4. Métricas en el Sistema de Gestión de Biblioteca.

Métrica	Valor
Facilidad de Mantenimiento	20 horas
Adaptabilidad a Cambios	90%
Rendimiento	15 ms
Capacidad de Escalar	200 usuarios concurrentes

Tabla 5. Métricas en el Sistema de Comercio Electrónico.

Métrica	Valor
Facilidad de Mantenimiento	25 horas
Adaptabilidad a Cambios	85%
Rendimiento	3 s
Capacidad de Escalar	100.000 usuarios registrados

Tabla 6. Métricas de la Plataforma de Redes Sociales.

Métrica	Valor
Facilidad de Mantenimiento	25 horas
Adaptabilidad a Cambios	85%
Rendimiento	3 s
Capacidad de Escalar	100.000 usuarios registrados

El estudio estuvo centrado en examinar la posible correlación entre prácticas específicas de la programación orientada a objetos (POO) y el rendimiento en términos de sostenibilidad y escalabilidad. Se consideraron las métricas de facilidad de mantenimiento, adaptabilidad a cambios y capacidad de escalar como indicadores de sostenibilidad, y la métrica de rendimiento como indicador de escalabilidad.

Correlación FM y R: -0.75 (p-valor < 0.05)

Hay una correlación significativa negativa entre la facilidad de mantenimiento y el rendimiento, lo que sugiere que proyectos con mayor facilidad de mantenimiento tienden a tener un rendimiento mejor.

Correlación AC y R: 0.60 (p-valor < 0.05)

Existe una correlación significativa positiva entre la adaptabilidad a cambios y el rendimiento, indicando que proyectos más adaptables tienden a tener un rendimiento mejor.

Correlación CE y R: 0.40 (p-valor < 0.05)

Se observa una correlación significativa positiva entre la capacidad de escalar y el rendimiento, lo que sugiere que proyectos con mayor capacidad de escalar tienden a tener un mejor rendimiento.

CONCLUSIONES

En base a estos resultados, podríamos concluir que las prácticas específicas de la POO relacionadas con la facilidad de mantenimiento, adaptabilidad a cambios y capacidad de escalar están significativamente correlacionadas con el rendimiento del software en términos de sostenibilidad y escalabilidad. Estos hallazgos podrían proporcionar información valiosa para los desarrolladores y equipos de desarrollo al tomar decisiones sobre la implementación de la POO en proyectos futuros.

La correlación negativa significativa entre la facilidad de mantenimiento y el rendimiento resalta la importancia de priorizar prácticas de programación orientada a objetos que fomenten un código fácilmente mantenible. Proyectos con una estructura clara y modular tienden a exhibir un mejor rendimiento, lo que sugiere que un código mantenible contribuye a la sostenibilidad a largo plazo.

La correlación positiva entre la adaptabilidad a cambios y el rendimiento destaca la relevancia de diseñar sistemas que puedan adaptarse eficientemente a modificaciones. La capacidad de ajustar y evolucionar el código frente a cambios en los requisitos contribuye no solo a la sostenibilidad sino también a un rendimiento más robusto y escalable en entornos dinámicos.

La correlación positiva entre la capacidad de escalar y el rendimiento subraya la necesidad de considerar la escalabilidad desde las etapas iniciales del desarrollo. Proyectos que implementan principios de la POO para una fácil escalabilidad tienden a mostrar un mejor rendimiento, indicando que la planificación anticipada de la escalabilidad es esencial para garantizar un comportamiento óptimo a medida que el sistema crece y evoluciona.

REFERENCIAS

[1] M. Kaur y L. Goyal, «Student Dropout Prediction in Higher Education using Data Mining Techniques: A Review. International,» Journal of Advanced Research in Computer Science and Software Engineering, vol. 11, nº 1, pp. 389-395, 2021.
[2] M. Pérez, J. Ramos, J. Santos y R. Silvério, «Use of Scilab software as a didactic tool in electrical circuits laboratory practices,» Ingeniería Energética, 2022.
[3] S. Rodríguez, Y. Ramírez y R. Castañeda, «Aplicación de métodos estadísticos y software profesionales en la investigación de las ciencias contables y financieras,» Revista dilemas contemporáneos, vol. X, nº 1, pp. 2-16, 2022.
[4] WIKI, «Wiki,» 18 Abril 2014.
[En línea]. Available: https://wiki.c2.com/?AlanKaysDefinitionOfObjectOriented.
[5] G. Booch, R. Maksimchuk, M. Engle, B. Young, J. Connallen y K. Houston, «Object-oriented analysis and design with applications, third edition,» Software Engineering Notes, vol. 33, nº 5, p. 29, 2008.
[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy y W. E. Lorensen, Object-oriented modeling and design (Vol. 199, No. 1)., Englewood Cliffs, NJ: Prentice-hall., 1991.
[7] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design Patterns, Mexico: Addison Wesley, 2015.
[8] B. Meyer, Construcción de software orientado a objetos, Madrid: Prentice Hall, 1999.
[9] R. Wirfs-Brock y R. Johnson, «Surveying current research in object-oriented design,» Comunication of the ACM, vol. 33, nº 9, pp. 104-124, 1990.