ISSN-E: 2697-3650

Minerva Journal
Vol.4, Issue N°12, (pp. 39-47)

https://doi.org/10.47460/minerva.v4i12.135

Impacto de las innovaciones en la programación orientada a objetos en la creación de nuevos patrones de diseño

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

Julio Agustín Molina Zambrano
https://orcid.org/0009-0002-4005-4014
gmolina@espam.edu.ec
Escuela Superior Politécnica Agropecuaria de Manabí, Manuel Félix López
Centro de Aprendizaje de Aplicaciones Informáticas
Campus Politécnico, Sitio El Limón, Calceta
Manabí, Ecuador

Yimmy Salvador Loor Vera
https://orcid.org/0009-0004-7516-3207
yloor@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 (23/09/2023), Aceptado (21/11/2023)

Resumen: Esta investigación está centrada en profundizar sobre el Impacto de las innovaciones en la programación orientada a objetos en la creación de nuevos patrones de diseño. En este trabajo se presentan posibles innovaciones en la programación orientada a objetos y su impacto en los patrones de diseño para el desarrollo de software. Los desarrolladores de estos sistemas enfrentan el reto de diseñar no un sistema único, sino varias formas de resolver un mismo problema e introducir innovaciones que permitan modificar y extender los códigos de una manera rápida y sencilla. Para ilustrar cómo se aplica un patrón de diseño en un sistema orientado a objetos, se muestra un sistema de gestión de pedidos para una venta de repuestos de vehículos en línea. Para este caso, se selecciona el patrón de diseño Factory Method para gestionar la creación de objetos de productos (repuestos) de manera dinámica y eficiente, lo que constituye un primer paso para construir sistemas de grandes dimensiones.

Palabras clave: diseño de software, patrones de diseños, programación orientada a objetos, innovación.

Impact of innovations in object-oriented programming on the creation of new design patterns

Abstract.- This research focuses on delving into the impact of innovations in object-oriented programming on creating new design patterns. This paper presents possible innovations in object-oriented programming and its impact on design patterns for software development. The developers of these systems face the challenge of designing not a single system but several ways to solve the same problem and introduce innovations that allow them to modify and extend the codes quickly and easily. To illustrate how a design pattern is applied in an object-oriented system, an order management system for an online vehicle parts sale is shown. For this case, the Factory Method design pattern is selected to handle creating product objects (spare parts) dynamically and efficiently, which is a first step in building large systems.

Keywords: software design, design patterns, object-oriented programming, innovation.

I. INTRODUCCIÓN

Una característica común de los desarrolladores de software es la necesidad de actualizar un sistema. Esto se facilita con el uso e implementación de patrones de diseño, los cuales son soluciones típicas a problemas comunes en el desarrollo de software. La diferencia entre los patrones de diseño de funciones y bibliotecas es que no se pueden copiar directamente en el programa, debido que no son un fragmento de código, sino un concepto que sirve como solución [1][2]. Por tanto, para implementar un patrón de diseño, hay que adaptarlo al problema que se quiere resolver. Esto dependerá de las características del proyecto. De esta forma se crea una implementación que se ajusta exactamente a las necesidades de nuestra aplicación.

Por otra parte, la programación orientada a objetos es un modelo de programación donde el diseño de software se organiza alrededor de datos u objetos. Se enfoca en los objetos que los programadores necesitan manipular- Un objeto se puede definir como un campo de datos con atributos y comportamientos únicos [3] [4].

En la actualidad existen diferentes lenguajes de programación orientada a objetos, como C++, Objective C, Java, Ruby, Visual Basic, Visual C Sharp, Simula, Perl, TypeScript, Smalltalk, PHP o Python. C++ y Java son los dos lenguajes de programación orientada a objetos más usados. Por otra parte, Python, PHP y Ruby son otros lenguajes de programación orientada a objetos muy populares, aunque más enfocados en la programación, desarrollo web y de aplicaciones para móviles [1][5].

Entre las ventajas de la programación orientada a objetos se tiene la facilidad para detectar errores en el código. En los lenguajes de programación orientada a objetos no es necesario revisar línea por línea del código para detectar un error. Gracias a la encapsulación los objetos son autónomos, de manera que es más fácil abstraer un problema y saber dónde buscar el error cuando algo no funciona bien. La modularidad es relevante, ya que así un equipo puede trabajar en múltiples objetos a la vez mientras se minimizan las posibilidades de que un programador duplique la funcionalidad de otro. El trabajo modular también permite dividir los problemas en partes más pequeñas que se pueden probar de manera independiente [6][7].

Para desarrollar un software robusto es esencial centrarse en algunos atributos. El código debe ser simple, legible y fácil de entender, débilmente acoplado, eficiente, autodocumentado y comprobable. Los programadores dedican gran cantidad de tiempo a actualizar, modificar y mejorar el software. El costo, el esfuerzo y el tiempo pueden reducirse siguiendo buenas prácticas en el desarrollo de software [8].

Existen varias metodologías en el desarrollo orientado a objetos; los patrones de diseño es una de ellas. Debido que el diseño orientado a objetos es complejo, es de suma importancia que los programadores de software aprovechen la experiencia de otros mediante el uso de marcos o patrones de diseño [9]. El patrón de diseño orientado a objetos es un área de investigación muy activa en el desarrollo de software. En [10] exploran los beneficios de implementar patrones en diseños de software. Demuestran que un objeto que permite la codificación de un patrón para un entorno específico puede crearse y utilizarse en diferentes fases del desarrollo del software [11]. En [12], se realizó un análisis para verificar la reutilización de patrones de diseño y paquetes de software, que describe algunas ventajas y desventajas de estos patrones. Desarrollaron ejemplos que emplean patrones de diseño, junto a diseños alternativos que resuelven el mismo problema, además se utilizó una herramienta para facilitar la extensibilidad.

En este trabajo se muestran los resultados de una investigación empírica para estudiar el Impacto de las innovaciones en la programación orientada a objetos en la creación de nuevos patrones de diseño. El artículo se divide en la forma siguiente; en sección 2 se muestra el desarrollo del trabajo, la sección 3 explica las estrategias metodológicas seguidas en la investigación, la sección 4 en resultados, se muestra un ejemplo del uso de patrones de diseño. Finalmente, se muestran las conclusiones y las referencias más relevantes usadas para desarrollar el trabajo.

II. DESARROLLO

La programación orientada a objetos y los patrones de diseño son dos conceptos fundamentales en el desarrollo de software. Su comprensión y aplicación efectiva son cruciales para construir softwares robustos. En adelante se explora la relación entre la programación orientación a objetos y los patrones de diseño, y cómo su combinación puede innovar para mejorar la eficiencia y la calidad en desarrollo de software [13].
La orientación a objetos es un paradigma se basa en el concepto de objetos. Los objetos son entidades que representan elementos del mundo real y encapsulan datos y comportamientos relacionados. Mientras, los datos y las funciones se tratan por separado, la orientación a objeto los combina en objetos interconectados. En la orientación a objetos, estos son instancias de clases, que actúan como plantillas para definir la estructura y el comportamiento de estos. Estas clases definen atributos (variables) y métodos (funciones) que describen las propiedades y acciones de los objetos [14].

La aplicación adecuada de patrones de diseño presenta algunas ventajas en el desarrollo de software [15] [16]:
·Reutilización de Soluciones: ofrecen soluciones probadas que los programadores pueden reutilizar en diferentes proyectos o sistemas.

- Mantenibilidad: Facilitan la comprensión y el mantenimiento del código al proporcionar una estructura organizada, entendible y manejable.
- Flexibilidad: permiten que el software sea más flexible y adaptable a cambios acorde con el sistema a desarrollar.
- Comunicación efectiva: proporcionan un lenguaje común para describir soluciones y ayudan a los equipos de programadores a comunicarse de manera más efectiva.

A. Patrones de diseño en un entorno orientado a objetos

Los patrones de diseño se aplican de manera efectiva en un entorno orientado a objetos aprovechando los conceptos y principios fundamentales de la programación orientada a objetos. Varias formas de lograr esta aplicación son las siguientes [17]:

- Herencia y Abstracción: Se usa para implementar patrones estructurales y de comportamiento. Las clases base pueden servir como puntos de partida para aplicar patrones como el Adapter o el Strategy.

- Polimorfismo: Permite que objetos de diferentes clases respondan de manera única a los mismos mensajes. Esto es fundamental para la implementación de patrones de comportamiento, como el Observer o el Command, donde varios objetos pueden reaccionar de manera diferente a un evento.

- Encapsulación: Asegura que los detalles internos de un objeto se mantengan ocultos, lo que facilita la aplicación de patrones creacionales como el Singleton o el Factory Method para gestionar la creación y acceso a objetos.

Los conocimientos sobre el diseño de software son extensos y significativos. Sin embargo, existen nuevos retos que necesariamente implican innovaciones. Los retos actuales que se vislumbran en el diseño de software actualmente son:

1. La fuerte influencia de las bibliotecas y los marcos en la toma de decisiones y diseño de software. Los programadores reutilizan varias bibliotecas, marcos, y arquitecturas no solo para acelerar el desarrollo de software, sino también para apoyar sus diferentes funciones y requisitos no funcionales. Estas bibliotecas, marcos, y arquitecturas frecuentemente obligan a los desarrolladores a cambiar la forma de implementar sus modelos de orientación a objetos.
2. El software será más complejo a medida de la complejidad del problema, por la cantidad de variables y parámetros involucrados. El mismo software podría tener múltiples usuarios que tienen sus perspectivas e intereses muy distintos, los cuales también tienen sus requerimientos específicos. En la práctica, esto significa que la representación de un modelo único del problema no es suficiente.
3. Medición La calidad del diseño es fundamental para apoyar a los programadores decidan sobre cual patrón de diseño utilizar. Actualmente, los desarrolladores no confían en el diseño automatizado existente.

III. METODOLOGÍA

La investigación es descriptiva debido detalla el proceso estudiado a través de la medición de uno o más variables. Con esta investigación, se infiere cómo se relacionan o vinculan diversos procesos entre sí. Lo principal es saber cómo se comporta una variable conociendo el desempeño de otra variable relacionada. También es aplicada, debido que se enfoca en el diseño de un sistema real.

La metodología aplicada en el desarrollo de un software en Python implica la selección y aplicación de patrones de diseño adecuados para resolver problemas comunes. Algunos patrones de diseño comunes en Python incluyen el patrón creacional Abstract Factory, Builder, Factory Method, Prototype y Singleton, así como el patrón estructural Adapter. En el caso de estudio mostrado en este trabajo se usa Factory Method.

Como caso de estudio se tiene una tienda que vende repuestos de vehículos. Se desarrolla un código, que permite al cliente conocer una serie de características de los repuestos (objeto) antes de proceder a la compra en línea. El código se desarrolla para tres productos, pero se puede extender para introducir más elementos en la variable repuesto.

Figura 1. Algoritmo ejecutado en el trabajo propuesto.
Fuente: Elaboración propia.

IV. RESULTADOS

A. Ejemplo de un patrón de diseño en un sistema orientado a objetos

Para ilustrar cómo se aplica un patrón de diseño en un sistema orientado a objetos, se plantea un estudio de caso: desarrollar un sistema de gestión de pedidos para una venta de repuestos de vehículos en línea. Se selecciona el patrón de diseño Factory Method para gestionar la creación de objetos de productos (repuestos) de manera dinámica y eficiente.

La tienda en línea ofrece una variedad de productos, como neumáticos, radiadores e inyectores; es evidente que cada tipo de producto tiene una estructura y un proceso distinto. En lugar de crear manualmente cada gestión de producto, se implementa el patrón Factory Method para simplificar el proceso.

En la implementación del Factory Method, en primer lugar, se define una interfaz común llamada repuesto, que todas las clases de productos deben implementar. Esto asegura que cada tipo de repuesto tenga formas consistentes, sobre cómo obtener se descripción y calcular su precio.

A continuación, se crean clases concretas para cada tipo de repuesto: neumáticos, radiadores e inyectores, que implementan la interfaz repuesto y proporcionan detalles específicos sobre cómo se describen y calculan los precios de esos repuestos.

Es fundamental la creación de una clase *FabricaRepuestos* que contiene un método llamado *crearRepuesto*, el cual acepta un parámetro que indica qué tipo de repuesto se debe crear. Por ejemplo, si el parámetro es neumático, la fábrica crea una instancia de la clase neumático. Esto permite la creación dinámica de objetos de repuestos, de tal forma que el cliente no conoce los detalles específicos de creación ni del software.

La implementación del patrón Factory Method en este sistema orientado a objetos ofrece varias ventajas. Además, la abstracción del proceso de creación es uno de los puntos más resaltantes. Esta abstracción permite una fácil extensión del sistema con nuevos tipos de productos sin necesidad de modificar el código existente. La encapsulación es otro aspecto importante: los detalles de creación de objetos se encapsulan en la fábrica, lo que oculta la complejidad al cliente y facilita en gran medida el mantenimiento. Debido a estas ventajas, la reutilización se convierte en una práctica efectiva. La fábrica puede reutilizarse en todo el sistema para crear diversos productos, facilitando la reutilización del código.

B. El Código

A continuación, se presenta el código en Python, donde se muestra cómo se pueden crear instancias de diferentes tipos de productos utilizando el Factory Method en la fábrica *FabricaRepuestos*. Cada clase de repuesto implementa los métodos “obtener_descripcion” y “calcular_precio” de la interfaz Producto. Esto permite la creación dinámica de objetos de producto sin conocer los detalles específicos de creación.

```
# Definición de la interfaz Repuesto
# Variables Neumáticos, Radiador, Inyectores
class Repuesto:
  def obtener_descripcion(self):
    pass

  def calcular_precio(self):
     pass

# Clase concreta para productos de Neumáticos
class Neumáticos(Repuesto):
  def obtener_descripcion(self):
    return Neumático: Ring 14, 184, 72"

  def calcular_precio(self):
    return 40.0

# Clase concreta para productos de Radiador
class Radiador(Repuesto):
  def obtener_descripcion(self):
    return "Radiador: Capacidad de agua+fluido refrigerante"

  def calcular_precio(self):
    return 84.0

# Clase concreta para productos de Inyectores
class Inyectores (Repuesto):
  def obtener_descripcion(self):
    return "Inyectores"

  def calcular_precio(self):
    return 80.0

# Venta de repuestos de vehículos que utiliza el Factory Method
class FabricaRepuestos:
  def crear_Repuestos(self, tipo_repuesto):
    if tipo_Repuesto == "Neumáticos":
      return Neumáticos()
    elif tipo_Repuesto == "Radiador":
      return Radiador()
    elif tipo_Repuesto == "Inyectores":
      return Inyectores()
    else:
      raise ValueError("Tipo de producto no válido")

# Uso del Factory Method
fabrica = FabricaRepuestos()

producto_Neumático = fabrica.crear_repuestoto("Neumático")
print(producto_Neumático.obtener_descripcion()) # Salida: Neumático: Ring 14,184, 72
print(repuesto_neumáticoo.calcular_precio())      # Salida: 84.0
```

```
producto_radiador = fabrica.crear_repuesto("Radiador")
print(producto_radiador.obtener_descripcion()) # Salida: Radiador: agua+fluido refrigerante
print(producto_radiador.calcular_precio()) # Salida: 84.0

producto_Inyectores = fabrica.crear_repuesto("Inyectores")
print(repuesto_Inyectores.obtener_descripcion()) # Salida: Inyectores: inyectores
print(producto_Inyectores.calcular_precio()) # Salida: 80.0
```

En el código mostrado, si el programador quiere extender el código, debe introducir más elementos en la variable repuesto, es fácil seguir la sintaxis sin dificultad. A nivel de innovación se puede introducir: i) marca, modelo y año del vehículo para el cual el cliente está comprando el repuesto, ii) un mismo repuesto de distintas marcas y iii) el país de procedencia del repuesto (China, Taiwan, EEUU, Brasil). Otro elemento, a considerar a la hora de extender el programa es la interconexión con otras tiendas de repuestos, tal que, si el comercio consultado inicialmente no lo posee, el código lo direcciona a otro comercio del mismo ramo. Además, se pueden añadir elementos que permitan un control de inventario sin margen de errores.

También, a nivel de innovación, el código se puede extender de tal forma de incluir todos los elementos asociados, por ejemplo, al sistema de inyección o al sistema refrigerante del vehículo, buscando la satisfacción del cliente. Bajo estas premisas el patrón de diseño es innovador, aportando valor agregado a la empresa o comercio.

Los programadores o desarrolladores de software deben ser capaces de conocer ampliamente el problema, las variables relevantes para la empresa, los procesos internos de la empresa viéndola como un sistema donde las variables interactúan de tal forma, que el cliente sienta que prácticamente recibe un trato personalizado, conduciendo a que el propio sistema se autoregule y autocontrole constituyendo en si un sistema de gestión y de información.

CONCLUSIONES

La principal contribución de esta investigación fue mostrar con un ejemplo sencillo cómo el uso de patrones de diseño puede mejorar la flexibilidad del software de una aplicación desarrollada utilizando la programación orientada objeto. La forma sencilla como se muestran las herramientas usadas en el ejemplo, permite que el software sea mucho más flexible a los cambios. Estos cambios se pueden agregar como extensión en lugar de modificar los componentes existentes. Esto reduce significativamente el costo, el tiempo y el esfuerzo para implementar los cambios.

Conocer y utilizar patrones de diseño reviste gran importancia en el desarrollo de software. La aplicación de estos patrones ayuda a abordar los retos más recurrentes en el desarrollo de la programación orientada a objetos, además de ofrecer una mayor productividad y amplio conocimiento a los desarrolladores de software.

Para la innovación en la programación orientada a objetos en la creación de nuevos patrones de diseño, es necesaria la aparición de nuevos frameworks, nuevas plataformas, nuevos tipos de acceso a datos. Estos se someten apruebas de validez y robustez por la comunidad de desarrolladores. Para ello deberá demostrar que es nuevo, que es correcto y que es útil para solucionar problemas de diseño.

La programación orientada a objeto proporciona una base sólida para la representación de objetos del mundo real en el código, promoviendo la reutilización, la modularidad y la flexibilidad. Los principios clave de la orientación a objetos, como la encapsulación, la herencia, el polimorfismo y la abstracción, forman la base sobre la cual se pueden aplicar los patrones de diseño. Se mostró un ejemplo concreto de cómo se aplican estos patrones en un entorno orientado a objetos, desde el uso del patrón Factory Method.

Es importante también innovar en un trabajo futuro, con el la implementación de inteligencia artificial en el diseño de patrones. Herramientas como PatternedAI, se utilizan para generar patrones para productos. En el diseño industrial, la inteligencia artificial influye a través del uso de algoritmos de aprendizaje automático para analizar grandes volúmenes de datos, extraer patrones y tendencias, y sugerir cambios orientados a la funcionalidad y la reducción de costos en el diseño de productos nuevos e innovadores.

REFERENCIAS

[1] R. Subburaj, J. Gladman, C. Hwata "Impact of Object-Oriented Design Patterns on Software Development International", Journal of Scientific & Engineering Research, vol. 6, no. 2, February-2015.
[2] L. Ackerman and C. Gonzalez, "The value of pattern implementations", The World of Software Development Journal, Computer Science, Vol. 32 no. 6, pp. 28-32, 2011.
[3] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[4]. M. Weiss and H. Mouratidis, "Selecting security patterns that fulfill security requirements", 16th International Conference on Requirements Engineering (RE'08), IEEE, 2008, pp. 169–172.
[5] U. Zdun, "Systematic pattern selection using pattern language grammars and design space analysis", Software: Practice & Experience, vol. 37, pp. 983–1016, 2007.
[6] H. Marouane, C. Duvallet, A. Makni, R. Bouaziz, and B. Sadeg, "An UML profile for representing real-time design patterns", Journal of King Saud University-Computer and Information Sciences, vol. 30, no. 4, pp. 478-497, 2018.
[7] M. Aniche, J. W. Yoder and F. Kon, "Current Challenges in Practical Object-Oriented Software Design. In P. Kellenberger (Ed.)", Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER 2019 (pp. 113-116).
[8] J. Bräuer, R. Plösch, M. Saft, and C. Körner, "Measuring object-oriented design principles: The results of focus group-based research", Journal of Systems and Software, vol. 140, pp. 74-90, 2018.
[9] C. Gravino, and M. Risi, "How the use of design patterns affects the quality of software systems: a preliminary investigation", In 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA) 2017, pp. 274-277.
[10] K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani and M. Sharbaf, "A survey of model transformation design patterns in practice", Journal of Systems and Software, vol. 140, pp. 48-73, 2018.
[11] H. Marouane, C. Duvallet, A. Makni, R. Bouaziz, and B. Sadeg, "An UML profile for representing real-time design patterns", Journal of King Saud University-Computer and Information Sciences, vol. 30, no. 4, pp. 478-497, 2018.
[12] M. Ehsan, E. Khonica, W, Wan, M. Azmi and R. Binti, " Impact of Design Principles and Patterns on Software Flexibility: An Experimental Evaluation Using Flexible Point", Journal of Computer Science, vol. 17, no. 7, pp. 624-638, 2021.
[13] J. Bansiya and C. Davis, "A hierarchical model for object-oriented design quality assessment", Transaction on Software Engineering, IEEE Computer Society, Vol. 1, 2002.

[14] J. Bräuer, R. Plösch, M.Saft, and C. Körner.)" Measuring object-oriented design principles: The results of focus group-based research", Journal of Systems and Software, vol. 140, pp.74-90, 2018.
[15] F. Khomh,and Y. G. Guéhéneuc, "Design patterns impact on software quality: Where are the theories?". In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER) 2018, pp. 15-25.
[16] H. Marouane, C. Duvallet, A. Makni, R. Bouaziz, and B. Sadeg, "An UML profile for representing real-time design patterns". Journal of King Saud University-Computer and Information Sciences, vol. 30, no. 4, pp. 478-497, 2018.
[17] K. Lano, S. Kolahdouz-Rahimi, S. Yassipour-Tehrani and M. Sharbaf, "A survey of model transformation design patterns in practice". Journal of Systems and Software, vol. 140, pp. 48-73, 2018
[18] D. Abdullah, M. H. Khan, and R. Srivastava, "Flexibility: A Key Factor to Testability". International Journal of Software Engineering & Applications,(IJSEA), vol. 6, no. 1, 2015
[19] A. H. Eden and T. Mens, "Measuring software flexibility", IEE Proceedings-Software, vol. 153, no. 3, pp. 113-125, 2006.
[20] M. Oruc, F. Akal, and H. Sever, "Detecting design patterns in object-oriented design models by using a graph mining approach". In 2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT), 2016, pp. 115-121.