Polo del Conocimiento

Pol. Con. (Edición núm. 70) Vol. 7, No 7
Julio 2022, pp. 2145-2565
ISSN: 2550 - 682X
DOI: 10.23857/pc.v7i7

Análisis comparativo de Patrones de Diseño de Software

Comparative Analysis of Software Design Patterns

Análise Comparativa de Padrões de Projeto de Software

Oscar Danilo Gavilánez Alvarez [I]
oscar.gavilanez@espoch.edu.ec
https://orcid.org/0000-0002-7245-5640

Natalia Layedra [II]
natalia.layedra@espoch.edu.ec
https://orcid.org/0000-0003-1017-1746

Vinicio Ramos [III]
vinicio.ramos@espoch.edu.ec
https://orcid.org/0000-0003-3033-2404

Correspondencia: oscar.gavilanez@espoch.edu.ec

Ciencias Técnicas y Aplicadas
Artículo de Investigación

*** Recibido:** 23 de mayo de 2022 ***Aceptado:** 12 de junio de 2022 *** Publicado:** 28 de julio de 2022

I. Ingeniero en Sistemas, Magíster en Interconectividad de Redes, Docente Investigador Escuela Superior Politécnica de Chimborazo (ESPOCH), Riobamba, Ecuador.
II. Ingeniera en Sistemas Informáticos, Magíster en Informática Educativa, Docente Investigador Escuela Superior Politécnica de Chimborazo (ESPOCH), Riobamba, Ecuador.
III. Ingeniero en Sistemas Informáticos, Magíster en Interconectividad de Redes, Docente Investigador Escuela Superior Politécnica de Chimborazo (ESPOCH), Riobamba, Ecuador.

Resumen

Los patrones de diseño brindan soluciones a problemas que se presentan durante el desarrollo de software, evitan duplicaciones de código y facilitan su reutilización. En el presente artículo se detallan la estructura, componentes, ventajas y desventajas de los patrones de diseño: Template Method, Model-View-Controller, Model-View-Presenter, Model Front Controller y Model-View-View-Model MVVM. La investigación se realizó a través de una revisión bibliográfica en bases de datos científicas y consecuentemente se determinaron las métricas que permitieron comparar los patrones en estudio. Mediante el análisis comparativo de métricas y parámetros entre los patrones se establece que no existe un patrón superior a nivel general, pues cada patrón tiene su propósito definido y el desarrollador de software es quien debe identificar cuando un patrón se adapta mejor a la solución que desea desarrollar. Se concluye que los patrones de diseño son estructuras bien definidas que permiten mantener una lógica de organización en el código de un sistema, gracias a esto se puede crear software de calidad, con más facilidad de mantenimiento y con una mejor comprensión del código al buscar modularidad en el sistema.

Palabras Clave: Patrones de diseño; patrones arquitectónicos; software; modelo plantilla; MVC; MVP; controlador frontal; MVVM.

Abstract

Design patterns provide solutions to problems that arise during software development, avoid code duplication, and facilitate code reuse. This article details the structure, components, advantages and disadvantages of the design patterns: Template Method, Model-View-Controller, Model-View-Presenter, Model Front Controller and Model-View-View-Model MVVM. The research was carried out through a bibliographical review in scientific databases and consequently the metrics that allowed comparing the patterns under study were determined. Through the comparative analysis of metrics and parameters between the patterns, it is established that there is no superior pattern at a general level, since each pattern has its defined purpose and the software developer is the one who must identify when a pattern is best suited to the solution he wants. develop. It is concluded that design patterns are well-defined structures that allow maintaining an organization logic in the code of a system, thanks to this, quality software can be created, with easier maintenance and with a better understanding of the code when looking for modularity. in the system.

Keywords: Design patterns; architectural patterns; software; template model; MVC; MVP; front controller; MVVM.

Resumo

Padrões de projeto fornecem soluções para problemas que surgem durante o desenvolvimento de software, evitam a duplicação de código e facilitam a reutilização de código. Este artigo detalha a estrutura, componentes, vantagens e desvantagens dos padrões de projeto: Template Method, Model-View-Controller, Model-View-Presenter, Model Front Controller e Model-View-View-Model MVVM. A pesquisa foi realizada por meio de revisão bibliográfica em bases de dados científicas e consequentemente foram determinadas as métricas que permitiram comparar os padrões em estudo. Através da análise comparativa de métricas e parâmetros entre os padrões, estabelece-se que não existe um padrão superior em nível geral, pois cada padrão tem seu propósito definido e o desenvolvedor de software é quem deve identificar quando um padrão é mais adequado para a solução que ele quer desenvolver. Se concluye que los patrones de diseño son estructuras bien definidas que permiten mantener una lógica de organización en el código de un sistema, gracias a esto se puede crear software de calidad, con más facilidad de mantenimiento y con una mejor comprensión del código al buscar modularidad no sistema.

Palavras-chave: Padrões de design; padrões arquitetônicos; Programas; modelo de modelo; MVC; MVP; controlador frontal; MVVM.

Introducción

El gran crecimiento del sector de las tecnologías de la información ha hecho que las prácticas de desarrollo de software evolucionen. Anteriormente se requería completar todo el software antes de realizar pruebas, lo que suponía encontrarse con problemas. Para ahorrar tiempo y evitar volver a la etapa de desarrollo una vez que éste ha finalizado, se introdujo una práctica de prueba durante la fase de desarrollo. Esta práctica se usa para identificar condiciones de error y problemas en el código que pueden no ser evidentes en ese momento.

Con la llegada del desarrollo de software, los profesionales en este campo han buscado una solución para disminuir la complejidad presente en el mantenimiento y organización del código. A través de la experiencia y el análisis de distintos sistemas informáticos se ha observado una serie repetitiva

de elementos que son necesarios para el desarrollo de un sistema, al abstraer estos elementos, se da origen a los patrones de diseño.

Un patrón de diseño es una técnica para resolver problemas simples y comunes que se encuentran en la vida diaria del desarrollo de software, sobretodo constituye una solución respecto al desarrollo de interacciones o interfaces. Se puede definir un patrón como un modelo que explica cómo resolver un determinado problema en el desarrollo de software, no consiste en un conjunto de herramientas que se implementan en el código de una aplicación para resolver el problema. Un patrón es un concepto, es el razonamiento lógico que existe para resolver determinado problema. Aprender a implementar patrones de diseño permite tener el conocimiento para resolver todo tipo de problemas utilizando principios del diseño orientado a objetos.

Con el desarrollo de nuevos patrones, el abanico de posibilidades para estructurar un aplicativo se ha expandido, a tal punto de encontrarse con modelos poco usados o conocidos en la industria de desarrollo. Tener una amplia gama de patrones permite al profesional escoger aquel que mejor se adecúe a la realidad del sistema, sin embargo, se puede volver complicado comprender las peculiaridades de cada uno, dificultando su proceso de selección. En definitiva, los patrones de diseño aseguran la validez del código, ya que son soluciones que funcionan y han sido probados por muchos desarrolladores siendo menos propensos a errores.

Los patrones de diseño de software permiten afrontar proyectos de forma eficiente, puesto que ayudan a elegir opciones de diseño que hacen que un sistema sea reutilizable, además, mejoran sustancialmente el proceso de documentación y mantenimiento.

El artículo documenta un análisis comparativo para evaluar las características de cinco patrones de diseño de software: Template Method (Modelo Plantilla), Model-View-Controller (Modelo Vista Controlador MVC), Model-View-Presenter (Modelo Vista Presentador MVP), Model Front Controller (Modelo Controlador Frontal) y Model-View-ViewModel (Modelo Vista Vista Modelo MVVM).

Revisión de la literatura

Patrones de Diseño

Se entiende por patrón de arquitectura de software a aquellas reglas que determinan el contexto bajo el cual se llevará a cabo el desarrollo, estas reglas tienen comof inalidad la obtención de las características esperadas del software en cuestión. Los patrones arquitectónicos de software

definen un enfoque específico para el manejo de alguna característica de comportamiento del sistema [1].

Un patrón de diseño es una solución estándar a un problema, los patrones de diseño definen y organizan un vocabulario de elementos básicos en un conjunto de componentes y conectores de tal manera que se garanticen una adecuada composición de los elementos que lo conforman según el esquema de organización asociado [2].

Si al querer desarrollar una aplicación robusta y fácil de mantener sabemos cumplir con ciertas reglas podemos usar patrones de diseño que son recomendables, pero no obligatorios. Esto debido a que se establece un lenguaje común entre el equipo de desarrollo; los patrones de diseño están ampliamente documentados y testados y ayudarán a todo el equipo a comprender lo implementado, cómo y por qué.

Entre los beneficios que ofrecen los patrones se puede mencionar: reducción de tiempos, disminución del esfuerzo de mantenimiento, aumento de la eficiencia, aseguramiento de la consistencia, aumento de la fiabilidad y protección de la inversión en desarrollos.

Existen diferentes tipos de patrones de arquitectura cada una con sus particularidades, mismas que muchas de las veces propician el nombre a éstos, entre los más distinguidos se encuentran:

- Template Method
- Model-View-Controller MVC
- Model-View-Presenter MVP
- Model Front Controller
- Model-View-View-Model MVVM

A) Patrón Template Method - Modelo Plantilla

El código duplicado es el gran enemigo del “código limpio”. Existe un amplio abanico de recursos para interntar eliminar este problema. Algunos patrones de diseño tienen el objetivo de evitar que el código se duplique y que se pueda aplicar buenas prácticas de programación.

Template Method es un patrón que define el esqueleto de un algoritmo en una operación, difiriendo algunos pasos a las subclases. Template Method permite a las subclases redefinir ciertos pasos de un algoritmo sin cambiar la estructura de este [3].

Es un patrón de diseño de comportamiento donde se define el esqueleto de un algoritmo en la superclase, mientras que, las subclases pueden sobrescribir los pasos del algoritmo sin la necesidad

de cambiar su estructura. Es un patrón de diseño de comportamiento donde se define el esqueleto de un algoritmo en la superclase, mientras que, las subclases pueden sobrescribir los pasos del algoritmo sin la necesidad de cambiar su estructura [4].

El objetivo del patrón Template Method nace de la necesidad de extender determinados comportamientos dentro de un mismo algoritmo por parte de diferentes entidades. Es decir, diferentes entidades tienen un comportamiento similar pero que difiere en determinados aspectos puntuales en función de la entidad concreta [2].

El patrón Template Method tiene como objetivos [5]:

- Proporcionar un esqueleto para un método permitiendo que las subclases redefinan partes específicas del método.
- Centralizar las partes de un método que se definen en las subclases, pero que poseen una mínima diferencia en cada subclase.
- Controlar las operaciones, mismas que deben ser redefinidas en las subclases.

La figura 1 muestra los componentes que forman parte del patrón de diseño de comportamiento Template Method [5].

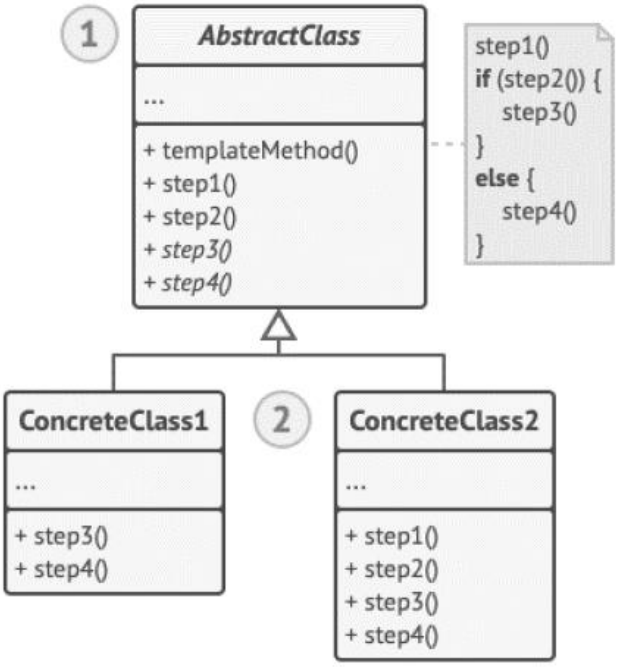


Figura 1. Estructura Template Method

- Clase Abstracta: En esta clase se declaran métodos, lo cuales actúan como pasos de un algoritmo y también el propio método “templateMethod()”, el cual invoca a los otros métodos en un orden específico. Los pasos que se coloquen pueden ser declarados como abstractos o disponer de una implementación por defecto.
- Clases Concretas: Estas clases pueden sobrescribir todos los pasos declarados anteriormente en la clase abstracta, pero no el método “templateMethod()”.

a) Ventajas

- Evita duplicación de código: Se puede obtener un código sin duplicaciones, dado que, el código que varía entre las distintas subclases permanece en cada una.
- Código reutilizable: Se puede reutilizar el código, ya que utiliza herencia.
- Es fácil de entender e implementar: El patrón es fácil de utilizar y además proporciona un código legible.
- Flexible: Es flexible porque el patrón permite que las subclases decidan cómo implementar los pasos del algoritmo.

b) Desventajas

- Comprender y depurar la secuencia de flujo del patrón puede ser confuso, ya que puede darse el caso de que se implemente un método que no debía implementarse, o, a su vez, no implementar un método abstracto.
- Realizar el mantenimiento del framework plantilla puede resultar difícil, porque al realizar cambios, ya sean, de alto o bajo nivel pueden afectar en la implementación.
- Debido a que el patrón aplica un diseño en particular, algunos desarrolladores pueden verse limitados a la hora de programar.

El patrón Template Method es aconsejable en los siguientes supuestos [6]:

• Cuando se cuenta con un algoritmo aplicable a varias situaciones, cuya implementación difiere únicamente en algunos pasos.

• Arquitecturas donde los pasos de un proceso están definidos (el qué), pero sea necesario establecer los detalles sobre cómo realizarlos (el cómo).

• Módulos en los que exista una gran cantidad de código duplicado que pueda ser factorizado en pasos comunes

B) Model-View-Controller MVC

El MVC es un patrón de diseño arquitectónico de software, que sirve para clasificar la información, la lógica del sistema y la interfaz que se le presenta al usuario. En este tipo de arquitectura existe un sistema central o controlador que gestiona las entradas y la salida del sistema, uno o varios modelos que se encargan de buscar los datos e información necesaria y una interfaz que muestra los resultados al usuario final [7].

El patrón Model View Controller posee los siguientes componentes [8]:

- Modelo: este componente se encarga de manipular, gestionar y actualizar los datos. Si se utiliza una base de datos aquí es donde se realizan las consultas, búsquedas, filtros y actualizaciones.
- Vista: este componente se encarga de mostrarle al usuario final las pantallas, ventanas, páginas y formularios; el resultado de una solicitud. Desde la perspectiva del programador este componente es el que se encarga del frontend; la programación de la interfaz de usuario si se trata de una aplicación de escritorio, o bien, la visualización de las páginas web (CSS, HTML, HTML5 y Javascript).
- Controlador: este componente se encarga de gestionar las instrucciones que se reciben, atenderlas y procesarlas. Por medio de él se comunican el modelo y la vista: solicitando los datos necesarios; manipulándolos para obtener los resultados; y entregándolos a la vista para que pueda mostrarlos. La figura 2 muestra la estructura del patrón MVC

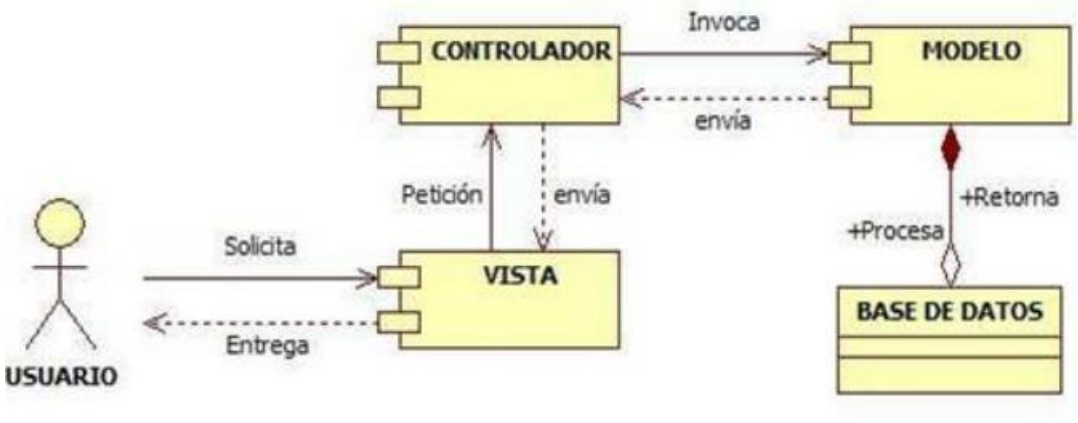


Figura 2. Estructura del patrón MVC

Las ventajas y desventajas que presenta el patrón modelo vista controlador [3] son:

a) Ventajas

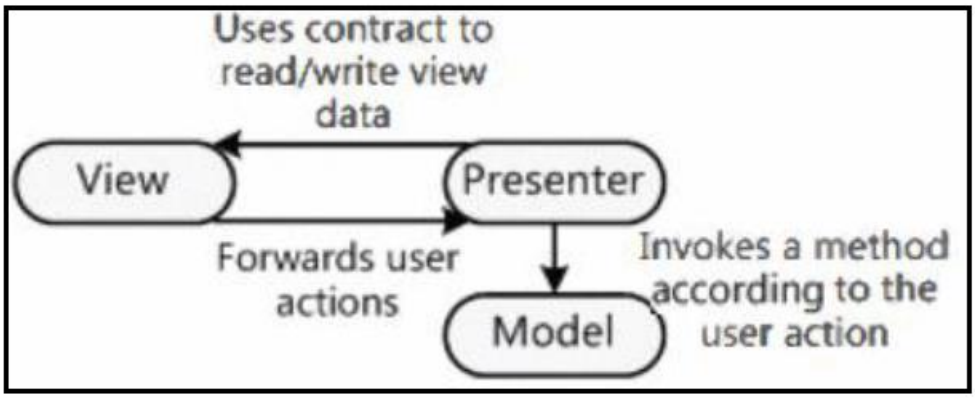
- Separa las componentes (interfaz de usuario – lógica de negocio – acceso a datos) y permite la implementación de cada uno por separado, por lo tanto, en caso de que algún componente falle se puede modificar fácilmente.
- Se puede construir varias vistas para un único modelo sin necesidad de modificarlo
- Facilidad para la realización de pruebas unitarias
- Muchos frameworks utilizan este patrón como parte de su arquitectura.
- Permite controlar de mejor manera los recursos del servidor, evita errores que puedan repercutir en el rendimiento.

b) Desventajas

- Es necesario el desarrollo de una gran cantidad de clases, ya que normalmente para cada entidad se construye un controlador.
- Su implementación es difícil si se realiza desde un lenguaje que no es compatible con el paradigma orientado a objetos.
- La separación en capas hace que la aplicación sea más compleja.
- Curva de aprendizaje alta.

C) Model-View-Presenter MVP

MVP es un patrón arquitectónico para la capa de presentación de las aplicaciones software. El patrón fue desarrollado por Taligent en la década de 1990 y se implementó por primera vez en los lenguajes de programación C++ y Java. MVP se basa en los principios del patrón Modelo-Vista-Controlador (MVC) desarrollado por Xerox PARC a finales de 1970. La figura 3, muestra los componentes del MVP [10].



Pol. Con. (Edición núm. 70) Vol. 7, No 7, Julio 2022, pp. 2145-2565, ISSN: 2550 - 682X

Figura 3. Componentes MVC

- El modelo: es el componente que almacena los datos y la lógica de negocio; sólo expone un conjunto de interfaces de programación para conectarse con el presentador, es decir, oculta los detalles internos de implementación.
- La vista: es la interfaz de usuario, que se encarga de recibir peticiones del usuario, las misma son transportadas al presentador para ser procesadas y finalmente el resultado es presentado por las vistas mediante interfaces de programación.
- El presentador: es el componente intermediario entre la vista y el modelo; recibe las peticiones del usuario e invoca métodos para extraer información del modelo. A continuación, obtiene el resultado y actualiza la vista.

Las ventajas del patrón Model View Presenter [11] son:

- Existe un bajo acoplamiento entre el modelo y la vista. Eso significa que, si el modelo o la vista se cambian, no se necesita realizar más modificaciones. Esto representa flexibilidad en la arquitectura.
- El presentador ignora cualquier tecnología para el diseño detrás de la vista, permitiendo la sustitución de tecnologías sin afectar al resto de la arquitectura.
- La vista y el modelo pueden ser testeados de manera independiente. De manera tradicional es imposible probar la vista o el componente de lógica de negocio de manera independiente por el acoplamiento que existe entre los dos. En consecuencia, las pruebas unitarias para la vista o el componente de lógica de negocio son difíciles. Todos estos problemas se resuelven con el patrón MVP, debido a que no hay dependencia directa entre la vista y el modelo. Por esa razón, el desarrollador puede utilizar un objeto falso para inyectar en la vista o el modelo para que puedan ser probados por uno mismo.

Entre las desventajas [12] se pueden mencionar:

- Es compleja su implementación, por lo tanto, es importante que el desarrollar tenga el conocimiento técnico.
- No es adecuado para soluciones simples y pequeñas

D) Model Front Controller

El Front Controller es un patrón de diseño fácil de entender en el que tiene un controlador principal que maneja cada solicitud de un sitio web. Es un patrón de diseño de uso común para muchas

aplicaciones web basadas en MVC. Para usar el patrón de controlador frontal, debe tener una sola parte de su sitio web que sea completamente responsable de manejar todas las solicitudes entrantes a su sitio/aplicación. A menudo (pero no siempre) funcionará con un sistema de enrutamiento y plantillas para devolver una respuesta relevante a la solicitud [6].

Front Controller es un patrón de diseño que se centra en el manejo de peticiones, usando como punto inicial un controlador que realiza la gestión de solicitudes, entre las gestiones que efectúa se encuentran: comprobación de seguridad, manejo de errores, mapeo y delegación de solicitudes a los demás componentes de la aplicación, y así poder trabajar en una vista adecuada para los usuarios [13].

Componentes que forman parte del patrón de diseño Front Controller [14]:

- Controller: es el punto central donde se procesan y se gestionan las peticiones dentro del sistema. Existe funciones que pueden ser delegadas al asistente como: completar la autenticación, iniciar la recuperación de contacto o brindar autorizaciones a los usuarios.
- Dispatcher: se encarga de gestionar la vista y navegación, brindando el mecanismo para el control de la siguiente vista que se quiera mostrar.
- Helper: ayuda al controlador y a la vista a terminar su procesamiento, poseen varias responsabilidades y pueden adaptar el modelo de datos para que la vista pueda usarla.

En la figura 4 se observa cómo se estructura el patrón de diseño Front Controller.

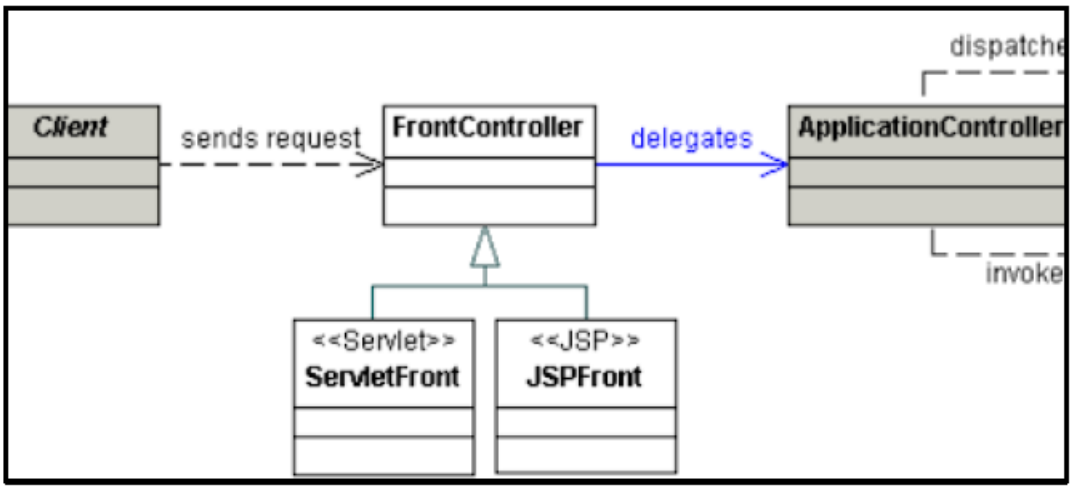


Figura 4. Estructura del patrón Front Controller

Las ventajas que presenta el patrón Front Controller [15] son las siguientes:

Pol. Con. (Edición núm. 70) Vol. 7, No 7, Julio 2022, pp. 2145-2565, ISSN: 2550 - 682X

- Existe una mejora significativa en la manejabilidad de la seguridad, ya que, proporciona una barrera para controlar los intentos de acceso ilícitos en la aplicación.
- Es posible reutilizar código, ya que, el controlador proporciona el espacio para el particionamiento limpio de la aplicación, tomando en cuenta que el código entre componentes puede ser común.
- Evita tener distribuida la gestión de peticiones, con lo cual, las aplicaciones diseñadas en base al patrón Front Controller no serán categorizadas como de baja calidad.
- No limita el número de controladores en el sistema como lo hacen otros patrones.

Según [16] Front Controller posee las siguientes ventajas:

• Control centralizado: el controlador frontal maneja todas las solicitudes a la aplicación web. Esta implementación de control centralizado que evita el uso de múltiples controladores es deseable para hacer cumplir las políticas de toda la aplicación, como el seguimiento y la seguridad de los usuarios.

• Seguridad de subprocesos: surge un nuevo objeto de comando cuando se recibe una nueva solicitud y los objetos de comando no están destinados a ser seguros para subprocesos. Por lo tanto, estará seguro en las clases de comando. Aunque la seguridad no está garantizada cuando se recopilan problemas de subprocesamiento, los códigos que actúan con el comando siguen siendo seguros para subprocesos

Las desventajas que se describen para el patrón Front Controller [15] son:

- Es mucho más difícil escalar una aplicación que trabaja con el patrón de diseño Front Controler.
- La velocidad de respuesta disminuye al tener que ser procesadas las peticiones primero por el controlador.
- Obliga a los desarrolladores a ver la aplicación desde el mismo enfoque de los usuarios.
- Depende de la aplicación web a partir de su entorno de alojamiento.

E) Model-View-ViewModel MVVM

El Patrón Modelo Vista Vista Modelo (MVVM en sus siglas del Ingles) permite aislar limpiamente la lógica de negocios y presentación de una aplicación de su interfaz de usuario (UI), permitiendo mitigar problemas de desarrollo y facilitar los procesos de prueba, mantenimiento y escalado de una aplicación software. Por otro lado, permite reutilizar el código, además, desarrolladores y

diseñadores colaboran de forma eficiente al desarrollar los respectivos módulos de una aplicación [17].

Model View ViewModel o MVVM es un patrón de diseño desarrollado como alternativa al patrón MVC. Este patrón a diferencia de otros busca desligar a la vista del modelo con el fin de facilitar las pruebas unitarias en el sistema, manteniendo los principios generales de programación modular [18].

Los componentes que forman parte del patrón de diseño Model View ViewModel [19] son:

- Modelo: Es el responsable del acceso a la fuente de datos y de trabajar con esos datos.
- Vista: Representa los datos de forma pertinente, reflejando el estado de los datos y recibiendo los eventos y las interacciones del usuario.
- Vista del modelo: Responsable de representar la forma en que se espera que sea una vista y como se comportará ante las interacciones que tiene con el usuario. Además, describe el conjunto de principios y estructuras que presentan los datos recuperados del modelo. Como último trabajo, es el puente de comunicación entre el modelo y la vista. La figura 5 muestra los componentes del patrón de diseño Model View ViewModel

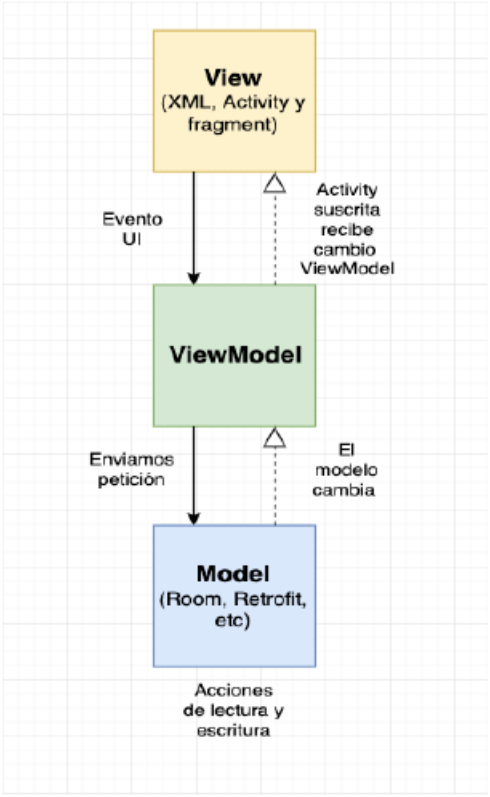


Figura 5. Componentes patrón de diseño Model View ViewModel

Pol. Con. (Edición núm. 70) Vol. 7, No 7, Julio 2022, pp. 2145-2565, ISSN: 2550 - 682X

Las ventajas que presenta el patrón Model View ViewModel [20] son las siguientes:

- Facilita el mantenimiento del sistema al mantener una alta modularidad al desligar de manera directa tanto a la vista como al modelo.
- Disminuye la cantidad de código escrito tanto en los componentes de la vista como del modelo.
- Facilita el proceso de pruebas unitarias de cada componente al desligar a la vista del modelo completamente.

Las desventajas que se describen para el patrón Model View ViewModel [20] son:

- Adaptarse a una estructura predefinida puede resultar complicado sobre todo para interfaces de usuario simples.
- La curva de aprendizaje para nuevos desarrolladores puede resultar más compleja que la de otros modelos más simples.
- La depuración del sistema puede resultar difícil cuando se tiene enlaces de datos complejos y un extenso número de ficheros.

Metodología

Para realizar el análisis comparativo que permitió evaluar las características de los cinco patrones de diseño de software: Template Method, Model-View-Controller MVC, Model-View-Presenter MVP, Model Front Controller y Model-View-ViewModel MVVM se desarrolló inicialmente la respectiva revisión bibliográfica, la información acerca de cada uno de los patrones fue extraída de base de datos científicas para localizar artículos de investigación, tanto de fuentes primarias como secundarias.

Se introdujeron las palabras clave “patrones de diseño”, “patrones arquitectónicos” y “software” en el cuadro de búsqueda para generar los artículos sobre el tema, los artículos resultantes generados en la lista de búsqueda se filtraron según el año de publicación y se determinó la relevancia de los resultados para el tema en cuestión, seleccionando 20 artículos para la revisión.

Consecuentemente en base a las características, ventajas y desventajas de cada uno de los patrones de diseño se determinaron métricas que permitan comparar los mismos.

Resultados y Discusión

Los patrones poseen ciertas características en común, Template Method es un patrón de diseño de comportamiento y demás patrones de diseño estructural. Los patrones poseen propiedades bien

definidas y tienen como propósito mejorar la manera de estructurar y codificar la aplicación, por lo tanto, de los resultados obtenidos a pesar de las diferencias que poseen, se puede afirmar que no existe un patrón superior a nivel general, pues cada patrón tiene su propósito definido y el desarrollador de software es quien debe identificar cuando un patrón se adapta mejor a la solución que desea desarrollar. La tabla 1 muestra la comparativa entre patrones de diseño de software.

Tabla 1. Comparativa entre patrones de software

Métrica	**Template Method**	**Model-View-Controller MVC**	**Model-View-Presenter MVP**	**Model Front Controller**	**Model-View-ViewModel MVVM**
Escalabilidad de la aplicación	Medio	Alto	Alto	Bajo	Alto
Mantebilidad de la aplicación	Medio	Alto	Alto	Alto	Alto
Acoplamiento con los módulos	Medio	Bajo	Bajo	Bajo	Bajo
Facilidad de implementación	Alto	Bajo	Bajo	Bajo	Bajo
Compatibilidad con programación modular	Alto	Alto	Alto	Alto	Alto
Facilidad para testing	Alto	Alto	Alto	Alto	Alto
Compatibilidad con multiparadigmas de programación	Bajo	Bajo	Bajo	Bajo	Bajo

Pol. Con. (Edición núm. 70) Vol. 7, No 7, Julio 2022, pp. 2145-2565, ISSN: 2550 - 682X

Basado en arquitecturas multicapa	No aplica	Alto	Alto	Alto	Alto
Reutilización de código	Alto	Alto	Alto	Alto	Alto

La definición de parámetros para la comparativa entre patrones de diseño se muestra en la tabla 2; se describen a continuación los parámetros:

Lenguajes de Programación: Los patrones de diseño son independientes del lenguaje de programación, sin embargo, se analiza este parámetro para determinar qué lenguajes de programación son más comunes en la utilización de cada patrón.

Complejidad: Permite definir cuán complejo es el aprendizaje y utilización del patrón de diseño.

Seguridad: Determina el nivel de seguridad que proporciona el patrón al diseñar el software.

Uso: Ámbito de desarrollo de software que utiliza el patrón de diseño

Tabla 2. Comparativa entre patrones de diseño

Parámetros	**Template Method**	**Model-View-Presenter MVP**	**Model Front Controller**	**Model-View-ViewModel MVVM**
Lenguajes de Programación	PHP C++ Java	.Net Java PHP	Java PHP Python Ruby Raku	.Net JavaScript C++
Complejidad	Baja	Baja	Alta	Alta
Seguridad	-	Baja	Alta	Baja
Uso	Frameworks	Aplicaciones Android	Frameworks Web	Windows y Gráficos Web

Cada uno de los patrones posee características distintas entre parámetros, comparativa presentada permitirá a desarrolladores escoger un patrón de diseño según sus necesidades, pues cada patrón es fuerte en un campo distinto.

Conclusiones

El patrón Template Method facilita la reutilización de código, por eso es fundamental en muchos Frameworks, se vuelve de especial utilidad cuando es necesario realizar un algoritmo que sea común para muchas clases, pero con pequeñas variaciones entre una y otras.

El patrón Template Method es uno de los patrones más conocidos en el mundo de la programación por su gran ayuda al momento de reescribir código para reutilizarlo en un mismo proyecto, ayudando así optimizar el tiempo del programador y agilizando procesos para brindar un software mucho más liviano y efectivo.

El patrón Model-View-Controller es el más empleado en todo el mercado del desarrollo, además, la estructura es fácil de implementar ya que solo se maneja en los componentes vista y un controlador, además, existen muchas derivaciones de este patrón como por ejemplo MVVM, MVP entre otros. MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación.

El patrón Front Controller implementa un controlador único y lo usa como el punto inicial de contacto para manejar las peticiones. No es posible escalar una aplicación utilizando un patrón Front Controller.

MVP permite separar la interfaz de la lógica en Android de forma sencilla evitando que las actividades terminen degradando en clases muy acopladas que consisten en cientos o incluso miles de líneas. En aplicaciones grandes, es esencial organizar bien el código. De lo contrario, se hace imposible mantenerlas y extenderlas. MVP consume bastante memoria debido a que depende de los atributos, además, crea un atributo por cualquier necesidad.

El patrón MVVM ayuda a separar la lógica de las vistas de manera que se pueda tener una alta mantenibilidad en relaciones, además de brindarle al programador un enfoque mayor a la lógica en vez de programar las vistas a un nivel más complejo. Se recomienda utilizar este patrón si se tiene experiencia trabajando con patrones similares como MVC, caso contrario la complejidad puede llegar a ser mayor. En MVVM no es factible la reutilización de código en presencia de enlaces de datos bidireccionales.

Los patrones de diseño son estructuras bien definidas que permiten mantener una lógica de organización en el código de un sistema, gracias a esto se puede crear software de calidad, con más facilidad de mantenimiento y con una mejor comprensión del código al buscar modularidad en el sistema.

Referencias

1. Pressman, R. Ingeniería del Software: Un enfoque práctico (2010). McGrawHill.
2. INGAR - Instituto de Desarrollo y Diseño, M. J. Blas, H. Leone, INGAR - Instituto de Desarrollo y Diseño, S. Gonnet, y INGAR - Instituto de Desarrollo y Diseño, «Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube», risti, n.o 35, pp. 1-17, dic. 2019, doi: 10.17013/risti.35.1-17.
3. E. Gamma, R. Helm, R. Johnson, y J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.
4. A. Shvets, Sumergete en los patrones de diseño. 2019.
5. J. E. McDonough, «Object-Oriented Design with ABAP», Object-Oriented Des. with ABAP, 2017.
6. ORACLE, «Core J2EE Patterns - Front Controller», 2022. https://www.oracle.com/java/technologies/front-controller.html
7. A. Sunardi y Suharjito, «MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based», Procedia Comput. Sci., vol. 157, pp. 134-141, ene. 2019.
8. C. Giridhar, Learning Python design patterns : leverage the power of Python design patterns to solve real-world problems in software architecture and design. 2016.
9. R. Jiménez, «Utilización de la arquitectura Modelo - Vista – Controlador (MVC) en el desarrollo de una aplicación web de catálogos privados.», Ambato, 2017.
10. J. M. Keller, «The MVP Model: Overview and Application», New Dir. Teach. Learn., vol. 2017, n.o 152, pp. 13-26, dic. 2017.

11. S. Paul, A. Chatterjee, y D. Guha, «Study of Smart Inventory Management System Based on the Internet of Things (Iot)», IJRTBT Int. J. Recent Trends Bus. Tour. |, vol. 3, n.o 3, pp. 27-34, 2019.

12. G. Carrera y J. Germania, «Análisis comparativo de la productividad entre los patrones de dieño Modelo Vista Controlador (MVC) y Modelo Vista Presentador (MVP) aplicado al desarrollo del Sistema Nómina de Empleados y Rol de Pagos de la Distribuidora Soria C.A.», 2014.

13. N. Almazova, A. Rubtsova, E. Krylova, y A. Almazova-ilyina, «BLENDED LEARNING AS THE BASIS FOR SOFTWARE DESIGN.», 2019. [En línea]. Disponible en: https://go.gale.com/ps/i.do?id=GALE%7CA627003485&sid=googleScholar&v=2.1&it=r&linkaccess=abs&issn=17269679&p=AONE&sw=w&userGroupName=anon~249b91cf. [Accedido: 08-jun-2022].

14. G. Santana Franco, «Entorno de usuario para una aplicación 'Fintech': Finbook», 2020.

15. S. Fontan Llamas, «Construcción de un sitio web para KV Ingeniería de Tecnología e Infraestructuras - Archivo Digital UPM», 2019.

16. S. Kumar, "Front Controller Design Pattern," Geeks for Geeks, 2020. https://www.geeksforgeeks.org/front-controller-design-pattern/.

17. Microsoft, "El patrón Model-View-ViewModel - Xamarin | Microsoft Docs," 2017. https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

18. G. Hurtado y H. Ramos, «Implementación de sistema Web para optimizar los procesos de negocio en la empresa MN Catering Sánchez, Los Olivos - 2013», Universidad de Ciencias y Humanidades, 2017.

19. G. Arcos-Medina, J. Menéndez, y J. Vallejo, «Comparative Study of Performance and Productivity of MVC and MVVM design patterns», KnE Eng., vol. 1, n.o 2, p. 241, ene. 2018.

20. C. Loor, «Desarrollo e implementación de un sistema para la gestión y control de los recursos utilizados en los proyectos de investigación de naturaleza estadística», 2015.