INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

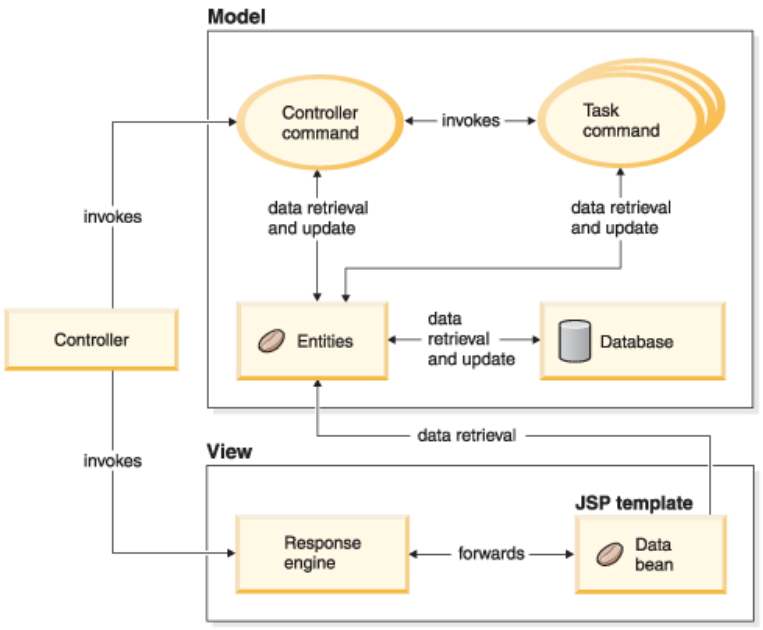
Tarea 2 - Ensayo: Patrones de Diseño

Unidad de aprendizaje: Web Application Development

Grupo: 3CM6

Alumnos(a):
Ramos Diaz Enrique

Profesor(a):
Montes Casiano Hermes Francisco



22 de febrero 2019

Índice

1 Introducción **2**

2 Desarrollo **2**

3 Conclusión **4**

4 Bibliografía **4**

1. Introducción

En la actualidad, dentro de la rama de la ingeniería de software, la gran mayoría de equipos de trabajo que se dedican a analizar, diseñar e implementar proyectos que requieran software utilizan el tan famoso paradigma de programación orientado a objetos.

Es bien sabido que por medio de los lenguajes cuya esencia esta basada enteramente en este paradigma, los desarrolladores de software tienen más posibilidades y facilidades de abstraer problemas del mundo real al mundo informático, y resolverlos por medio de la programación.

Otro punto de vista es que sin las adecuadas prácticas de reutilización de código al programar, que podemos ver en conceptos como herencia o polimorfismo, los equipos de trabajo están perdidos en el limbo de la implementación.

Desde mi punto de vista, el proceso más complicado en el desarrollo completo de una aplicación, es justo analizar el entorno, con variables tan complejas y sujetas a muchos cambios como lo son sus problemas, necesidades, objetivos y soluciones existentes, para darles sentido por medio de simples líneas de código.

No obstante, superando ésta etapa de interpretación de la situación, estando ya lista para su diseño e implementación, no podemos evitar caer en las mismas complicaciones de siempre: ¿cómo mitigar aquellos problemas que ni siquiera en el mundo real han podido ser solucionados?

Afortunadamente, existen los llamados patrones de diseño. En éste ensayo se explicará la propuesta que éstos ofrecen, así como las ventajas dentro de la ingeniería de software en general.

2. Desarrollo

Podemos definir a un patrón de diseño como el esqueleto para solución a un problema que en repetidas ocasiones ha surgido a lo largo de la historia de la programación, y que demás es reusable en distintas circunstancias.

La diferencia entre un programador experto de uno inexperto es la identificación del problema, y el conocimiento de cuándo y cómo aplicar el patrón de diseño especifico para superarlo satisfactoriamente.

El uso de interfaces en la programación orientada a objetos se ve muy facilitado para el programador cuando se aplican patrones de diseño, identificando sus elementos clave y las relaciones las demás y las clases del sistema en conjunto.

Aunque sin duda, opino que lo más importante es la gran ventaja de agregar nuevos elementos o componentes en un futuro, sin afectar a ningún componente crítico del sistema una vez puesto en producción, en caso de requerir un ajuste o actualización, dándole el atributo de mantenibilidad al sistema de forma sencilla.

Esto es algo que van a agradecer mucho los programadores encargados de dar mantenimiento a un software o sistema en el que ellos no participaron.

Podemos clasificar a los patrones de diseño en tres grandes grupos, que son los aceptados por la comunidad de desarrollo de software como imprescindibles:

- **Creacionales:** Sirven para configurar e inicializar objetos.
  - Abstract Factory: Muy utilizado para crear elementos dentro de interfaces gráficas de una forma rápida y cómoda.
  - Builder: Para crear distintos objetos con el mismo proceso.
  - Factory Method: Las subclases son encargadas de escoger la clase a implementar.
  - Prototype: Crear o copiar un objeto a partir de otro.
  - Singleton: Instanciar objetos una única vez.
- **Estructurales:** Separan las interfaces de la implementación.
  - Adapter: Insertar un nuevo componente a un sistema sin alterarlo significativamente.
  - Bridge: Separar la abstracción de la implementación, para modificar alguna sin alterar a la otra.
  - Composite: Construir objetos complejos a partir de otros más simples.
  - Decorator: Añadir funcionalidades extras a una clase sin necesidad de herencia.
  - Facade: Simplificar interfaces.
  - Flyweight: Compartir objetos en el sistema sin repetirlos o crearlos de nuevo.
  - Proxy: Proveer de obligaciones de acceso de un objeto a otro.
- **Comportamiento:** Establecen la comunicación entre objetos.
  - Chain of responsability: Permitir que más de un objeto atienda la misma petición.
  - Command: Encapsula la petición como un objeto, teniendo la posibilidad de desecharla.
  - Interpreter: Construye la interpretación de la gramática de un lenguaje.
  - Iterator: Acceder a los elementos de una serie de objetos sin exponer su estructura interna.
  - Mediator: Coordina la relación entre objetos, sin la necesidad de unirlos estrechamente.
  - Memento: Guarda el estado de un objeto.
  - State: Simula el cambio de clase en un objeto.

Volviendo a los patrones de diseño, hay uno que olvide mencionar, no por su nula importancia, sino todo lo contrario, porque merece un apartado especial para hablar de él: **el Modelo - Vista - Controlador (MVC).**

Siempre hemos escuchado o nos han dicho que tratemos de imaginar la composición de un software por medio de capas, como si de una cebolla se tratara, que se coordinan para otorgarle la funcionalidad completa a una aplicación, pero a su vez son independientes entre sí.

El MVC nos dice que estas capas son tres:

1. **Modelo:** Representa la realidad, es decir, toda la información del contexto en que se aplica o trabaja la aplicación. El ejemplo clásico sería la base de datos, aunque pueden estar almacenados todos los métodos o algoritmos de procesamiento, resultados de una entrada y que crean una salida.

2. **Vista:** Todo los elementos que el usuario ve directamente y con los que interactúa; por medio de ella se ingresa, accede y devuelve información de la capa de Modelo, pero no hace procesamiento de ella.

3. **Controlador:** Es la conexión o el puente entre las dos capas anteriores. Lee las peticiones o solicitudes del usuario, reguladas por medio de la capa de Vista, y ejecuta la reacción o resultado esperado, apoyándose o consultando la capa de Modelo, ya que conoce a esta ultima en su totalidad. Es el intérprete de lo que el usuario quiere que haga la aplicación.

Con ayuda de éste patrón de diseño, nosotros como creadores de software podemos separar los datos en distintos objetos, para evitar tener concentración de éstos en un solo lugar, corrupción total del sistema, falla completa del sistema, mantenimiento tedioso, actualizaciones que más bien parecen rediseños completos del software, accesos no autorizados, y vulnerabilidades de seguridad, entre otras cosas.

3. Conclusión

En la actualidad, hay mucho trabajo ya implementado en el desarrollo de software, sin olvidar todas aquellas normas y conceptos de la ingeniería de software, que son como el plano que nos llevará a la creación exitosa y sin líos de aplicaciones.

Los patrones de diseño ofrecen miles de soluciones para cualquier problema que pueda surgir, experiencia a programadores novatos, buenas prácticas de programación, mantenimiento amigable, y sobretodo, nos ayudan a reducir tiempos de desarrollo y a minimizar costos.

Además de delegar funciones entre el equipo de trabajo, asignando módulos de una capa específica que no interfiera con el desarrollo de otra, para al final simplemente integrar todas las partes en conjunto incluso con ayuda de otro patrón de diseño, como los estructurales y creacionales.

Más aún cuando estos consisten en docenas de personas, que deben organizarse, planificar, y llegar a la culminación de la solución en conjunto. Sin contar aspectos como la solidez, calidad, seguridad, compatibilidad, y demás atributos que determinan sí un software es o no robusto y funcional.

4. Bibliografía

[1] (2008) ¿Qué es un Patrón de Diseño?. Accessed february 2019. [Online]. Available: `https://docs.microsoft.com/es-es/previous-versions/bb972240(v=msdn.10)`

[2] (2019) Patrones de Diseño. Accessed february 2019. [Online]. Available: `https://sites.google.com/site/yorkfv/patrones-disenno`

[3] (2019) Patrón de diseño de modelo-vista-controlador. Accessed february 2019. [Online]. Available: `https://www.ibm.com/support/knowledgecenter/es/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.htm`

[4] (2019) Design Patterns: Elements of Reusable Object-Oriented Software. Accessed february 2019. [Online]. Available: `http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf`