Capítulo 1

Análisis de Sistemas Software

Mónica Elizabeth Páez Padilla

Resumen

En este capítulo abordaremos varios temas relacionados con el "Análisis de Sistemas Software". En este apartado, exploraremos conceptos como la definición del análisis de sistemas software, su evolución a lo largo del tiempo, los estándares y principios que guían este proceso. Además, también se abordará el análisis de sistemas de inteligencia artificial, explorando cómo estos sistemas también requieren un enfoque analítico.

Palabras claves: software, historia, inteligencia artificial.

Páez Padilla, M.E. (2023). Análisis de Sistemas Software. En M.E. Páez Padilla (ed). *Análisis y diseño de sistemas.* (pp. 27-84). Religación Press. http://doi.org/10.46652/religacionpress.89.c82

1.1 Definición

El proceso de análisis del sistema de software juega un papel importante en el desarrollo de aplicaciones informáticas y sistemas tecnológicos. Su principal objetivo es comprender, documentar y definir los requisitos, funciones y limitaciones de un sistema de software antes de su implementación. Este proceso garantiza que los requisitos del usuario y los objetivos comerciales se traduzcan con precisión en diseños técnicos que los desarrolladores puedan implementar (Kendall & Kendall, 2005).

Por lo general, el análisis de sistemas de software implica una serie de pasos y actividades que incluyen:

Recolección de requisitos: Durante esta fase, se recopila información minuciosa sobre las necesidades de los usuarios, los propósitos empresariales y las funciones necesarias. Esto se logra mediante entrevistas, encuestas, reuniones y otros enfoques de interacción con las partes interesadas (Domínguez Coutiño, 2012).

Análisis de requisitos: Los requisitos recopilados son sometidos a un análisis detenido para detectar posibles discrepancias, ambigüedades y lagunas. El objetivo es lograr una comprensión profunda de lo que se requiere (Paredes Hernández & Velasco Espitia, 1998).

Modelado y diseño: En esta etapa, se crean representaciones visuales para describir el sistema desde diferentes perspectivas, como diagramas de flujo, casos de uso y diagramas de clases (Sommerville, 2006).

Definición de la arquitectura: Se establece la estructura global del sistema, incluyendo los principales componentes, interacciones y comunicaciones. Esto sienta las bases para el crecimiento y garantiza que el sistema sea escalable, manejable y eficiente (Sommerville, 2011).

Especificación de requisitos: Los requisitos se documentan minuciosamente, detallando las funcionalidades, reglas de negocio y restricciones técnicas. Esto proporciona una referencia precisa para los desarrolladores mientras implementan el sistema (Domínguez Coutiño, 2012).

Validación y verificación: Se verifica la coherencia, integridad y comprensibilidad de los requisitos recopilados. Esto puede implicar revisiones por parte de las partes interesadas y pruebas conceptuales (Tavera, 2018).

Gestión de cambios: Conforme avanza el proyecto, es común que surjan modificaciones en los requisitos. Es fundamental contar con un procedimiento para gestionar estos cambios y evaluar su impacto en el sistema (Kendall & Kendall, 2011).

Entrega de especificaciones: Una vez completadas y verificadas, las especificaciones y visualizaciones se pasan al equipo de desarrollo para su implementación (Paredes Hernández & Velasco Espitia, 1998).

1.2 Evolución de sistemas de software

La evolución de los sistemas de software se refiere al proceso continuo de crecimiento y cambio que experimentan las aplicaciones informáticas y las plataformas tecnológicas desde el concepto inicial hasta el estado actual y futuro. Este proceso resulta fundamental para mantener la pertinencia, funcionalidad y eficacia de los sistemas en un entorno en continua transformación. La evolución de los sistemas de software puede abarcar mejoras, ajustes, actualizaciones y adaptaciones que abordan diversos aspectos, como características funcionales, rendimiento, seguridad y usabilidad (Sommerville, 2006).

Algunos aspectos clave del desarrollo de sistemas de software incluyen:

Mantenimiento Correctivo: Implica identificar y rectificar errores, fallos y problemas que surgen durante el funcionamiento del sistema. Esto asegura que el software opere de manera confiable y conforme a lo previsto.

Mantenimiento Adaptativo: Incluye personalizaciones y modificaciones de software para adaptarse a los cambios en el entorno tecnológico, los requisitos del usuario y los requisitos comerciales. Esto puede incluir actualizaciones para cumplir con nuevas regulaciones, cambios en la plataforma principal o nuevas capacidades.

Mantenimiento Perfectivo: La atención se centra en mejorar y optimizar el software en términos de rendimiento, eficiencia y

usabilidad. Esto se hace para aumentar la satisfacción del usuario y la competitividad del sistema en el mercado.

Mantenimiento Preventivo: Implica tomar medidas preventivas para prevenir posibles problemas futuros identificando y abordando áreas de riesgo y debilidad potenciales en el software. Esto puede involucrar revisiones de código, actualizaciones de seguridad proactivas y ajustes preventivos.

Actualizaciones de Seguridad: Dada la creciente amenaza de ataques cibernéticos, las actualizaciones de seguridad son vitales para abordar vulnerabilidades y salvaguardar el sistema contra ataques maliciosos.

Refactorización: Implica reorganizar y reestructurar el código interno del sistema sin alterar su comportamiento externo. Esto mejora la calidad del código y su mantenibilidad a largo plazo.

Mejoras Incrementales: Con el tiempo, se pueden agregar nuevas características y funcionalidades al sistema para mantenerlo actualizado y competitivo en el mercado.

Reingeniería: En situaciones más profundas, puede ser necesario rediseñar significativamente la estructura y arquitectura del sistema, lo que implica una reingeniería completa para abordar problemas fundamentales o alcanzar un mejor rendimiento.

El desarrollo de sistemas de software es un proceso en constante evolución que requiere planificación, gestión y seguimiento continuos. Un enfoque efectivo en la evolución del software permite que los sistemas mantengan su utilidad, eficacia y competi-

tividad en un entorno tecnológico en constante evolución (Bournissen, 1999).

1.3 Estándares y principio de análisis

Los estándares y principios de análisis representan directrices y enfoques que dirigen el proceso de análisis de sistemas y software. Su objetivo es garantizar la calidad, la coherencia y la eficacia en la obtención de requisitos, el diseño y la creación de soluciones tecnológicas. Estas normas y principios establecen un marco sólido que brinda apoyo a los analistas al afrontar de manera eficiente los retos que surgen durante todo el ciclo de vida del desarrollo de software

Algunos de los estándares y principios comunes en el análisis son los siguientes: (Dominguez, 2012).

Principio de Responsabilidad Única (Single Responsibility Principle): Este principio estipula que cada módulo o componente debe tener una responsabilidad claramente definida. Esto hace que el código sea más fácil de entender, usar y mantener.

Principio de Apertura y Cierre (Open/Closed Principle): Este principio fomenta la creación de software que sea escalable pero difícil de modificar. En otras palabras, le permitirá agregar nuevas funciones sin tener que cambiar el código existente.

Principio de Sustitución de Liskov (Liskov Substitution Principle): Esta regla establece que los objetos de clase derivados pue-

den reemplazar a los objetos de clase base sin causar problemas de integridad del programa. Esto se refiere a la coherencia en el tipo y el comportamiento.

Principio de Separación de Interfaces (Interface Segregation Principle): Este principio sugiere que las interfaces deben ser específicas y no contener mucha funcionalidad. Esto previene que las clases se vean obligadas a implementar métodos que no son pertinentes para ellas.

Principio de Inversión de Dependencias (Dependency Inversion Principle): Este principio propone invertir las relaciones de dependencia entre módulos y clases. Esto se logra al depender de abstracciones en lugar de detalles concretos, lo cual incrementa la flexibilidad y la adaptabilidad.

Estándares de Documentación: Establecen las formas adecuadas para documentar requisitos, diseños y otros aspectos del software. Esto asegura la difusión y el acceso al conocimiento entre los miembros del equipo.

Estándares de Denominación: Definen cómo deben nombrarse las variables, funciones y componentes. Esto mejora la claridad y la comprensibilidad del código.

Estándares de Pruebas y Validación: Establecen métodos apropiados para realizar pruebas de software para garantizar su calidad y funcionalidad.

Estándares de Seguridad: Definen prácticas y enfoques para garantizar la seguridad del software y protegerlo de vulnerabilidades.

Estándares de Arquitectura: Ofrecen lineamientos para la estructura global de un sistema, incluyendo la distribución de componentes, las interacciones y las comunicaciones.

Estándares de Usabilidad: Establecen las directrices para diseñar interfaces de usuario con el propósito de asegurar una experiencia óptima para el usuario.

Principios Ágiles: Estos principios, como los presentes en el Manifiesto Ágil, orientan el desarrollo de software hacia enfoques colaborativos, adaptables y orientados a proporcionar valor de manera continua.

La adhesión a estos estándares y principios fomenta la creación de sistemas de software sólidos, mantenibles, escalables y alineados con las necesidades tanto de los usuarios como del negocio.

1.4 Análisis de sistemas estructurados.

El análisis de sistemas estructurados constituye un enfoque metodológico aplicado en el ámbito, establecen métodos apropiados para realizar pruebas de software para garantizar su calidad y funcionalidad. Este método se centra en dividir sistemas complejos en componentes más pequeños y manejables, haciendo que el sistema general sea más fácil de entender, diseñar e implementar de manera más efectiva. El análisis de sistemas estructurales se basa en el supuesto de que el sistema se puede dividir en segmentos más simples, lo que permite analizar y diseñar estas partes de

forma independiente antes de integrarlas en una solución completa (Dominguez, 2012).

Según Tavera (2018), dice que los elementos distintivos del análisis de sistemas estructurados son:

Descomposición Jerárquica: Implica dividir el sistema en módulos o subsistemas más pequeños. Cada módulo realiza una tarea específica e interactúa con otros módulos a través de interfaces predefinidas.

Diagramas de Flujo de Datos: Los diagramas de flujo de datos se utilizan para mostrar el flujo de datos en un sistema. Estos diagramas visualizan las entradas, procesos y salidas del sistema.

Diccionario de Datos: Implica la creación de un glosario que define todos los datos utilizados en el sistema, incluyendo su descripción, formato y relaciones con otros datos.

Diagramas de Estructura de Datos: Estos diagramas ilustran la organización y almacenamiento de datos en el sistema, incluyendo las relaciones entre distintos tipos de datos.

Diagramas de Proceso: Estos diagramas representan la ejecución de procesos y funciones dentro del sistema. La atención se centra en la lógica y las operaciones inherentes a cada proceso.

Tablas de Decisión: Se utilizan para exponer decisiones lógicas tomadas en el sistema, basadas en condiciones y criterios específicos.

Diseño Top-Down: Implica seguir un enfoque descendente en el diseño, partiendo de la visión general del sistema y descendiendo a niveles más detallados mientras se definen los componentes.

Especificaciones Detalladas: Cada componente se documenta minuciosamente, lo cual simplifica la comprensión, implementación y mantenimiento.

El enfoque de análisis de sistemas estructurados resulta especialmente provechoso en la concepción de sistemas complejos, al permitir abordar individualmente cada parte antes de su integración en una solución completa. A pesar de haber sido ampliamente utilizado en el pasado, en la actualidad este enfoque ha sido complementado y, en algunos casos, reemplazado por métodos más contemporáneos y ágiles, como el desarrollo orientado a objetos y las metodologías ágiles. Estos enfoques se adecuan mejor a la dinámica cambiante de las necesidades y requisitos del software (Dominguez, 2012).

1.5 Análisis de sistemas orientados a objetos.

El análisis de sistemas orientado a objetos es un método utilizado en la ingeniería de software y el desarrollo de sistemas de información. Se basa en los conceptos básicos de la programación orientada a objetos (POO). Este método favorece la representación de sistemas como colecciones de objetos interconectados, donde cada objeto representa una entidad, real o abstracta. Es-

tos objetos tienen propiedades y comportamientos específicos. El análisis de sistemas orientado a objetos se centra en comprender y representar las interacciones y relaciones entre estos objetos para crear soluciones flexibles, adaptables y reutilizables (Domínguez Coutiño, 2012).

Según Paredes Hernández & Velasco Espitia (1998), dicen que las características principales del análisis del sistema de objetos son:

Identificación de Objetos y Clases: Se reconocen objetos que denotan entidades del contexto problemático y se agrupan en clases. Estas clases definen las propiedades y acciones compartidas por los objetos.

Encapsulación: Los datos y funcionalidades relacionados se encapsulan en las clases, lo que conlleva a la ocultación de información y a la disminución de la complejidad.

Utilización de la Herencia: Relaciones de herencia se establecen entre clases, permitiendo así la creación de nuevas clases basadas en atributos de clases preexistentes. Esto fomenta la reutilización de código y la formación de jerarquías de clases.

Aplicación del Polimorfismo: Los objetos pueden adquirir distintas formas y reaccionar de modos variados en función del contexto. Esto posibilita que diferentes clases implementen comportamientos similares de manera específica.

Asociaciones y Relaciones: Las interacciones entre objetos se modelan por medio de asociaciones y relaciones. Estas intercone-

xiones pueden ser de uno a uno, de uno a muchos o de muchos a muchos.

Empleo de Diagramas de Clases: Los diagramas de clases se utilizan para visualizar clases, propiedades, métodos y relaciones entre ellos. Estos diagramas le ayudarán a comprender la estructura y el diseño del sistema.

Representación del Comportamiento: Modelar la interacción de objetos a lo largo del tiempo. Para ello se utilizan diagramas de secuencia y diagramas de estado.

Reutilización de Componentes: La modularidad inherente al enfoque orientado a objetos fomenta la reutilización de componentes y permite construir sistemas a partir de piezas más pequeñas.

Focalización en el Contexto del Problema: El análisis de sistemas orientado a objetos se centra en comprender y representar con precisión las entidades y relaciones inherentes al problema.

Adaptabilidad y Acomodo a los Cambios: El sistema de objetos es flexible a los requisitos cambiantes. La modificación de una parte del sistema a menudo tiene un impacto limitado en otras partes.

Uso de la Representación Visual: Las notaciones visuales como el Lenguaje de modelado unificado (UML) se utilizan para comunicar conceptos y relaciones del sistema.

El análisis de sistemas orientados a objetos posibilita un diseño más escalable, mantenible y modular, al poner el énfasis en

representar de manera precisa las relaciones y los comportamientos del sistema. Esta metodología es particularmente idónea para sistemas complejos y sujetos a cambios, ya que facilita la adaptación y la incorporación de nuevas funcionalidades.

1.6 Análisis de Sistemas de Inteligencia Artificial.

El análisis de sistemas de inteligencia artificial (IA) es un enfoque metodológico utilizado en el desarrollo de software y la creación de sistemas tecnológicos que utilizan las capacidades de la inteligencia artificial. Este proceso se centra en comprender, diseñar y desarrollar sistemas que utilizan algoritmos y técnicas de inteligencia artificial para realizar tareas que normalmente requieren habilidades cognitivas y de razonamiento humanas. Dichas tareas incluyen el procesamiento del lenguaje natural, el reconocimiento de patrones, la toma de decisiones y la resolución de problemas complejos (Cáceres, 2014).

Características básicas del análisis de sistemas de inteligencia artificial:

Definición de Metas: Se determinan las metas y tareas que el sistema de IA deberá cumplir, delimitando los logros que se persiguen mediante la introducción de la inteligencia artificial.

Elección de Métodos y Técnicas: Se opta por algoritmos y técnicas de IA apropiados para abordar las tareas específicas del sistema.

Recopilación y Preprocesamiento de Datos: Los sistemas de IA dependen de datos para ser entrenados y perfeccionar su desempeño. Por lo tanto, se recolectan y depuran los datos pertinentes para nutrir los modelos de IA.

Creación de Modelos: Se diseña los modelos de IA que serán entrenados para realizar tareas concretas.

Proceso de Entrenamiento y Aprendizaje: Los modelos de IA son entrenados utilizando los datos recolectados. Durante esta fase, los parámetros del modelo se ajustan con el objetivo de mejorar el rendimiento en base a ejemplos suministrados.

Evaluación y Validación: Se examina el rendimiento del sistema de IA utilizando medidas específicas ajustadas a la tarea en cuestión. Esto permite evaluar la precisión y efectividad del sistema.

Ajuste y Optimización: En caso necesario, se efectúan ajustes en los modelos de IA para potenciar su desempeño. Este proceso puede implicar la afinación de hiperparámetros y la incorporación de técnicas de regularización.

Integración en el Sistema: El sistema de IA se integra en la aplicación o plataforma más amplia en la que se empleará, garantizando su coherencia y eficacia.

Pruebas y Certificación: Se realizan pruebas exhaustivas para comprobar que el sistema de IA funcione adecuadamente en diversas circunstancias y escenarios.

Supervisión y Mejora Continua: Una vez implantado, el sistema de IA se supervisa con el fin de identificar posibles problemas, realizando ajustes y mejoras conforme sea necesario.

Consideraciones Éticas y Legales: Se tienen en consideración las ramificaciones éticas y legales asociadas al uso de la inteligencia artificial, como la protección de la privacidad de los datos y la equidad en las decisiones automatizadas.

El análisis de sistemas de inteligencia artificial busca garantizar que los sistemas construidos sean precisos, confiables y efectivos en la ejecución de tareas específicas. Asimismo, implica una reflexión sobre las implicaciones éticas y legales vinculadas con la implementación de tecnologías de IA.

AUTOEVALUACIÓN 1

1. ¿Cuál es la definición de un sistema de software?

1. Un conjunto de programas de computadora interconectados.
2. Hardware y software combinados.
3. Un único programa de computadora.
4. Un sistema de hardware.

2. ¿Cuál de las siguientes afirmaciones describe mejor la evolución de los sistemas de software?

1. Los sistemas de software han permanecido estáticos a lo largo del tiempo.
2. Los sistemas de software han crecido en complejidad y funcionalidad.
3. Los sistemas de software han disminuido en complejidad con el tiempo.
4. Los sistemas de software solo existen desde hace unos pocos años.

3. ¿Cuál de los siguientes no es un estándar comúnmente utilizado en el análisis de sistemas de software?

1. ISO 9001
2. CMMI
3. IEEE 802.11
4. ITIL

4. ¿Cuál es uno de los principios clave del análisis de sistemas?

1. Maximizar la complejidad del sistema.
2. Minimizar la comunicación entre los miembros del equipo.
3. Identificar y resolver problemas de manera sistemática.
4. Ignorar las necesidades del usuario.

5. ¿Qué es el análisis de sistemas estructurados?

1. Un enfoque que se centra en el diseño de la interfaz de usuario.
2. Un enfoque que divide un sistema en módulos independientes.
3. Un enfoque que se enfoca en la recopilación de requisitos del usuario.
4. Un enfoque que ignora por completo la estructura del sistema.

6. ¿Qué es el análisis de sistemas orientados a objetos?

1. Un enfoque que se centra en la estructura de datos y funciones.
2. Un enfoque que se basa en la programación procedimental.
3. Un enfoque que modela el sistema en términos de objetos y sus interacciones.
4. Un enfoque que no utiliza la programación orientada a objetos.

7. ¿Cuál de las siguientes tecnologías está más estrechamente relacionada con el análisis de sistemas de inteligencia artificial?

1. Redes neuronales artificiales
2. Bases de datos relacionales
3. Lenguajes de programación tradicionales
4. Sistemas operativos

8. ¿Qué es la ingeniería de requisitos en el contexto del análisis de sistemas de software?

1. El proceso de diseñar interfaces de usuario atractivas.
2. La disciplina que se encarga de identificar, documentar y gestionar los requisitos del sistema.
3. El proceso de codificar el software.
4. La fase de pruebas del desarrollo de software.

9. ¿Qué es el ciclo de vida del software?

1. Una etapa única de desarrollo de software.
2. El proceso completo de desarrollo de software, desde la concepción hasta la retirada.
3. La fase de mantenimiento del software.
4. El proceso de copiar el software en discos.

10. ¿Cuál es el objetivo principal del análisis de sistemas?

1. Desarrollar el software directamente.
2. Comprender y definir los requisitos del sistema.
3. Probar el software en busca de errores.
4. Implementar el sistema en hardware.

11. ¿Qué es un diagrama de flujo de datos en el análisis de sistemas?

1. Un diagrama que muestra la estructura de datos de un sistema.
2. Un diagrama que representa visualmente el flujo de datos entre componentes de un sistema.
3. Un diagrama que muestra la jerarquía de módulos en un sistema.
4. Un diagrama que solo se utiliza en la programación orientada a objetos.

12. ¿Cuál es uno de los beneficios clave de utilizar un enfoque orientado a objetos en el análisis de sistemas?

1. Mayor complejidad en el diseño del sistema.
2. Reutilización de código y componentes.
3. Mayor dependencia de la programación procedural.
4. Menos flexibilidad en el desarrollo.

13. ¿Qué es el principio SOLID en programación orientada a objetos?

a) Un conjunto de principios para diseñar bases de datos relacionales.
b) Un conjunto de principios para escribir código eficiente.
c) Un conjunto de principios para escribir código legible y mantenible.
d) Un conjunto de principios para la gestión de proyectos de software.

14. ¿Cuál es la diferencia entre el aprendizaje supervisado y el aprendizaje no supervisado en el contexto de la inteligencia artificial?

1. En el aprendizaje supervisado, se requiere un conjunto de datos etiquetado, mientras que en el aprendizaje no supervisado no se requiere etiquetado.
2. En el aprendizaje no supervisado, un experto proporciona supervisión constante, mientras que en el aprendizaje supervisado no es necesario.
3. No hay diferencia entre ellos; son términos intercambiables.
4. El aprendizaje supervisado se utiliza solo en la programación orientada a objetos.

15. ¿Qué es la ingeniería de software?

a) La disciplina que se enfoca en la construcción de hardware.
b) La disciplina que se encarga de desarrollar software de manera sistemática.

c) La disciplina que se enfoca en la seguridad de redes.
d) La disciplina que se encarga de la administración de servidores.

16. ¿Cuál es el objetivo principal de la ingeniería de requisitos?

1. Escribir código eficiente.
2. Identificar y documentar los requisitos del sistema.
3. Realizar pruebas exhaustivas del software.
4. Diseñar la interfaz de usuario.

17. ¿Cuál es el propósito del modelado de sistemas en el análisis de sistemas de software?

1. Representar visualmente los requisitos del usuario.
2. Identificar y gestionar riesgos en el proyecto de desarrollo.
3. Definir la estructura de la base de datos.
4. Crear documentación técnica detallada.

18. ¿Qué es un diagrama de casos de uso en el contexto del análisis de sistemas?

a) Un diagrama que muestra las interacciones entre objetos en un sistema.
b) Un diagrama que representa las funciones que un sistema realiza para los actores externos.
c) Un diagrama que muestra la estructura de datos de un sistema.
d) Un diagrama que solo se utiliza en el análisis estructurado.

19. ¿Cuál es el propósito de la fase de prueba en el ciclo de vida del software?

1. Identificar y documentar los requisitos del sistema.
2. Desarrollar el software.
3. Encontrar y corregir errores en el software.
4. Mantener el software.

20. ¿Qué es la programación genética en el campo de la inteligencia artificial?

1. Un enfoque para resolver problemas de matemáticas avanzadas.
2. Un enfoque para evolucionar automáticamente programas informáticos.
3. Un enfoque para el análisis de datos en grandes conjuntos de datos.
4. Un enfoque para la programación de videojuegos.

1.7 Técnicas de Análisis

1.7.1 Técnicas estructurales

Las técnicas estructurales en el ámbito del análisis y diseño de sistemas tienen como enfoque fundamental la descomposición y organización de un sistema en sus elementos constituyentes y subsistemas. Este enfoque tiene la finalidad de facilitar la comprensión y la concepción de la estructura del sistema de manera más eficiente. Dichas técnicas juegan un papel crucial en la gestión de la complejidad inherente y aseguran la coherencia, modularidad y claridad del sistema. A continuación, se presentan algunas técnicas estructurales esenciales utilizadas en el análisis y diseño de sistemas (Cedeño, 2015).

Diagramas de Flujo de Datos (DFD):

Los DFD ofrecen una representación visual que ilustra cómo fluye la información en un sistema. Su utilidad radica en modelar los procesos, las entradas, las salidas y los depósitos de datos (Senn, 1997).

Diagramas de Estructura de Datos (DED):

DED se utiliza para representar la estructura lógica de la base de datos. Estos diagramas son necesarios para determinar cómo se relacionan los diferentes datos entre sí en el sistema y cómo se organizan en tablas y campos. Esta fase es crítica para garantizar la integridad y eficiencia en la administración de datos (Whitten, Bentley, & Barlow, 2003).

Diagramas de Entidad-Relación (ERD):

Los ERD son empleados para modelar las conexiones entre diferentes entidades (objetos o conceptos) presentes en una base de datos. Estos diagramas resultan esenciales para comprender cómo las entidades están interconectadas y cómo interactúan en el contexto del sistema (Pressman, 1993).

Diagramas de Contexto:

Los diagramas de contexto proporcionan una visión panorámica del sistema, exhibiendo cómo interactúa con elementos externos. Cumplen una función crucial al definir los límites del sistema y al brindar una comprensión de su interacción con el entorno (Larman, 1999).

Diagramas de Flujo de Proceso (DFP):

Los DFP representan visualmente las etapas o actividades que tienen lugar en un proceso o secuencia de procesos. Son útiles para modelar la dinámica del flujo de trabajo, las decisiones y las acciones dentro del sistema (Rumbaugh, 1996).

Diagramas de Estructura de Programa (DEP):

Los DEP son aplicados en el diseño de software para representar la organización y estructura lógica de un programa. Estos diagramas facilitan la descomposición del programa en módulos y presentan cómo estos módulos se interrelacionan (Tavera, 2018).

Diagramas de Interacción (como Diagramas de Secuencia y Diagramas de Colaboración):

Estos diagramas se centran en ilustrar cómo los distintos elementos del sistema interactúan entre sí, a través de intercambios de mensajes y colaboraciones. Estos diagramas resultan particularmente valiosos en la modelización de procesos en sistemas orientados a objetos.

Por lo tanto, estos métodos estructurales juegan un papel importante en el proceso de análisis y diseño del sistema al proporcionar una representación visual consistente y bien organizada de la estructura y el flujo de información dentro del sistema. Su uso combinado se puede adaptar a las características y necesidades específicas de cada proyecto.

1.7.2 Técnicas Orientadas a Objetos

Los métodos basados en el enfoque orientado a objetos juegan un papel clave en el análisis y diseño de sistemas porque permiten modelar y representar sistemas con mayor precisión de lo que realmente son, utilizando el concepto de objetos y sus interacciones.

Según Cáceres (2014), a continuación, se enumeran algunas técnicas orientadas a objetos ampliamente empleadas en este contexto:

El Proceso de Modelado de Clases y Objetos se centra en la identificación de las clases (objetos) pertinentes dentro del sistema, junto con sus atributos y métodos inherentes. A través de este proceso, se configura un esquema de clases que encapsula tanto la estructura como las responsabilidades inherentes a los objetos.

Los Diagramas de Clases, en su versión gráfica, representan las clases de forma visual, detallando sus atributos, métodos y relaciones. Estos diagramas esclarecen cómo las clases se vinculan en términos de herencia, asociación, composición, entre otros.

Los Diagramas de Secuencia tienen como objetivo presentar las interacciones entre objetos a lo largo del tiempo, pormenorizando el orden en que los mensajes son transmitidos y recibidos por los objetos durante operaciones o escenarios específicos.

En el ámbito de los Diagramas de Colaboración (también conocidos como Diagramas de Comunicación), se traducen las interacciones entre objetos en forma de mensajes intercambiados. Estos diagramas proveen una visión global de la sinergia entre los objetos para lograr una funcionalidad en conjunto.

El Modelado de Estados se concentra en cómo un objeto experimenta cambios de estado como respuesta a eventos, especialmente útil para retratar objetos con estados cambiantes.

El proceso de Modelado de Casos de Uso entra en detalle respecto a la interacción entre usuarios y el sistema, identificando tanto actores como los casos de uso correspondientes. Estos últimos esquematizan cómo los usuarios interactúan con el sistema para lograr objetivos concretos.

La Herencia y el Polimorfismo añaden flexibilidad al permitir que las clases hereden atributos y métodos de otras clases, además de admitir que diferentes clases implementen métodos con el mismo nombre de manera específica.

El Encapsulamiento, por otro lado, desempeña un rol crucial al ocultar la estructura interna de un objeto y exponer únicamente interfaces públicas.

Mediante el proceso de Abstracción, se simplifica la representación de objetos al enfocarse únicamente en sus aspectos esenciales, omitiendo detalles irrelevantes y contribuyendo a una comprensión más precisa.

En el ámbito de los Diagramas de Actividad, se plasma la secuencia de actividades y acciones dentro de un proceso o función particular, proveyendo un recurso útil para visualizar procesos y flujos de trabajo.

La conjunción de estas técnicas permite lograr una representación más realista y coherente de los sistemas, lo cual agiliza tanto el proceso de diseño como de desarrollo de sistemas de software efectivos y sostenibles. Cuando se utilizan de manera integrada, estas técnicas asisten en la comprensión de cómo los objetos interactúan, colaboran y contribuyen a alcanzar los objetivos del sistema.

1.7.3 Técnicas para Aplicaciones Web y Móviles

Las metodologías empleadas en el análisis y diseño de sistemas para aplicaciones web y móviles están adaptadas de manera específica a las características particulares de estas plataformas y sus necesidades únicas. En este contexto, se presentan una serie de técnicas que son comunes

Según Monseñor (2012), nos indica que**:**

Prototipado Interactivo: Esta es una práctica común al crear aplicaciones web y móviles, prototipos interactivos que permitan a los usuarios tener una experiencia anticipada de cómo funcionará la aplicación antes de que esté completamente desarrollada. Esto resulta útil para identificar problemas tempranamente y validar conceptos.

Diseño Responsivo: Dado que las aplicaciones web y móviles deben funcionar en diversos dispositivos y tamaños de pantalla, el diseño responsivo es esencial. Se busca que la interfaz se adapte de manera fluida y eficiente a distintas resoluciones de pantalla.

Creación de Wireframes y Mockups: Utilizar wireframes y mockups es una técnica valiosa para visualizar la estructura y el diseño de la aplicación antes de su implementación. Estos esquemas visuales facilitan definir la distribución de los elementos y la navegación.

Historias de Usuario: Las historias de usuario son historias cortas que describen una funcionalidad específica desde la perspectiva del usuario. Estas historias lo ayudan a comprender las necesidades de los usuarios y establecer los requisitos necesarios.

Diseño Centrado en el Usuario: Al analizar y diseñar sistemas para aplicaciones web y móviles, es extremadamente importante mantener las necesidades y preferencias del usuario en el centro. Esto puede incluir investigaciones de usuarios, encuestas y pruebas de usabilidad para garantizar que la interfaz sea intuitiva y utilizable.

Integración de Plataformas y APIs: En muchas aplicaciones web y móviles, es esencial incorporar servicios de terceros mediante APIs (Interfaces de Programación de Aplicaciones) para funciones como autenticación, pagos, mapas, entre otros.

Diseño de la Interfaz de Usuario (UI): La UI debe ser visualmente atractiva y coherente con la identidad de la marca, además de ser fácil de utilizar. Elementos como colores, tipografías, iconos y la disposición de los elementos en pantalla son considerados.

Diseño de la Experiencia de Usuario (UX): La UX se enfoca en ofrecer una experiencia positiva y fluida al usuario. Factores como la fluidez en la navegación, la rapidez de carga y la capacidad de respuesta se toman en cuenta.

Pruebas en Dispositivos Reales: Dado que las aplicaciones web y móviles se ejecutan en una variedad de dispositivos y sistemas operativos, es crucial realizar pruebas en dispositivos reales para asegurarse de que la aplicación funcione correctamente en todas las circunstancias.

Desarrollo Ágil: El enfoque ágil en el desarrollo de software permite realizar iteraciones frecuentes y adaptarse continuamente en base a los comentarios de los usuarios. Este enfoque es particularmente útil en el contexto de aplicaciones web y móviles, donde las actualizaciones rápidas pueden ser necesarias para estar al tanto de las demandas del mercado y las expectativas de los usuarios.

Estas técnicas específicas para aplicaciones web y móviles toman en consideración las características propias de estas pla-

taformas, incluyendo la diversidad de dispositivos, la interacción táctil, la conectividad en línea y las cambiantes expectativas de los usuarios. Su implementación adecuada puede desempeñar un papel importante en el éxito de las aplicaciones en estos entornos.

Autoevaluación 2

1. **¿Qué es el análisis de sistemas?**
   a) Un proceso de diseño de interfaces de usuario.
   b) La identificación y definición de requisitos del sistema.
   c) Un enfoque para la programación de aplicaciones web.
   d) La fase de prueba de un proyecto de software.
2. **¿Cuál de las siguientes técnicas se utiliza comúnmente para la recopilación de requisitos?**
   1. Diagramas de flujo de datos.
   2. Programación orientada a objetos.
   3. Pruebas de rendimiento.
   4. Optimización de bases de datos.
3. **¿Qué es un diagrama de casos de uso en el análisis de sistemas?**
   a. Un diagrama que muestra la estructura de una base de datos.
   b. Un diagrama que representa las interacciones entre actores y el sistema.
   c. Un diagrama que muestra la arquitectura de hardware de un sistema.
   d. Un diagrama que solo se utiliza en la programación estructurada.

4. **¿Qué objetivo principal tiene el análisis de sistemas?**
    1. Desarrollar software sin tener en cuenta los requisitos del usuario.
    2. Comprender y definir los requisitos del sistema.
    3. Realizar pruebas exhaustivas del software.
    4. Implementar el sistema en hardware.
5. **¿Qué es la descomposición modular en el análisis de sistemas estructurados?**
    1. Un enfoque que combina todos los módulos en uno solo.
    2. La división de un sistema en módulos independientes y bien definidos.
    3. Un proceso para crear diagramas de casos de uso.
    4. Una técnica para la optimización de bases de datos.
6. **¿Cuál es el objetivo principal de la programación estructurada?**
    1. Crear interfaces de usuario atractivas.
    2. Dividir un programa en estructuras de control lógicas.
    3. Desarrollar software sin una estructura definida.
    4. Ignorar por completo la estructura del software.
7. **¿Qué es una clase en programación orientada a objetos?**
    1. Un programa independiente.
    2. Una instancia de un objeto.
    3. Un plano de diseño para un objeto.
    4. Una función que realiza una tarea específica.
8. **¿Qué es la herencia en programación orientada a objetos?**
    a. La capacidad de una clase para tener múltiples instancias.
    a. La capacidad de una clase para heredar atributos y métodos de otra clase.
    b. La eliminación de objetos no utilizados.
    c. La capacidad de una clase para ser independiente de otras clases.

9. ¿Qué es la usabilidad en el contexto de las aplicaciones web y móviles?

1. La capacidad de una aplicación para funcionar sin conexión a Internet.
2. La facilidad con la que los usuarios pueden utilizar una aplicación de manera efectiva y satisfactoria.
3. La seguridad de una aplicación.
4. La capacidad de una aplicación para funcionar en múltiples dispositivos.

10. ¿Qué es el diseño responsivo en el desarrollo de aplicaciones web y móviles?

a. Un enfoque que ignora por completo la apariencia en diferentes dispositivos.

b. Un enfoque que adapta la apariencia y funcionalidad de una aplicación según el dispositivo del usuario.

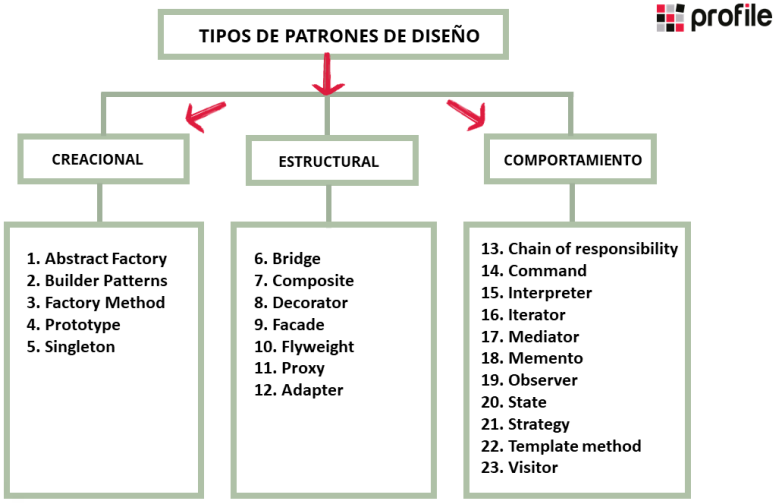
c. Un enfoque que solo funciona en dispositivos móviles.

d. Un enfoque que solo funciona en navegadores web de escritorio.

1.8 Patrones de Diseño de Software (GoF of Four)

El proceso de identificación de modelos de diseño GOF en procesos de creación de software requiere un conjunto de actividades; Inicialmente, se construyó un conjunto de criterios, lo que le permite determinar el punto de análisis para determinar los procesos de desarrollo más representativos, lo que necesariamente implementa estos criterios para la categoría. Este producto y la selección de procesos de desarrollo. El llenado de condiciones para los procesos de desarrollo es estricto e importante porque el propósito de estos estudios es obtener los resultados oficiales de la investigación en los procesos de desarrollo y, por lo tanto, un proceso de procesos seleccionados del desarrollo, correspondiente al control de calidad estricto en la etapa de Requisitos, diseño, desarrollo, desarrollo y desarrollo y desarrollo (Canelo, 2020).

Figura 1.Tipos de patrones de diseño de software.



Nota. Como podemos observar en la imagen de arriba, los diseños más populares se dividen en tres categorías básicas y juntos forman 23 diseños individuales. Estos tres tipos principales son: modelos creativos, modelos estructurales, modelos de comportamiento.

1.8.1 Patrones de creación

Los patrones de diseño proporcionan diferentes enfoques para la creación de objetos, aumentando la flexibilidad y reutilizando el código existente de una manera específica del contexto. Esto le da al programa más flexibilidad para determinar qué objetos crear según el caso de uso específico.

Según Ccori Huaman (2018), los patrones de creación incluidos son:

Abstract Factory

Según este patrón, la interfaz se encarga de crear colecciones o grupos de objetos relacionados sin proporcionar el nombre de una clase específica.

Builder Patterns

Facilita la generación de variados tipos y formas de un objeto empleando un único código de construcción. Su utilidad radica en la construcción gradual de un objeto complicado mediante la combinación de elementos más simples. La formación última de los objetos se basa en las etapas del proceso de construcción, pero es independiente de los demás objetos.

Factory Method

Ofrece una interfaz en la superclase para generar objetos, pero permite a las subclases modificar el tipo de objetos que serán generados. Brinda una instanciación de objetos implícita mediante interfaces compartidas.

Prototype

Le permite clonar objetos existentes sin crear dependencias en sus clases. Se utiliza para limitar operaciones en memoria o base de datos, minimizando los cambios mediante el uso de copias de objetos.

Singleton

Este diseño de patrón limita la creación de una instancia de clase a un solo objeto.

1.8.2 Patrones Estructurales

Facilitan la toma de decisiones y un enfoque eficiente para la composición de clases y la configuración de objetos. El principio de herencia se utiliza para conectar interfaces y establecer métodos que combinan objetos para obtener nuevas funcionalidades.

Según Canelo (2020), entre los patrones estructurales tenemos:

Adapter

Este patrón se emplea para enlazar dos interfaces que no son compatibles, permitiendo que utilicen sus funcionalidades. El adaptador habilito a las clases para colaborar de manera que, de otro modo, no sería posible debido a las incompatibilidades entre sus interfaces.

Bridge

En este patrón, las modificaciones estructurales ocurren en la clase principal y en la clase que implementa la interfaz sin afectarse entre sí. Ambas capas se pueden desarrollar de forma independiente y conectarse a través de una interfaz intermedia.

Composite

Se utiliza para agrupar objetos en un solo objeto. Esto facilita la creación de estructuras de árbol con objetos y le permite trabajar con estas estructuras como si fueran objetos separados.

Decorator

Este patrón limita los cambios en la estructura de un objeto cuando se agregan nuevas características. La capa original permanece sin cambios, pero la capa decorativa añade funcionalidad adicional.

Facade

Proporciona una interfaz simplificada para acceder a una biblioteca, marco o conjunto complejo de clases. Esto facilita su uso al ocultar la complejidad y proporcionar una accesibilidad más sencilla.

Flyweight

El patrón Flyweight se utiliza para reducir el consumo de memoria y mejorar el rendimiento minimizando la creación de objetos. Encuentre objetos existentes similares para reutilizarlos en lugar de crear nuevos objetos similares.

Proxy

Se utiliza para crear objetos que representan funciones de otras clases u objetos, y se utiliza una interfaz para acceder a estas funciones.

1.8.3 Patrones de Comportamiento

Un patrón de comportamiento se centra en las interacciones

entre objetos de una clase y se utiliza para descubrir y manipular patrones de comunicación existentes. Estos patrones están directamente relacionados con las interacciones entre objetos.

Según (Ccori Huaman, 2018) los patrones de comportamiento son los siguientes:

Chain of Responsibility

El patrón Cadena de Responsabilidad es un diseño de comportamiento que impide la comunicación entre el remitente de la solicitud y el destinatario de la solicitud, permitiendo que múltiples entidades respondan a la solicitud.

Command

Convierta la solicitud en un objeto autónomo que contenga toda la información necesaria. Esto le permite parametrizar métodos con diferentes consultas, controlar la ejecución de consultas retrasadas o en cola y admitir operaciones reversibles.

Interpreter

Se utiliza para evaluar el lenguaje o la expresión mediante la creación de una interfaz que establece el contexto para la interpretación.

Iterator

Facilita el acceso secuencial a elementos dentro de un objeto de colección sin exponer detalles de su implementación.

Mediator

Proporciona comunicación simple a través de una capa proxy que permite la comunicación entre múltiples capas.

Observer

Defina un mecanismo de suscripción para notificar a varios objetos sobre eventos que ocurren en un observable.

State

En el patrón state, el comportamiento de una clase depende de su estado actual y está representado por un objeto de contexto.

Strategy

Le permite definir un grupo de algoritmos, cada uno en una clase separada, y hace que los objetos sean intercambiables.

Template Method

Se utiliza con componentes similares donde se puede implementar un ejemplo de código para probar ambos componentes. El código puede variar con cambios menores.

Visitor

El patrón Visitor tiene como objetivo definir nuevas operaciones sin modificar la estructura de objetos existente.

Autoevaluación 3

1. **¿Cuál de los siguientes es un patrón de creación que se utiliza para crear objetos de una clase determinada?**
   a) Patrón Singleton
   b) Patrón Decorator
   c) Patrón Observer
   d) Patrón Strategy
2. **El patrón de creación que se utiliza para construir objetos complejos paso a paso es:**
   1. Patrón Abstract Factory
   2. Patrón Prototype
   3. Patrón Builder
   4. Patrón Factory Method
3. **¿Cuál de los siguientes patrones de creación se utiliza para crear objetos a partir de una interfaz abstracta?**
   1. Patrón Singleton
   2. Patrón Abstract Factory
   3. Patrón Prototype
   4. Patrón Adapter
4. **¿Qué patrón de creación permite que una clase tenga una sola instancia y proporcione un punto de acceso global a esa instancia?**
   a. Patrón Prototype
   b. Patrón Singleton
   c. Patrón Factory Method
   d. Patrón Builder

5. **¿Cuál de los siguientes patrones estructurales se utiliza para agregar funcionalidad adicional a objetos existentes de manera dinámica?**
    1. Patrón Decorator
    2. Patrón Adapter
    3. Patrón Facade
    4. Patrón Proxy
6. **El patrón de diseño que se utiliza para crear una interfaz unificada para un conjunto de interfaces en un subsistema es:**
    1. Patrón Bridge
    2. Patrón Composite
    3. Patrón Facade
    4. Patrón Adapter
7. **¿Cuál de los siguientes patrones estructurales se utiliza para separar la interfaz de un objeto de su implementación?**
    a) Patrón Bridge
    b) Patrón Decorator
    c) Patrón Proxy
    d) Patrón Composite
8. **El patrón de comportamiento que se utiliza para definir una serie de pasos para realizar una operación y permitir que las subclases proporcionen la implementación concreta de esos pasos es:**
    a) Patrón Strategy
    b) Patrón Observer
    c) Patrón Template Method
    d) Patrón State

9. ¿Qué patrón de comportamiento permite que un objeto altere su comportamiento cuando su estado interno cambia?

1. Patrón Command
2. Patrón Observer
3. Patrón State
4. Patrón Memento

10. El patrón de comportamiento que se utiliza para definir una dependencia uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente es:

a) Patrón Mediator
b) Patrón Memento
c) Patrón Observer
d) Patrón Command

11. ¿Qué patrón de comportamiento se utiliza para encapsular una solicitud como un objeto, lo que permite parametrizar a los clientes con operaciones, poner en cola solicitudes o registrar sus ejecuciones?

1. Patrón Command
2. Patrón Observer
3. Patrón State
4. Patrón Visitor

12. El patrón de comportamiento que se utiliza para definir una familia de algoritmos, encapsular cada uno de ellos y hacer que sean intercambiables es:

a) Patrón Observer
b) Patrón Strategy
c) Patrón Visitor
d) Patrón Memento

13. ¿Cuál de los siguientes patrones de comportamiento permite que un objeto capture su estado interno y lo restaure más tarde?

1. Patrón Memento
2. Patrón Observer
3. Patrón Command
4. Patrón State

14. ¿Qué patrón de comportamiento se utiliza para definir una jerarquía de clases de algoritmos, encapsular cada uno de ellos y permitir que el algoritmo sea seleccionado en tiempo de ejecución?

a) Patrón Observer
b) Patrón Strategy
c) Patrón Command
d) Patrón Template Method

15. El patrón de comportamiento que se utiliza para representar un conjunto de operaciones a realizar sobre elementos de una estructura de objetos es:

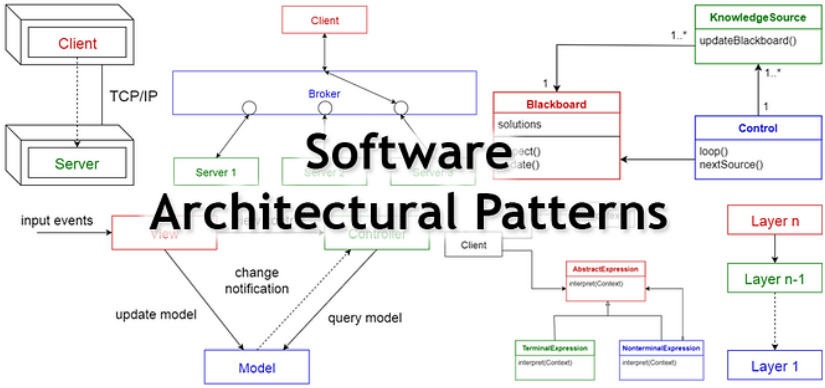
1. Patrón Mediator
2. Patrón Visitor
3. Patrón Command
4. Patrón Observer

1.9 Patrones de arquitectura

Un patrón arquitectónico es una solución genérica y reutilizable para resolver problemas arquitectónicos de software comunes en un contexto específico. Estos patrones arquitectónicos son similares a los patrones de diseño de software pero tienen un alcance más amplio.

En esta guía metodológica, se describirán detalladamente los patrones arquitectónicos comunes, junto con su aplicación, ventajas y desventajas.

Figura 2. Arquitectura de Software



Nota. En la figura anterior podremos observar, el proceso de diseño de sistemas empresariales de gran envergadura, antes de emprender un desarrollo de software crucial, es imperativo seleccionar una arquitectura apropiada que garantice la funcionalidad

requerida y los aspectos de calidad deseados. Por lo tanto, es fundamental adquirir un entendimiento de diversas arquitecturas antes de aplicarlas en nuestro proceso de diseño.

1.9.1 Filtros y tuberías

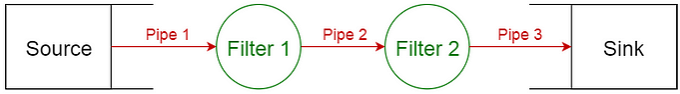
Este patrón encuentra aplicabilidad en la organización de sistemas que generan y procesan un cuadro de datos. Cada distancia de procesamiento se encapsula en un integrante de brebaje. Los datos destinados a emplumar son dirigidos a través de tuberías. Estas tuberías pueden emplearse tanto para el almacenaje tangible como para propósitos relacionados con audio

Usos

En la construcción de compiladores, los filtros sucesivos realizan tareas como análisis léxico, análisis sintáctico y generación de código.

En el ámbito de la bioinformática, se emplea en flujos de trabajo para gestionar datos biológicos y genómicos.

Figura 3. Filtros y Tuberías



Nota. Filtros y tuberías como muestra en la figura 3, ofrece un marco para la organización de sistemas de flujo de datos procesales.

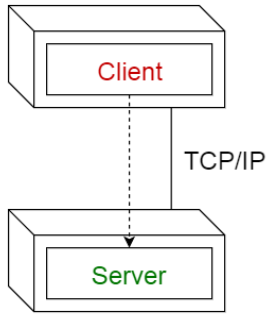
1.9.2 Cliente Servidor

La arquitectura consta de un servidor y varios clientes. ¿Cuáles son los componentes principales? Los servicios son proporcionados por el componente del servidor a diferentes componentes del cliente. El servidor proporciona los servicios necesarios a los clientes que realizan solicitudes. En consecuencia, el servidor espera la entrada de los clientes.

Usos

Este enfoque se encuentra en uso en aplicaciones en línea tales como el correo electrónico, el intercambio de documentos y las plataformas bancarias.

Figura 4. Patrón de capas



Nota. Como se muestra en la figura 4 la estructura del sistema se divide en servicios y sus respectivos servidores, junto con clientes que acceden y utilizan esos servicios.

1.9.3 M.V.C (Modelo, Vista, Controlador)

Una aplicación interactiva se divide en tres componentes según el patrón MVC, también conocido como:

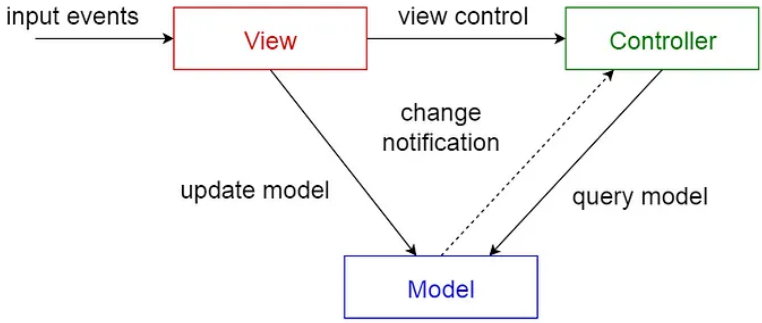
- Modelo: contiene la funcionalidad y los datos fundamentales.
- Vista: presenta la información al usuario (y es posible definir múltiples vistas).
- Controlador: gestiona la entrada del usuario.

Esta división busca separar la representación interna de la información de las formas en que es presentada y recibida por el usuario. El proceso de disociar elementos permite la reutilización de código de manera eficiente.

Uso

Este enfoque arquitectónico es empleado en aplicaciones de la World Wide Web en diferentes lenguajes de programación fundamentales. Además, se utiliza en frameworks web como Django y Rails.

Figura 5. Modelo Vista Controlador.



Nota. El modelo contiene la funcionalidad básica y datos. La vista muestra la información al usuario. El controlador maneja la entrada del usuario. La vista y el controlador en conjunto constituyen la interfaz de usuario.

1.9.4 N Capas

Esta norma puede ser usado para alertar programas que puedan ser desglosados en grupos de subáreas, cada una de ellas delimitada en un grado específico de meditación. Cada madre proporciona servicios al álveo solemne inmediata (Ccori Huaman, 2018).

Las cuatro capas más comunes en un sistema de información convencional son las siguientes:

Capa de presentación (también denominada interfaz de usuario–UI).

Capa de aplicación (también conocida como capa de servicio).

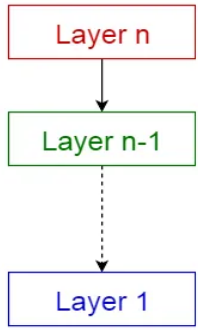
Capa de lógica de negocios (igualmente llamada capa de dominio).

Capa de acceso a datos (también identificada como capa de persistencia).

Uso

Este enfoque se emplea tanto en aplicaciones de escritorio de uso variado como en aplicaciones web de comercio electrónico.

Figura 6. N Capas.



Nota. La Arquitectura de Capas es una de las estrategias más frecuentes que los arquitectos de software emplean para descomponer sistemas de software.

Autoevaluación 4

1. **¿Qué patrón de arquitectura se utiliza para separar la lógica de presentación de la lógica de negocio en una aplicación?**
    1. Cliente-Servidor
    2. Patrón de Tuberías y Filtros
    3. MVC (Modelo, Vista, Controlador)
    4. Patrón de Arquitectura de Microservicios
2. **En una arquitectura de microservicios, ¿cómo se dividen las funcionalidades de una aplicación?**
    1. En módulos que se comunican directamente entre sí.
    2. En servicios independientes que se comunican a través de una red.
    3. En una sola unidad monolítica.
    4. En una única vista.
3. **¿Cuál es el objetivo principal del patrón M.V.C (Modelo, Vista, Controlador)?**
    a. Separar la interfaz de usuario de la lógica de presentación y el control de la aplicación.
    b. Facilitar la comunicación entre clientes y servidores.
    c. Dividir la aplicación en componentes que se ejecutan en diferentes máquinas.
    d. Definir un conjunto de reglas para la transmisión de datos en una red.

4. **¿Qué patrón de arquitectura se utiliza para dividir una aplicación en módulos independientes que se comunican mediante interfaces bien definidas?**
   a) Cliente-Servidor
   b) MVC (Modelo, Vista, Controlador)
   c) Patrón de Tuberías y Filtros
   d) Patrón de Arquitectura de Microservicios
5. **¿Qué patrón se utiliza para procesar datos de manera secuencial, donde la salida de un componente se convierte en la entrada del siguiente componente?**
   a. Patrón de Arquitectura de Microservicios
   b. Patrón de Tuberías y Filtros
   c. MVC (Modelo, Vista, Controlador)
   d. Patrón de N Capas
6. **En el patrón de tuberías y filtros, ¿cuál es el propósito de un filtro?**
   1. Almacenar datos.
   2. Procesar datos en una secuencia.
   3. Presentar datos al usuario.
   4. Conectar diferentes componentes.
7. **¿Cuál de las siguientes afirmaciones sobre el patrón de tuberías y filtros es verdadera?**
   a. Los componentes en una tubería siempre se ejecutan de manera paralela.
   b. Los componentes en una tubería pueden comunicarse directamente entre sí.
   c. Los componentes en una tubería son independientes y se conectan mediante un flujo de datos.
   d. Los componentes en una tubería deben estar escritos en el mismo lenguaje de programación.

8. ¿Qué tipo de aplicaciones se benefician especialmente del patrón de tuberías y filtros?

1. Aplicaciones monolíticas.
2. Aplicaciones de tiempo real.
3. Aplicaciones de procesamiento de datos en lotes.
4. Aplicaciones de juegos en línea.

9. ¿Cuál es el papel principal del servidor en la arquitectura cliente-servidor?

1. Presentar la interfaz de usuario al usuario final.
2. Realizar el procesamiento de negocios y gestionar los datos.
3. Actuar como intermediario entre el cliente y otros servidores.
4. Recopilar datos del usuario.

10. En una arquitectura cliente-servidor, ¿qué componente se encarga de presentar la interfaz de usuario al usuario final?

a. Cliente
b. Servidor
c. Controlador
d. Base de datos

11. ¿Cuál de las siguientes afirmaciones es cierta en una arquitectura cliente-servidor?

1. El servidor no tiene capacidad para procesar solicitudes de múltiples clientes al mismo tiempo.
2. El cliente y el servidor siempre se ejecutan en la misma máquina.
3. El servidor proporciona recursos o servicios a los clientes que solicitan información o acciones.
4. El cliente se encarga exclusivamente del procesamiento de negocios.

12. ¿Qué tipo de aplicaciones se benefician especialmente de la arquitectura cliente-servidor?

a. Aplicaciones de procesamiento por lotes.
b. Aplicaciones de juegos en línea.
c. Aplicaciones monolíticas.
d. Aplicaciones de simulación de vuelo.

13. ¿Cuál es el propósito principal del patrón M.V.C (Modelo, Vista, Controlador)?

1. Separar la interfaz de usuario de la lógica de presentación y el control de la aplicación.
2. Facilitar la comunicación entre clientes y servidores.
3. Dividir la aplicación en componentes que se ejecutan en diferentes máquinas.
4. Definir un conjunto de reglas para la transmisión de datos en una red.

14. ¿Qué componente del patrón M.V.C se encarga de la lógica de negocios y el procesamiento de datos?

1. Modelo
2. Vista
3. Controlador
4. Servidor

15. En el patrón M.V.C, ¿qué componente se encarga de presentar la interfaz de usuario al usuario final?

1. Modelo
2. Vista
3. Controlador
4. Servidor

16. ¿Qué beneficios proporciona el patrón M.V.C en el desarrollo de software?

a. Mayor complejidad y acoplamiento entre componentes.
b. Facilita la comunicación directa entre el modelo y la vista.
c. Separación de preocupaciones, lo que facilita el mantenimiento y la escalabilidad.
d. No tiene ningún impacto en la estructura de la aplicación.

17. ¿Cuál es el objetivo principal del patrón de arquitectura de N Capas?

a. Separar la interfaz de usuario de la lógica de presentación y el control de la aplicación.
b. Dividir la aplicación en tres capas: Presentación, Negocios y Datos.
c. Reducir la cantidad de capas en una aplicación.
d. Combinar la lógica de presentación y la lógica de negocios.

18. En la arquitectura de N Capas, ¿cuál de las siguientes capas se encarga de la lógica de presentación y la interacción con el usuario?

1. Capa de Presentación
2. Capa de Negocios
3. Capa de Datos
4. Capa de Infraestructura

19. ¿Cuál es una ventaja importante de utilizar una arquitectura de N Capas en el desarrollo de software?

1. Simplifica el diseño de interfaces de usuario.
2. Aumenta la complejidad y la dependencia entre componentes.
3. Facilita la reutilización de código y el mantenimiento.
4. Reduce el rendimiento de la aplicación.

20. En una aplicación de N Capas, ¿cuál es el propósito principal de la Capa de Datos?

a. Procesar la lógica de negocio de la aplicación.
b. Presentar la interfaz de usuario al usuario final.
c. Gestionar el acceso a la base de datos y el almacenamiento de datos.
d. Proporcionar servicios de red para la aplicación.

1.10 Conclusión

En resumen, la exploración exhaustiva De los temas vinculados al "Análisis de Software de Sistemas". Ha brindado una visión integral de los aspectos fundamentales en la ingeniería de software. Cada área abordada ha contribuido a comprender cómo se conciben, diseñan y transforman los sistemas de software con el paso del tiempo. Desde la definición que establece los cimientos hasta la evolución que resalta la constante transformación en respuesta a las tecnologías cambiantes y necesidades, se ha explorado un espectro completo.

Los estándares y principios de análisis han demostrado ser esenciales al proporcionar directrices que garantizan la calidad y la eficiencia en el proceso de desarrollo de sistemas. La inclusión de enfoques estructurados y orientados a objetos ha ampliado las perspectivas para abordar el diseño de software, teniendo en cuenta tanto la estructura interna como las interacciones entre componentes. Además, el análisis de sistemas de Inteligencia Artificial ha destacado las oportunidades para incorporar capacidades avanzadas y lograr sistemas más inteligentes y automatizados.

Las técnicas de análisis, tanto estructurales como orientadas a objetos, han proporcionado herramientas esenciales para comprender y diseñar sistemas complejos. Asimismo, la exploración de técnicas específicas para aplicaciones web y móviles ha resaltado la importancia de adaptar métodos de análisis a las plataformas tecnológicas contemporáneas. En resumen, esta exploración enriquecedora ha dotado a estudiantes y profesionales con las herramientas y conocimientos necesarios para enfrentar

los desafíos de crear sistemas de software efectivos, eficientes y de alta calidad en un entorno tecnológico en constante cambio.

Referencias

Bournissen, J.M. (1999). La evolución del software. *Enfoques, XI*(1 y 2), 123-140.

Ccori Huaman, W. (2018, 07 de septiembre). Los 10 patrones comunes de arquitectura de software. *Medium*. https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b

Dominguez Coutiño, L.A. (2012). *Análisis de sistemas de información*. Red Tercer Milenio.

James, O. (2007). *Diseño de Sistemas de Información Gerencial*. McGraw-Hill

Kendall, K. E., & Kendall, J. E. (2005). *Análisis y diseño de sistemas.* Robyn Goldenberg.

Maida, E.G., y Pacienzia, J. (2015). *Metodologías de desarrollo de software* [Tesis Licenciatura, Universidad Católica Argentina]

Sommerville, I. (2011). *Ingeniería de Software*. Pearson Edcucación.

§

Software Systems Analysis
Análise de sistemas de software

Mónica Elizabeth Páez Padilla
Instituto de Educación Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0006-1030-1394
monica.paez@intsuperior.edu.ec

Abstract

In this chapter we will address several topics related to "Software Systems Analysis". In this section, we will explore concepts such as the definition of software systems analysis, its evolution over time, the standards and principles that guide this process. In addition, the analysis of artificial intelligence systems will also be addressed, exploring how these systems also require an analytical approach.
Keywords: software, history, artificial intelligence.

Resumo

Neste capítulo, abordaremos vários tópicos relacionados à "Análise de sistemas de software". Nesta seção, exploraremos conceitos como a definição de análise de sistemas de software, sua evolução ao longo do tempo, os padrões e os princípios que orientam esse processo. Além disso, a análise de sistemas de inteligência artificial também será abordada, explorando como esses sistemas também exigem uma abordagem analítica.
Palavras-chave: software, história, inteligência artificial.