Religación Press
Ideas desde el Sur Global

Instituto Nelson Torres

ANÁLISIS Y DISEÑO DE SISTEMAS

MÓNICA ELIZABETH PÁEZ PADILLA
[EDITORA]

| Colección Programación |

Análisis y diseño de sistemas

Mónica Elizabeth Páez Padilla
(Editora)

RELIGACION PRESS
QUITO · 2023

Religación Press, es una iniciativa del Centro de Investigaciones CICSHAL-RELIGACIÓN.
Diseño, diagramación y portada: Religación Press.
CP 170515, Quito, Ecuador. América del Sur.
Correo electrónico: press@religacion.com
www.religacion.com

Análisis y diseño de sistemas

Analysis and design of systems
Análise e projeto de sistemas

Primera Edición: 2023 Mónica Elizabeth Páez Padilla©, Diego Javier Portilla Martinez©, Religación Press©

Editorial:	Religación Press
Materia Dewey:	005 - Programación. programas. datos de computadores
Clasificación Thema:	UMB - Algoritmos y estructuras de datos
	UMWS - Servicios web
BISCA:	COM084010 COMPUTERS / Business & Productivity Software
Público objetivo:	Profesional/Académico
Colección:	Programación
Soporte:	Digital
Formato:	Epub (.epub)/PDF (.pdf)
Publicado:	2023-12-07
ISBN:	978-9942-642-43-1

Disponible para su descarga gratuita en https://press.religacion.com

Citar como (APA 7)

Páez Padilla, M.E. (Ed). (2023). *Análisis y diseño de sistemas*. Religación Press. https://doi.org/10.46652/ReligacionPress.89

Patrocinio:
Instituto Superior Tecnológico Nelson Torres.
· Cayambe, Ecuador. Avenida Luis Cordero, Vía a Ayora.

Revisión por pares / Peer Review

Este libro fue sometido a un proceso de dictaminación por académicos externos. Por lo tanto, la investigación contenida en este libro cuenta con el aval de expertos en el tema, quienes han emitido un juicio objetivo del mismo, siguiendo criterios de índole científica para valorar la solidez académica del trabajo.

This book was reviewed by an independent external reviewers. Therefore, the research contained in this book has the endorsement of experts on the subject, who have issued an objective judgment of it, following scientific criteria to assess the academic soundness of the work.

Sobre los autores

Mónica Elizabeth Páez Padilla
Instituto de Educación Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0006-1030-1394
monica.paez@intsuperior.edu.ec
Ingeniero en Sistemas de Información con una Maestría en Seguridad Informática.

Diego Javier Portilla Martínez
Investigador independiente | Cayambe | Ecuador
https://orcid.org/0009-0008-7295-6227
dijapm@gmail.com
Ingeniero en Informática cursando una Maestría en Educación Mediada por TIC.

Resumen

El objetivo del libro es brindar a los estudiantes las competencias necesarias para el desarrollo de software y la implementación de tecnología, tiene un enfoque integral, ya que se relaciona de manera interdisciplinaria con otras materias. Esto permite que los estudiantes adquieran habilidades que son esenciales para su formación profesional. Se espera que los estudiantes sean capaces de: Modelar los procesos de desarrollo de software, ya sea a través de la UML o enfoques jerárquicos, analizar y diseñar sistemas informáticos, aprender sobre el ciclo de vida del software, desde la planificación hasta la implementación y el mantenimiento, aprender, aplicar la cultura profesional en el desarrollo de software, aplicar soluciones informáticas para resolver problemas. Los contenidos de desarrollo de software e Implementación de Tecnología es una asignatura esencial para la formación de profesionales en el ámbito de la informática. Proporciona a los estudiantes las competencias necesarias para el desarrollo de software y la implementación de tecnología, lo que les permitirá desempeñarse exitosamente en el campo laboral.

Palabras claves: software, formación profesional, informática, diseño.

Abstract

The objective of the book is to provide students with the necessary skills for software development and technology implementation. It has an integral approach, since it is related in an interdisciplinary way with other subjects. This allows students to acquire skills that are essential for their professional training. Students are expected to be able to: Model software development processes, either through UML or hierarchical approaches, analyze and design computer systems, learn about the software life cycle, from planning to implementation and maintenance, learn, apply professional culture in software development, apply computer solutions to solve problems. The contents of Software Development and Technology Implementation is an essential subject for the training of professionals in the field of computer science. It provides students with the necessary skills for software development and technology implementation, which will allow them to perform successfully in the labor field.

Key words: software, vocational training, computer science, design.

Resumo

O objetivo do livro é fornecer aos alunos as habilidades necessárias para o desenvolvimento de software e a implementação de tecnologia. Ele tem uma abordagem holística, pois se relaciona de forma interdisciplinar com outras matérias. Isso permite que os alunos adquiram habilidades essenciais para sua formação profissional. Espera-se que os alunos sejam capazes de: modelar processos de desenvolvimento de software, seja por meio de UML ou de abordagens hierárquicas, analisar e projetar sistemas de computador, conhecer o ciclo de vida do software, desde o planejamento até a implementação e a manutenção, aprender e aplicar a cultura profissional no desenvolvimento de software, aplicar soluções de computador para resolver problemas. O conteúdo de Desenvolvimento de Software e Implementação de Tecnologia é uma matéria essencial para o treinamento de profissionais na área de ciência da computação. Ele fornece aos alunos as competências necessárias para o desenvolvimento de software e a implementação de tecnologia, o que lhes permitirá atuar com sucesso no mercado de trabalho.

Palavras-chave: software, treinamento vocacional, ciência da computação, design.

.

Contenido

Figuras

| Colección Programación |

Análisis y diseño de sistemas

Prólogo

La asignatura aporta al perfil de los estudiantes a través de innovadoras metodologías en el desarrollo de software y la implementación de tecnología. Es importante destacar que esta asignatura tiene un impacto integral en el aprendizaje al relacionarse de manera interdisciplinaria con otras materias. Esto se logra mediante la modelación de procesos de desarrollo de software, ya sea a través de la UML o enfoques jerárquicos, tanto en contextos específicos como generales. Esto capacita al estudiante para adquirir y mejorar competencias en cultura profesional y en análisis y diseño de sistemas, habilidades esenciales en su formación.

La esencia de esta asignatura se basa en la comprensión de los sistemas como conjuntos de elementos interrelacionados y dinámicos que colaboran para lograr objetivos, operando en base a entradas y produciendo resultados. Esto ocurre en un entorno único, distinto de otros sistemas. La asignatura proporciona la capacidad de analizar problemas actuales y aplicar soluciones informáticas. Además, se brinda conocimiento sobre el ciclo de vida del software y se fomenta la aplicación de soluciones basadas en la consultoría tecnológica

Introducción

La asignatura de Análisis y Diseño de Sistemas está considerada en la malla curricular de la carrera de Tecnología en Desarrollo de Software consta de 3 créditos y es parte del campo de profesionalización de la carrera.

Esta asignatura se imparte con el objetivo de incorporar en el profesional en formación el concepto de conocer sobre análisis y diseño de sistemas, cual es el aporte al perfil de los estudiantes a través de innovadoras metodologías. Esto se logra mediante la modelación. Esto capacita al estudiante para adquirir y mejorar competencias en cultura profesional y en análisis y diseño de sistemas, habilidades esenciales en su formación.

La esencia de esta asignatura se basa en la comprensión de los sistemas como conjuntos de elementos interrelacionados y dinámicos que colaboran para lograr objetivos, operando en base a entradas y produciendo resultados. Esto ocurre en un entorno único, distinto de otros sistemas. La asignatura proporciona la capacidad de analizar problemas actuales y aplicar soluciones informáticas.

A continuación, se describen algunas de las habilidades y conocimientos que los estudiantes adquirirán en esta asignatura:

- Habilidades de disección y planificación de sistemas
- Habilidades de pacto de proyectos
- Habilidades de vía
- Habilidades de tarea en utilería

- Habilidades de prospección
- Habilidades de seguridad de problemas

Esta asignatura es una excelente oportunidad para que completen esta asignatura estarán bien preparados para trabajar en una variedad de roles relacionados con el desarrollo de software, la implementación de tecnología y la consultoría tecnológica.

Además de las habilidades y conocimientos técnicos, los estudiantes también desarrollarán una serie de habilidades blandas.

Estas habilidades blandas son esenciales para tener éxito en cualquier entorno laboral, pero son especialmente importantes en la industria de la tecnología. La industria de la tecnología es un entorno dinámico y cambiante, y las personas que trabajan en este campo deben ser capaces de adaptarse rápidamente a los cambios. Las habilidades blandas les ayudan a hacer esto.

Objetivos

Planificar el desarrollo de los sistemas de información mediante la comprensión y la especificación en detalle de lo que debe hacer un sistema y cómo deben implementarse los diferentes componentes de este para trabajar conjuntamente

Orientaciones básicas para el estudio.

1. Pruebas escritas: Las pruebas escritas son evaluaciones en las que los estudiantes responden preguntas en forma de ensayos,

preguntas de opción múltiple, verdadero/falso, completar espacios en blanco, entre otros. Estas pruebas miden la comprensión, el conocimiento.

2. Exámenes orales: En los exámenes orales, los estudiantes presentan su conocimiento de manera verbal, respondiendo preguntas formuladas por el profesor. Esto permite evaluar su capacidad para comunicar ideas de manera clara y coherente, así como su comprensión profunda del tema.

3. Trabajos escritos: El trabajo escrito puede incluir ensayos, informes, estudios de casos e investigaciones, entre otros. Estas tareas evalúan la capacidad de los estudiantes para encontrar, analizar y comunicar información por escrito.

4. Presentaciones: Las presentaciones requieren que los estudiantes expongan un tema frente a sus compañeros o profesores. Evalúan la habilidad de hablar en público, la organización de la información y la claridad en la presentación.

5. Proyectos: Los proyectos son tareas más extensas que pueden involucrar la creación de un producto, como un software, una maqueta, una investigación completa, etc. Evalúan la capacidad de planificación, ejecución y presentación del proyecto finalizado.

6. Portafolios: Los portafolios son colecciones organizadas de trabajos realizados por el estudiante a lo largo de un período de tiempo. Pueden incluir ensayos, proyectos, ejercicios y otras muestras de trabajo que demuestren el progreso y la amplitud del aprendizaje.

7. Pruebas prácticas: Estas pruebas se utilizan en disciplinas que requieren habilidades prácticas, como ciencias experimentales, informática y artes. Los estudiantes demuestran sus habilidades al realizar tareas reales en un entorno controlado.

8. Evaluaciones en línea: En la educación en línea, se utilizan plataformas y herramientas digitales para realizar pruebas y cuestionarios en línea. Esto puede incluir preguntas de opción múltiple, preguntas de verdadero/falso y ejercicios de respuestas cortas.

9. Evaluación entre pares: Los estudiantes evalúan el trabajo de sus compañeros según criterios específicos. Esta forma de evaluación fomenta la colaboración y el desarrollo de habilidades críticas de análisis.

10. Evaluación continua: Se trata de evaluar el progreso del estudiante a lo largo de un período, considerando la participación en clase, la asistencia, las tareas y otros elementos. Esto proporciona una imagen más completa del aprendizaje a lo largo del tiempo, es importante que los mecanismos de evaluación sean consistentes con los objetivos educativos y proporcionen una evaluación equitativa y justa para todos los estudiantes. La variedad en los métodos de evaluación ayuda a medir una gama más amplia de habilidades y competencias, y permite a los estudiantes demostrar su conocimiento y comprensión de diversas formas.

Proceso de enseñanza para el logro de competencias

Competencias de la carrera o asignatura

1. Conocer el proceso de desarrollo de software, actividades, recursos, diseño y desarrollo de aplicaciones a medida.

2. Conocer los principios y técnicas de análisis para el desarrollo de un proyecto software.

3. Comprender y aplica patrones de diseño y arquitectónicos en aplicaciones software

4. Conocer y aplica diagramas de especificación del diseño de software de alto nivel.

Planificación para el trabajo del alumno

Organiza tus actividades laborales, personales y familiares para darte tiempo de desarrollar actividades autodirigidas, como vinculaciones obligatorias a materias académicas.

Recomendamos reservar dos horas diarias para actividades académicas: revisión de material, preparación de trabajos independientes, evaluaciones y comunicación obligatoria con los profesores.

La asignatura consta de dos partes divididas en dos semestres y se cursa durante 16 semanas académicas.

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.

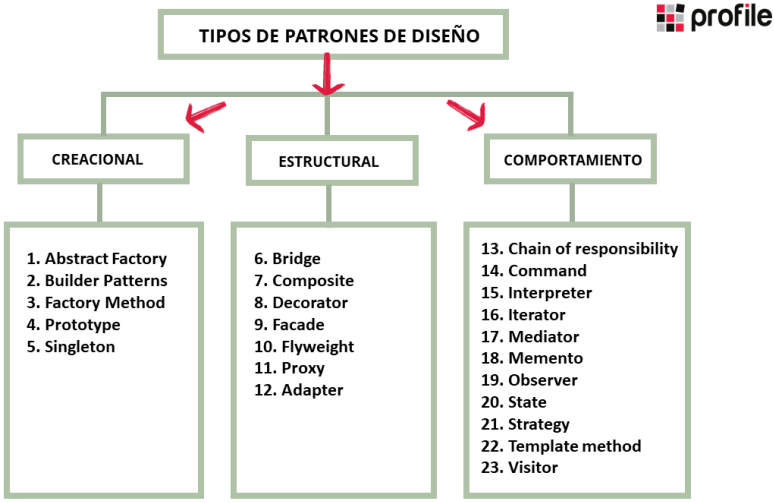
c. Un enfoque que solo funciona en dispositivos móviles.

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:

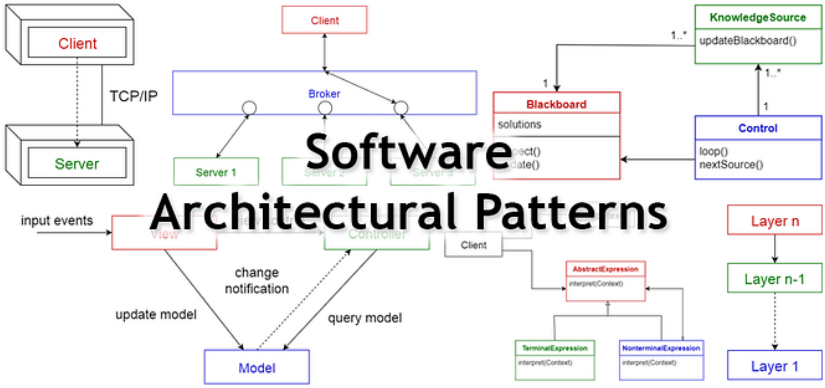
1. Patrón Mediator
2. Patrón Visitor
3. Patrón Command
4. Patrón Observer

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

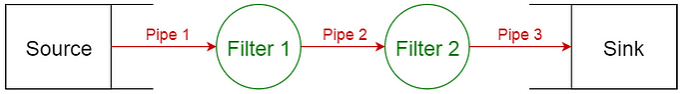
Este patrón encuentra aplicabilidad en la organización de sistemas que generan y procesan un cuadro de datos. Cada distancia de procesamiento se encapsula en un integrante de brebaje. Los datos destinados a emplumar son dirigidos a través de tuberías. Estas tuberías pueden emplearse tanto para el almacenaje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

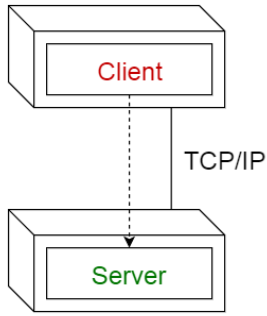
1.9.2 Cliente Servidor

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:

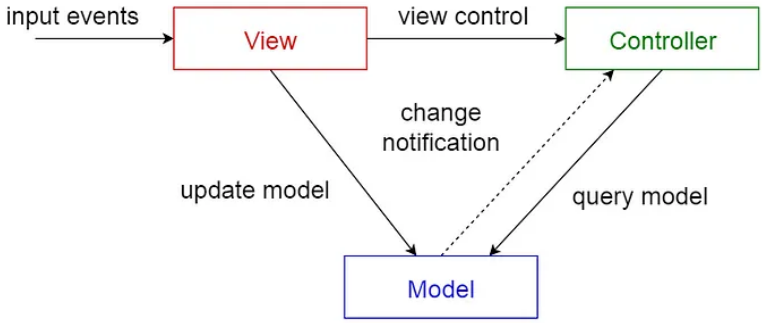
- Modelo: contiene la funcionalidad y los datos fundamentales.
- Vista: presenta la información al usuario (y es posible definir múltiples vistas).
- Controlador: gestiona la entrada del usuario.

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).

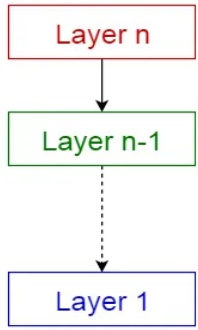
Capa de lógica de negocios (igualmente llamada capa de dominio).

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.

Capítulo 2

Conceptos de Diseño

Mónica Elizabeth Páez Padilla

Resumen

En este capítulo, exploraremos los "Conceptos de Diseño", analizando detalles relevantes como las "Características y Principios de Diseño" que establecen bases para sistemas eficientes y flexibles, y los "Estándares de Diseño" que aseguran coherencia y calidad. Además, se profundizará en el "Diseño de Arquitectura de Software", considerando enfoques estructurados y orientados a objetos para sistemas adaptables. También se abordará el "Diseño de Interfaz de Usuario", resaltando su importancia en la experiencia del usuario y la creación de interfaces visuales e intuitivas. Por último, se explorarán los "Modelos de Interfaz de Usuario", incluyendo interfaces de línea de comandos, texto, gráficas y de voz. En resumen, este capítulo será una inmersión completa en los fundamentos del diseño en análisis y diseño de sistemas, estableciendo una base sólida para soluciones efectivas.

Palabras claves: principios de diseño, interfaz usuario, arquitectura de software.

Páez Padilla, M.E. (2023). Conceptos de Diseño. En M.E. Páez Padilla (ed). *Análisis y diseño de sistemas.* (pp. 86-121). Religación Press. http://doi.org/10.46652/religacionpress.89.c83

2.1 Concepto de Diseño

Dentro del campo de estudio relacionado con el análisis y la creación de sistemas., el concepto de diseño alude al procedimiento creativo y organizado que implica la formulación de soluciones tangibles y representaciones concretas para satisfacer las exigencias y necesidades que han sido identificadas durante la etapa de análisis. Este momento corresponde a la transformación de la información recopilada en un esquema detallado que delineará el procedimiento de construcción del sistema (Coronel, 2012).

En esta materia, el diseño involucra la generación de representaciones visuales y técnicas que delinean el funcionamiento proyectado del sistema, las interconexiones entre sus componentes, el enfoque de la interfaz para los usuarios y el planteamiento de soluciones a los problemas identificados. Esta tarea engloba la toma de decisiones relativas a la arquitectura del sistema, la organización de los datos, la circulación de la información, las interfaces de usuario, las reglas de operación y otros aspectos esenciales.

La ejecución del diseño se rige por los preceptos y principios reconocidos en la ingeniería de software, con el objetivo de asegurar que el sistema resultante se distinga por su eficacia, su mantenibilidad, su escalabilidad y su adhesión a los objetivos establecidos. Asimismo, el proceso de diseño debe abarcar factores tales como la usabilidad, la seguridad, el desempeño y la integración con otros sistemas.

En resumen, en el ámbito de análisis y diseño de sistemas, la fase de diseño conlleva la concepción detallada de la propuesta para la construcción del sistema, considerando de manera exhaustiva todas las implicaciones técnicas y funcionales necesarias para garantizar la materialización de un sistema eficaz y exitoso (Coronel, 2012).

2.1.1 Características y principios de diseño

Los atributos y fundamentos de diseño en el dominio del análisis y diseño de sistemas desempeñan un papel fundamental en garantizar que los sistemas resultantes sean eficaces, eficientes y capaces de satisfacer los requisitos y demandas de los usuarios. A continuación, se enumeran algunos de los atributos y fundamentos esenciales:

Características de Diseño:

- Mantenibilidad: El sistema debe ser diseñado de manera que permita futuras modificaciones y actualizaciones de manera fácil y sin generar impactos no deseados.
- Flexibilidad: El diseño debe ser lo suficientemente adaptable para acomodar cambios y evoluciones en los requisitos del sistema con el menor esfuerzo posible.

- Usabilidad: El diseño debe priorizar la experiencia del usuario, creando interfaces y flujos de trabajo intuitivos y fáciles de usar.

- Seguridad: La seguridad del sistema debe ser considerada en todas las etapas del diseño para proteger la información y prevenir posibles vulnerabilidades.

- Escalabilidad: El diseño debe permitir que el sistema pueda crecer y expandirse para manejar incrementos en la carga de trabajo sin comprometer su rendimiento.

Según Schumacher (2023), nos señala que los principios de diseño son:

Principios de Diseño:

Principio de Responsabilidad Única (SRP): Cada componente o clase debe tener una única responsabilidad, lo que facilita la comprensión, el mantenimiento y la reutilización.

Principio de Abierto/Cerrado (OCP): Las entidades del sistema (clases, módulos, etc.) deben estar abiertas a extensiones, pero cerradas a modificaciones, lo que promueve la extensibilidad sin afectar el código existente.

Principio de Sustitución de Liskov (LSP): Los objetos de una subclase deben ser capaces de reemplazar a los objetos de la clase padre sin afectar la integridad del programa.

Principio de Segregación de Interfaces (ISP): Los clientes no deben depender de interfaces que no utilizan. En lugar de interfaces monolíticas, se deben preferir interfaces más específicas.

Principio de Inversión de Dependencia (DIP): Los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de abstracciones. Además, los detalles deben depender de las abstracciones, no al revés.

Principio de Separación de Intereses (SoC): Los diferentes aspectos del sistema (como la lógica de negocio, la presentación y el almacenamiento) deben ser separados en módulos distintos para facilitar el mantenimiento y la evolución independiente.

Principio de Composición sobre Herencia: Se debe preferir la composición (combinación de objetos) en lugar de la herencia (extender clases existentes) para construir sistemas más flexibles y reutilizables.

Estas características y principios de diseño guían la creación de sistemas robustos, mantenibles y adaptativos en el campo del análisis y diseño de sistemas.

2.1.2 Estándares de diseño

En el contexto de análisis y diseño de sistemas, los estándares de diseño son pautas y directrices que se establecen para asegurar la coherencia, calidad y eficiencia en el proceso de diseño de sistemas. Estos estándares abarcan diversas áreas y aspectos

del diseño, desde la arquitectura del sistema hasta la interfaz de usuario.

Algunos ejemplos de estándares de diseño incluyen (Coronel, 2012).

Estándares de Diseño de Interfaz de Usuario: Establecen principios para la disposición de elementos, el uso de colores, tipografías, iconos y la organización de información en la interfaz para lograr una experiencia de usuario coherente y efectiva.

Estándares de Diseño de Arquitectura: Definen la estructura general del sistema, la organización de los componentes y la interacción entre ellos. Estos estándares pueden abordar la modularidad, escalabilidad y reutilización de componentes.

Estándares de Nomenclatura: Establecen convenciones para nombrar variables, funciones, clases y otros elementos del código, lo que facilita la comprensión y colaboración entre los miembros del equipo.

Estándares de Documentación: Establecen el formato y el contenido de la documentación técnica, como diagramas de flujo, diagramas de clases, descripciones de funciones y otros artefactos que documentan el diseño del sistema.

Estándares de Seguridad: Definen las medidas de seguridad que deben implementarse en el diseño para proteger la información y prevenir vulnerabilidades.

Estándares de Rendimiento: Establecen requisitos y metas relacionados con el rendimiento del sistema, como tiempos de respuesta y capacidad de manejo de cargas.

Estándares de Usabilidad: Definen criterios para asegurar que la interfaz de usuario sea intuitiva y fácil de usar, considerando la navegación, la interacción y la experiencia global del usuario.

Estándares de Codificación: Establecen directrices para el estilo de codificación, la indentación, el uso de comentarios y otras prácticas que hacen que el código sea más legible y mantenible.

Estándares de Pruebas: Definen enfoques para la planificación y ejecución de pruebas, incluyendo pruebas unitarias, de integración y de aceptación, con el objetivo de asegurar la calidad del sistema.

Estándares de Integración: Establecen prácticas para la integración de componentes y sistemas, asegurando que las partes funcionen correctamente juntas.

Los estándares de diseño son fundamentales para garantizar la consistencia y la calidad en el proceso de diseño de sistemas, así como para facilitar la colaboración entre los miembros del equipo y el mantenimiento a largo plazo de los sistemas desarrollados.

Autoevaluación 5

1. **¿Cuál es el propósito principal del diseño en el análisis y diseño de sistemas?**
   a. Generar representaciones visuales y técnicas.
   b. Identificar necesidades y exigencias.
   c. Recopilar información detallada.
   d. Definir la arquitectura del sistema.
2. **¿Cuál de las siguientes consideraciones NO es parte del proceso de diseño en ingeniería de software?**
   1. Usabilidad
   2. Desarrollo de código fuente
   3. Seguridad
   4. Integración con otros sistemas
3. **¿Qué engloba la fase de diseño en el análisis y diseño de sistemas?**
   a. Toma de decisiones relativas a la arquitectura del sistema y organización de los datos.
   b. Identificación de necesidades y exigencias.
   c. Evaluación de la eficacia del sistema resultante.
   d. Recopilación de información detallada sobre el sistema.
4. **¿Cuál de las siguientes características de diseño se refiere a la capacidad del sistema para utilizar eficientemente los recursos disponibles?**
   1. Mantenibilidad
   2. Flexibilidad
   3. Eficiencia
   4. Usabilidad

5. **¿Qué principio de diseño establece que las entidades del sistema deben estar abiertas a extensiones, pero cerradas a modificaciones?**
    a. Principio de Responsabilidad Única (SRP)
    b. Principio de Abierto/Cerrado (OCP)
    c. Principio de Sustitución de Liskov (LSP)
    d. Principio de Inversión de Dependencia (DIP)
6. **¿Cuál de los siguientes principios de diseño se enfoca en la separación de los diferentes aspectos del sistema en módulos distintos?**
    1. Principio de Separación de Intereses (SoC)
    2. Principio de Composición sobre Herencia
    3. Principio de Segregación de Interfaces (ISP)
    4. Principio de Responsabilidad Única (SRP)
7. **¿Cuál es el propósito del principio de Inversión de Dependencia (DIP) en el diseño de sistemas?**
    1. Los módulos de alto nivel deben depender de los módulos de bajo nivel.
    2. Los detalles deben depender de las abstracciones.
    3. Promover la extensibilidad sin afectar el código existente.
    4. Facilitar la comprensión y el mantenimiento del código.
8. **¿Qué característica de diseño se enfoca en la protección de la información y la prevención de vulnerabilidades en el sistema?**
    1. Escalabilidad
    2. Usabilidad
    3. Seguridad
    4. Flexibilidad

9. **¿Cuál es el propósito principal de los estándares de diseño en el análisis y diseño de sistemas?**
   a. Aumentar la complejidad del sistema.
   b. Asegurar la coherencia, calidad y eficiencia en el proceso de diseño.
   c. Establecer una interfaz de usuario atractiva.
   d. Reducir la modularidad del sistema.
10. **¿Qué tipo de estándares se enfocan en la disposición de elementos, el uso de colores y la organización de información en la interfaz de usuario?**
   a. Estándares de Arquitectura
   b. Estándares de Codificación
   c. Estándares de Diseño de Interfaz de Usuario
   d. Estándares de Pruebas
11. **¿Qué función cumplen los estándares de nomenclatura en el proceso de diseño?**
   1. Definir la estructura general del sistema.
   2. Establecer requisitos de rendimiento.
   3. Facilitar la comprensión y colaboración entre el equipo.
   4. Planificar y ejecutar pruebas de aceptación.
12. **¿Por qué son importantes los estándares de seguridad en el diseño de sistemas?**
   1. Para definir la estructura del sistema.
   2. Para mejorar el rendimiento del sistema.
   3. Para proteger la información y prevenir vulnerabilidades.
   4. Para determinar la eficiencia de la interfaz de usuario.

13. ¿Qué área abordan los estándares de documentación en el diseño de sistemas?

a. Definición de nombres de variables.
b. Estructura general del sistema.
c. Formato y contenido de la documentación técnica.
d. Requisitos de rendimiento del sistema.

14. ¿Cuál es el propósito de los estándares de rendimiento en el diseño de sistemas?

a. Facilitar la colaboración entre el equipo.
b. Definir la estructura del sistema.
c. Establecer metas relacionadas con el rendimiento del sistema.
d. Planificar y ejecutar pruebas de unidad.

15. ¿Para qué son esenciales los estándares de usabilidad en el diseño de sistemas?

1. Para definir la estructura del sistema.
2. Para mejorar el rendimiento del sistema.
3. Para asegurar que la interfaz de usuario sea intuitiva y fácil de usar.
4. Para establecer convenciones de nomenclatura.

2.2 Diseño de arquitectura de software

El proceso del "Diseño de Arquitectura de Software" representa una fase crucial en el ámbito de la asignatura de análisis y diseño de sistemas, donde se configura la estructura y disposición general del sistema de software que se busca crear. En esta etapa, se enfoca en la elaboración meticulosa de un plan exhaustivo que delineará la forma en que los distintos elementos del sistema interactuarán entre sí para cumplir con los preceptos y metas previamente identificados durante la fase de análisis (Laudon & Laudon, 2020).

Durante esta etapa, es posible emplear diferentes enfoques de diseño, tales como el diseño estructurado y el diseño orientado a objetos. El diseño estructurado busca segmentar el sistema en módulos y submódulos, precisando sus funciones y responsabilidades. Por otro lado, el diseño orientado a objetos traza el sistema mediante la utilización de objetos, clases y relaciones entre ellos, lo cual facilita representar de manera más fiel la realidad y las interacciones dentro del sistema.

Además, el proceso de diseño de arquitectura de software implica la toma de decisiones en relación con aspectos como la intercomunicación entre los componentes, la administración de datos, la escalabilidad, la seguridad y la eficiencia del sistema. Para visualizar la arquitectura, es posible recurrir a diagramas y modelos visuales, como los diagramas de componentes, de despliegue y de secuencia.

La creación de una arquitectura de software sólida es crucial para lograr sistemas resistentes, fáciles de mantener y adaptables. Además, facilita una comprensión más clara de cómo interactúan los componentes y cómo se logran los objetivos del sistema. En resumen, el proceso de diseño de la arquitectura de software se erige como un componente fundamental en el campo del análisis y diseño de sistemas, ya que establece las bases para el desarrollo exitoso de sistemas de software que sean funcionales y eficaces (Laudon & Laudon, 2020).

2.2.1 Diseño Estructurado

El enfoque "Diseño Estructurado" en el ámbito del análisis y diseño de sistemas representa una metodología fundamental cuyo objetivo es generar soluciones organizadas y eficaces mediante la descomposición de un sistema en elementos más pequeños y manejables. Esta técnica se centra en la subdivisión del sistema en módulos interconectados, cada uno de los cuales posee funciones y responsabilidades claramente definidas (James, 2007).

En el transcurso del proceso de diseño estructurado, se identifican las funciones primordiales del sistema y se organizan en módulos lógicos, lo que simplifica la comprensión y la futura mantención del sistema. Cada módulo desempeña una tarea específica y se relaciona con otros módulos a través de interfaces precisas. Esto impulsa la modularidad y favorece la reutilización de componentes en distintos proyectos.

Para representar gráficamente el diseño estructurado, se emplean diagramas como el "Diagrama de Estructura de Programa" (DEP), que representa la conexión entre los módulos y la jerarquía de llamadas entre ellos. Este enfoque asiste a los desarrolladores en la comprensión de cómo interactúan las partes del sistema y en la implementación de cambios con un impacto limitado en otras áreas

El diseño estructurado resulta particularmente valioso en sistemas complejos, al fragmentar la complejidad en elementos más manejables y al fomentar la colaboración entre los integrantes del equipo. No obstante, puede presentar limitaciones en la representación de relaciones entre objetos en sistemas más dinámicos. En su conjunto, el diseño estructurado conforma una herramienta esencial en el repertorio del análisis y diseño de sistemas, proporcionando un enfoque metódico para encarar proyectos de desarrollo de software (James, 2007).

2.2.2 Diseño Orientado a Objetos

El "Diseño Orientado a Objetos" en el contexto de análisis y diseño de sistemas constituye un enfoque crucial que se centra en la representación de sistemas mediante objetos y sus interacciones. Este enfoque se basa en la idea de modelar el mundo real a través de objetos con propiedades y comportamientos, creando así un sistema más cercano a la realidad y de mayor flexibilidad (Coronel & Morris, 2017).

Durante el proceso de diseño orientado a objetos, se identifican las entidades relevantes en el sistema y se modelan como clases, definiendo sus atributos y métodos. Las clases se organizan en jerarquías de herencia, lo que permite la reutilización de código y la captura de relaciones entre conceptos. Los objetos son instancias de estas clases y colaboran entre sí a través de mensajes, lo que simula la comunicación entre objetos en el mundo real.

El diseño orientado a objetos se apoya en diagramas como el "Diagrama de Clases", que visualiza las clases, sus atributos y relaciones. Además, los "Diagramas de Secuencia" y "Diagramas de Colaboración" detallan las interacciones entre objetos en diferentes escenarios.

Este enfoque promueve la encapsulación, que protege la implementación interna de un objeto y expone solo su interfaz pública. También fomenta el polimorfismo, que permite que diferentes clases implementen métodos con el mismo nombre de manera específica. La herencia es otra característica clave, permitiendo que una clase herede atributos y métodos de otra.

El diseño orientado a objetos es altamente eficaz para modelar sistemas complejos y promover la reutilización de componentes. Sin embargo, su implementación exitosa requiere un profundo entendimiento de los principios orientados a objetos y una planificación cuidadosa para crear una arquitectura coherente y escalable. En resumen, el diseño orientado a objetos es un pilar esencial en la materia de análisis y diseño de sistemas, proporcionando un enfoque poderoso para crear sistemas flexibles y adaptativos que se asemejan a la realidad (Coronel & Morris, 2017).

Autoevaluación 6

1. **¿Qué representa el proceso de «Diseño de Arquitectura de Software» en el ámbito de la asignatura de análisis y diseño de sistemas?**

    1. Una fase opcional en el desarrollo de software.
    2. Una etapa donde se configura la estructura general del sistema de software.
    3. Un paso exclusivo del diseño orientado a objetos.

2. **¿Cuál es el enfoque del diseño estructurado en el proceso de diseño de arquitectura de software?**

    a. Utilizar objetos y clases para modelar el sistema.
    b. Segmentar el sistema en módulos y submódulos.
    c. Representar la realidad del sistema de manera fiel.

3. **¿En qué se diferencia el diseño orientado a objetos del diseño estructurado en el proceso de diseño de arquitectura de software?**

    1. El diseño orientado a objetos utiliza objetos y clases, mientras que el diseño estructurado se enfoca en módulos.
    2. El diseño estructurado es más adecuado para sistemas pequeños.
    3. El diseño orientado a objetos no utiliza diagramas visuales.

4. **¿Cuáles son algunos de los aspectos clave que se deben considerar durante el proceso de diseño de arquitectura de software?**

   a. La elección del lenguaje de programación.
   b. La ubicación geográfica de los desarrolladores.
   c. La intercomunicación entre componentes, la administración de datos, la escalabilidad y la seguridad.

5. **¿Por qué se considera esencial una arquitectura de software sólida en el desarrollo de sistemas de software?**

   1. Porque facilita la creación de documentación extensa.
   2. Porque permite comprender cómo interactúan los elementos y alcanzar los objetivos del sistema.
   3. Porque reduce el tiempo de desarrollo.

6. **¿Cuál es el objetivo principal del enfoque "Diseño Estructurado" en el análisis y diseño de sistemas?**

   a. Generar soluciones caóticas.
   b. Descomponer un sistema en elementos más pequeños y manejables.
   c. Crear sistemas sin módulos interconectados.

7. **¿Qué se hace durante el proceso de diseño estructurado para simplificar la comprensión y la mantención del sistema?**

   1. Se eliminan todos los módulos interconectados.
   2. Se organizan las funciones primordiales en módulos lógicos.
   3. Se crean interfaces complejas entre los módulos.

8. **¿Cómo se fomenta la reutilización de componentes en el diseño estructurado?**

   a. No se fomenta la reutilización de componentes.
   b. Cada módulo desempeña una tarea específica.
   c. No se relacionan los módulos entre sí.

9. **¿Qué tipo de diagrama se utiliza comúnmente para representar gráficamente el diseño estructurado?**
    1. Diagrama de flujo.
    2. Diagrama de Gantt.
    3. Diagrama de Estructura de Programa (DEP).

10. **¿Cuándo resulta especialmente valioso el diseño estructurado en el desarrollo de sistemas?**
    a. En sistemas dinámicos.
    b. En sistemas sin colaboración entre equipos.
    c. En sistemas complejos, al fragmentar la complejidad en elementos más manejables y fomentar la colaboración entre los integrantes del equipo.

11. **¿Qué es el «Diseño Orientado a Objetos» en el contexto del análisis y diseño de sistemas?**
    1. Un enfoque que no utiliza objetos ni interacciones.
    2. Un enfoque que se centra en la representación de sistemas mediante objetos y sus interacciones.
    3. Un enfoque que se basa en la programación lineal.

12. **¿Cómo se modelan las entidades relevantes en el diseño orientado a objetos?**
    a. Como funciones independientes.
    b. Como objetos con propiedades y comportamientos en forma de clases.
    c. Como módulos interconectados.

13. **¿Qué permite la herencia en el diseño orientado a objetos?**
    1. La creación de objetos.
    2. La reutilización de código y la captura de relaciones entre conceptos.
    3. La encapsulación de métodos.

14. **¿Qué tipo de diagramas se utilizan para visualizar las clases y sus relaciones en el diseño orientado a objetos?**

    1. Diagramas de flujo.
    2. Diagramas de estructura.
    3. Diagramas de clases.

15. **¿Qué características clave fomenta el diseño orientado a objetos, como la encapsulación, el polimorfismo y la herencia?**

    a. La obsolescencia del código.
    b. La confusión entre objetos.
    c. La flexibilidad y la reutilización de componentes

2.3 Diseño Interfaz de usuario

El "Diseño de Interfaz de Usuario" juega un rol crucial en el ámbito de análisis y diseño de sistemas al focalizarse en forjar interacciones efectivas y atractivas entre los usuarios y el sistema. Esta etapa se dedica a concebir cómo los usuarios interactuarán con la aplicación, garantizando que la experiencia sea fácil, eficiente y gratificante (Tavera, 2018).

En el transcurso del proceso de diseño de interfaz de usuario, se toman en cuenta aspectos tales como la disposición de elementos en la pantalla, la elección de colores, fuentes, íconos y la navegación entre pantallas. El objetivo es crear una interfaz que permita a los usuarios cumplir sus tareas de manera fluida y sin dificultades, maximizando la usabilidad y minimizando la curva de aprendizaje.

La trascendencia de la interfaz de usuario radica en su rol como el principal punto de contacto entre los usuarios y el sistema, impactando directamente en su percepción y satisfacción. Un diseño que sea visualmente atractivo y funcional puede influir positivamente en la aceptación y adopción del sistema por parte de los usuarios.

Durante la etapa de diseño de interfaz de usuario, es posible emplear wireframes y mockups para visualizar la estructura y disposición de elementos en pantalla antes de llevarlo a cabo. Se busca alcanzar un equilibrio entre un diseño que sea visualmente agradable y una experiencia de usuario que sea intuitiva y eficiente (Cáceres, 2014).

Asimismo, el diseño de interfaz de usuario debe tener en consideración la accesibilidad, garantizando que el sistema sea usable por personas con discapacidades. Además, el diseño debe ser coherente con la identidad visual de la marca y en sintonía con las expectativas y necesidades de los usuarios.

En síntesis, el diseño de interfaz de usuario es un componente esencial en la materia de análisis y diseño de sistemas, ya que contribuye en gran medida a la usabilidad, aceptación y éxito del sistema (Paredes Hernández & Velasco Espitia, 1998).

2.3.1 La importancia de la interfaz de usuario

El diseño de la interfaz de usuario, que también se conoce como diseño de la interfaz o ingeniería de la interfaz, implica la actividad de establecer múltiples aspectos que afectan la apariencia y el funcionamiento externo de las interfaces de usuario en diversos tipos de sistemas, que van desde computadoras y dispositivos móviles hasta software y páginas web. Esta disciplina se alinea con el diseño industrial y busca optimizar la usabilidad y la experiencia del usuario. Su propósito es simplificar y agilizar la interacción entre el usuario y el sistema, manteniendo el enfoque en los objetivos del usuario (Domínguez Coutiño, 2012).

En esta perspectiva centrada en el usuario, el diseño gráfico y la tipografía se combinan para mejorar tanto la usabilidad como la estética, equilibrando la funcionalidad técnica con elementos visuales. El diseño de la interfaz de usuario abarca una

diversidad de disciplinas, como diseño gráfico, diseño industrial, diseño web y ergonomía, y su aplicación abarca desde sistemas informáticos hasta proyectos de envergadura como el desarrollo de aeronaves comerciales. El objetivo final es lograr una comunicación efectiva y eficiente, así como una adaptabilidad a las necesidades cambiantes de los usuarios, al tiempo que se mantenga una operación técnica óptima (James, 2007).

2.3.2 Diseño visual, funcional e intuitivo

En el desarrollo de software, el diseño abarca tres elementos esenciales: visual, funcional e intuitivo. En primer lugar, el diseño visual se enfoca en la estética y presentación gráfica de la interfaz de usuario, buscando crear una representación atractiva y coherente de la información. Por otro lado, el diseño funcional se concentra en asegurar el funcionamiento efectivo y sin problemas de todas las funciones y características, con el objetivo de cumplir con los requisitos y necesidades de los usuarios (Coronel, 2012).

Por último, el diseño intuitivo desempeña un papel crucial al facilitar la interacción del usuario con el sistema. Esta dimensión se propone reducir la curva de aprendizaje al crear una interfaz comprensible y fácil de usar, permitiendo que los usuarios realicen tareas de manera natural y sin confusiones. En conjunto, estos tres aspectos del diseño operan en armonía para desarrollar sistemas de software visualmente atractivos, funcionalmente eficientes y de fácil comprensión para los usuarios.

Autoevaluación 7

1. **¿Cuál es el objetivo principal del diseño de interfaz de usuario?**
    a. Crear aplicaciones complejas.
    b. Maximizar la curva de aprendizaje.
    c. Facilitar interacciones efectivas y atractivas entre usuarios y el sistema.
    d. Elegir colores y fuentes llamativos.
2. **¿Qué aspectos se consideran durante el proceso de diseño de interfaz de usuario?**
    a. El costo del proyecto.
    b. La elección de colores y fuentes.
    c. La ubicación geográfica de los usuarios.
    d. La disposición de elementos en la pantalla y la navegación entre pantallas.
3. **¿Por qué es importante la accesibilidad en el diseño de interfaz de usuario?**
    1. Porque solo afecta a un pequeño grupo de usuarios.
    2. Porque garantiza que el sistema sea usable por personas con discapacidades.
    3. Porque no tiene ningún impacto en la usabilidad.
    4. Porque aumenta la curva de aprendizaje.
4. **¿Cómo puede influir positivamente el diseño de interfaz de usuario en la adopción del sistema por parte de los usuarios?**
    1. Haciendo que el sistema sea más complejo.
    2. Aumentando la curva de aprendizaje.
    3. Siendo visualmente atractivo y funcional.
    4. No tiene ningún impacto en la adopción del sistema.

5. **¿Qué herramientas se pueden utilizar durante la etapa de diseño de interfaz de usuario para visualizar la estructura de la pantalla?**
   a. Diagramas de flujo.
   b. Calculadoras.
   c. Wireframes y mockups.
   d. Hojas de cálculo.
6. **¿Cuál es el propósito principal del diseño de la interfaz de usuario?**
   1. Optimizar la estética.
   2. Simplificar la comunicación técnica.
   3. Mejorar la usabilidad y la experiencia del usuario.
   4. Centrarse en los objetivos del sistema.
7. **¿Qué disciplinas se involucran en el diseño de la interfaz de usuario?**
   a. Solo diseño gráfico.
   b. Diseño industrial y ergonomía.
   c. Diseño web y desarrollo de software.
   d. Todas las anteriores.
8. **¿Qué se busca equilibrar en el diseño de la interfaz de usuario?**
   a. La usabilidad y la estética.
   b. La programación y la tipografía.
   c. La velocidad y la seguridad.
   d. La conectividad y la funcionalidad.
9. **¿Cuál es el objetivo final del diseño de la interfaz de usuario?**
   1. Mantener una operación técnica óptima.
   2. Priorizar los objetivos del sistema.
   3. Lograr una comunicación efectiva y eficiente.
   4. Ignorar las necesidades cambiantes de los usuarios.

10. ¿En qué tipos de sistemas se aplica el diseño de la interfaz de usuario?

a. Únicamente en sistemas informáticos.
b. Exclusivamente en proyectos de diseño industrial.
c. En una variedad de sistemas, incluyendo computadoras, dispositivos móviles, software y sitios web.
d. Solo en proyectos relacionados con aeronaves comerciales.

11. ¿Cuáles son los tres elementos esenciales del diseño en el desarrollo de software mencionados en el texto?

1. Visual, interactivo y técnico.
2. Visual, funcional e intuitivo.
3. Estético, técnico y eficiente.
4. Atractivo, rápido y completo.

12. ¿En qué se enfoca principalmente el diseño visual en el desarrollo de software?

a. En garantizar que todas las funciones funcionen sin problemas.
b. En crear una interfaz fácil de usar.
c. En la estética y la presentación gráfica de la interfaz de usuario.
d. En reducir la curva de aprendizaje.

13. ¿Cuál es el objetivo principal del diseño funcional en el desarrollo de software?

a. Crear una interfaz atractiva.
b. Cumplir con los requisitos y necesidades de los usuarios.
c. Reducir la curva de aprendizaje.
d. Centrarse en la presentación gráfica.

14. ¿Qué papel desempeña el diseño intuitivo en la interacción del usuario con el sistema?

1. Aumenta la curva de aprendizaje.
2. Complica la realización de tareas por parte de los usuarios.
3. Facilita la interacción del usuario al reducir la curva de aprendizaje.
4. No tiene impacto en la interacción del usuario.

15. ¿Cómo trabajan en conjunto estos tres aspectos del diseño en el desarrollo de software?

1. Operan en discordia.
2. Operan de manera independiente.
3. Operan en armonía para desarrollar sistemas visualmente atractivos, funcionalmente eficientes y de fácil comprensión para los usuarios.
4. Solo el diseño visual y el diseño funcional trabajan juntos.

2.4 Los modelos más destacados de user interfaz

Implica un medio que simplifica el manejo de un software o hardware específico por parte de un usuario. En este sentido, la Interfaz de Usuario (UI) permite a un cliente llevar a cabo acciones e interactuar con las múltiples opciones proporcionadas por un dispositivo electrónico. Una interfaz de usuario efectiva se destaca por su simplicidad, facilidad de comprensión y alto nivel de usabilidad (Martins, 2023).

Además, también abarca las conexiones entre los usuarios y las aplicaciones web, creando un punto central de enfoque para las compañías que buscan mejorar globalmente la experiencia del usuario.

Según Martins (2023), dice que, para lograrlo, hace uso de una variedad de elementos y controles visuales que enriquecen la interacción de los clientes.

2.4.1 Interfaz de línea de comandos (CLI)

Es una de las interfaces más longevas que continúa en uso en la actualidad. Se basa en texto como medio para interactuar con la computadora, permitiendo la ejecución y administración de programas o archivos. Un ejemplo de esta interfaz de usuario es el sistema operativo MS-DOS, además del Shell de comandos que integra el sistema operativo Windows.

2.4.2 Interfaz de usuario de texto (TUI)

La Interfaz de Usuario de Texto, conocida como Text User Interface en inglés, es una forma de interfaz que se crea mediante el uso de caracteres. Para lograr esto, el sistema se enlaza con un dispositivo, como un teclado, y se concentra en establecer conexiones con el hardware de un dispositivo. Esta interfaz se emplea comúnmente en la configuración de sistemas operativos junto con sus terminales para acceder a diversos programas.

2.4.3 Interfaz gráfica de usuario (GUI)

Denominadas también como Interfaces Gráficas de Usuario (GUI, por sus siglas en inglés), estas representan el formato de interfaz más ampliamente empleado en la contemporaneidad. Este entorno se fundamenta en elementos gráficos y representaciones visuales que muestran tanto la información como las alternativas disponibles para la comunicación entre el usuario y el aparato. En efecto, este tipo de interfaz se diseña específicamente para su uso en dispositivos móviles.

2.4.4 Interfaz de usuario de voz (VUI)

La Interfaz de Usuario por Voz identifica y comprende patrones de habla para activar y llevar a cabo diversas funciones en un sistema. Como ejemplo destacado, mencionamos a Google Nest o Alexa, que, al recibir una orden, pueden reconocer lo

que el usuario dice y responder o llevar a cabo una acción. Estas capacidades proporcionan una mayor comodidad y flexibilidad al permitir que el usuario interactúe sin necesidad de contacto físico, especialmente cuando se está desplazando.

2.4.5 Interfaz de usuario natural (NUI)

Es una categoría de interfaz que proporciona una forma natural e intuitiva de interactuar con el usuario. Su base se encuentra en el uso de gestos y toques en dispositivos táctiles, como se observa en la tecnología Kinect de Xbox. En consecuencia, la Interfaz de Usuario Natural (NUI, por sus siglas en inglés) tiene la capacidad de detectar y comprender las acciones humanas derivadas de movimientos y expresiones faciales.

Autoevaluación 8

1. **¿Qué es una Interfaz de Usuario (UI) en el contexto de la informática?**
    1. Un dispositivo electrónico
    2. Un software específico
    3. Un medio que simplifica la interacción entre un usuario y un dispositivo o software
    4. Un punto central de enfoque para las compañías
2. **¿Cuál es una característica importante de una interfaz de usuario efectiva?**
    1. Complejidad
    2. Facilidad de comprensión
    3. Dificultad de uso
    4. Falta de opciones
3. **¿Qué objetivo busca lograr una interfaz de usuario efectiva?**
    a. Complicar la interacción del cliente
    b. Reducir la usabilidad
    c. Simplificar la interacción del cliente
    d. Aumentar la complejidad del software
4. **¿Qué elementos y controles visuales se utilizan en una interfaz de usuario para enriquecer la interacción del cliente?**
    a. Reducir la interacción
    b. Simplificar la experiencia
    c. Añadir complejidad
    d. Enriquecer la interacción

5. **¿Qué función cumple una interfaz de usuario en las aplicaciones web?**
    1. Simplificar la experiencia del usuario
    2. Complicar la experiencia del usuario
    3. No tiene ninguna función en las aplicaciones web
    4. Facilitar la programación de aplicaciones web
6. **¿Por qué es importante para las compañías mejorar la experiencia del usuario a través de una interfaz de usuario efectiva?**
    a. Porque aumenta la complejidad del software
    b. Porque reduce la usabilidad
    c. Porque atrae a más usuarios y mejora la satisfacción del cliente
    d. Porque complica la interacción del cliente
7. **¿Cuál es una característica destacada de la Interfaz de Línea de Comandos (CLI)?**
    1. Utiliza elementos visuales e imágenes.
    2. Se basa en texto para interactuar con la computadora.
    3. Permite la interacción mediante gestos.
    4. Se utiliza principalmente en dispositivos móviles.
8. **¿Cuál es un ejemplo de sistema operativo que utiliza una Interfaz de Línea de Comandos (CLI)?**
    a. MacOS
    b. Linux
    c. Windows
    d. Android

9. ¿Cuál es un ejemplo de un Shell de comandos que integra el sistema operativo Windows?

1. Terminal
2. PowerShell
3. Command Prompt
4. Terminal Emulator

10. ¿Qué significa la sigla TUI en inglés?

a. Texto Unificado Internacional
b. Texto Universal Integrado
c. Text User Interface
d. Tecnología de Usuarios Interactivos

11. ¿Cómo se crea una Interfaz de Usuario de Texto (TUI)?

1. Utilizando imágenes y elementos visuales.
2. Mediante el uso de caracteres.
3. A través de gestos y movimientos.
4. Usando comandos de voz.

12. ¿En qué se concentra una TUI en términos de hardware?

1. En conectar dispositivos móviles.
2. En establecer conexiones con el hardware de un dispositivo.
3. En mejorar la calidad de imagen.
4. En aumentar la velocidad de procesamiento.

13. ¿Dónde se utiliza comúnmente la TUI?

a. En dispositivos móviles.
b. En la configuración de sistemas operativos y terminales.
c. En la navegación web.
d. En la edición de videojuegos.

14. ¿Qué significa la sigla GUI en inglés?

a. General User Interface
b. Graphics User Interface
c. Gesture User Interface
d. Good User Interaction

15. ¿En qué se basa la Interfaz Gráfica de Usuario (GUI) para la interacción entre el usuario y el dispositivo?

1. En elementos auditivos.
2. En elementos visuales e imágenes.
3. En comandos de voz.
4. En gestos y toques.

16. ¿Qué tipo de acciones ejecuta una Voice User Interface (VUI) mediante el reconocimiento de patrones vocales?

1. Acciones visuales.
2. Acciones gestuales.
3. Acciones de texto.
4. Acciones de voz.

17. ¿Cuál es un ejemplo de dispositivo que utiliza una VUI según el texto?

a. iPhone
b. Xbox Kinect
c. Sistema operativo Windows
d. Dispositivos móviles

18. ¿Cuál es la base de la Interfaz de Usuario Natural (NUI) para la interacción con el usuario?

1. Uso de comandos de voz.
2. Uso de gestos y toques en dispositivos táctiles.
3. Uso de comandos de texto.
4. Uso de elementos visuales.

19. ¿Dónde se observa comúnmente el uso de gestos y toques en dispositivos táctiles como parte de una NUI?

a. En la tecnología Kinect de Xbox.
b. En la navegación web.
c. En sistemas operativos de computadoras.
d. En reproductores de música.

20. ¿Qué capacidad tiene una Interfaz de Usuario Natural (NUI) según el texto?

1. Detectar y comprender acciones de comandos de voz.
2. Detectar y comprender acciones humanas derivadas de movimientos y expresiones faciales.
3. Mostrar información visual.
4. Reproducir música de forma natural.

2.5 Conclusión

En conclusión, podemos decir que, en este capítulo, hemos explorado conceptos fundamentales del diseño en el desarrollo de software, discutiendo características, principios y estándares que guían la creación de soluciones efectivas y coherentes. Hemos analizado la importancia del diseño de la arquitectura de software, considerando enfoques estructurados y orientados a objetos para la construcción de sistemas sólidos y adaptables.

Destacamos la vitalidad de la Interfaz de Usuario (UI) en el diseño de software, reconociendo su rol crucial en mejorar la experiencia del usuario. Exploramos diversos tipos de interfaces, desde la tradicional Interfaz de Línea de Comandos (CLI) hasta la moderna Interfaz de Usuario de Texto (TUI) y la innovadora

Interfaz de Usuario Natural (NUI), cada una con un enfoque distintivo para interactuar con los sistemas.

En síntesis, abarcamos una amplia gama de temas en el diseño de software, desde fundamentos hasta modelos de interfaces destacados. Un diseño efectivo no solo influye en la apariencia visual, sino también en la funcionalidad, usabilidad y satisfacción del usuario en última instancia.

Referencias

Cáceres, E. (2014). *Análisis y Diseño de Sistemas de Información*. https://acortar.link/X2axIu

Coronel, C., Morris, S., y Rob, P. (2012). *Base de datos, diseño, implementación y administración*. CENGAGE Learning.

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

León Yacelga, A.R., Acosta Espinoza, J.L., & Díaz Vásquez, R.A. (2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Universidad Y Sociedad, 13*(5), 175-182. https://rus.ucf.edu.cu/index.php/rus/article/view/2223

Martínez Canelo, M. (2020, 24 de junio). ¿Qué son los Patrones de Diseño de software? *Design Patterns. Profile Software Services*. https://acortar.link/SDeRzu

Paredes Hernández, E., & Velasco Espitia, M. (1998). *Análisis y Diseño de Sistemas de Información.* Universidad de Pamplona

§

Design Concepts
Conceitos de design

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 explore the "Design Concepts", analyzing relevant details such as the "Design Characteristics and Principles" that establish the basis for efficient and flexible systems, and the "Design Standards" that ensure consistency and quality. In addition, "Software Architecture Design" will be discussed in depth, considering structured and object-oriented approaches for adaptive systems. User Interface Design" will also be addressed, highlighting its importance in the user experience and the creation of visual and intuitive interfaces. Finally, "User Interface Models" will be explored, including command line, text, graphical and voice interfaces. In summary, this chapter will be a complete immersion into the fundamentals of design in systems analysis and design, establishing a solid foundation for effective solutions.
Keywords: design principles, user interface, software architecture.

Resumo

Neste capítulo, exploraremos os "Conceitos de design", analisando detalhes relevantes, como os "Recursos e princípios de design", que estabelecem a base para sistemas eficientes e flexíveis, e os "Padrões de design", que garantem a coerência e a qualidade. Além disso, o "Design de arquitetura de software" será discutido em profundidade, considerando abordagens estruturadas e orientadas a objetos para sistemas adaptativos. O "Design da interface do usuário" também será abordado, destacando sua importância para a experiência do usuário e a criação de interfaces visuais e intuitivas. Por fim, serão explorados os "Modelos de interface do usuário", incluindo interfaces de linha de comando, de texto, gráficas e de voz. Em resumo, este capítulo será uma imersão completa nos fundamentos do design na análise e no design de sistemas, estabelecendo uma base sólida para soluções eficazes.
Palavras-chave: princípios de design, interface do usuário, arquitetura de software.

Capítulo 3

Metodologías y procesos de desarrollo de software

Mónica Elizabeth Páez Padilla

Resumen

Este capítulo se enfoca en la relevancia esencial de las metodologías y procesos de desarrollo de software para lograr programas confiables y de alta calidad. Su objetivo central es explorar en detalle una gama de metodologías y paradigmas que guían este proceso, delineando sus características fundamentales y su clasificación en categorías como estructuradas, orientadas a objetos y ágiles. Además, el capítulo explora los ciclos de vida del desarrollo de software, los paradigmas que orientan la planificación y ejecución de proyectos, y clasifica los paradigmas de proceso comunes. Finalmente, se analiza la importancia del proceso de desarrollo y la función de los estándares en asegurar coherencia y calidad en los proyectos de software. En resumen, el capítulo proporciona una comprensión sólida de las metodologías, paradigmas y procesos que moldean la creación de software en la industria de la ingeniería de software.
Palabras claves: desarrollo de software, proyectos, planificación.

Páez Padilla, M.E. (2023). Metodologías y procesos de desarrollo de software. En M.E. Páez Padilla (ed). *Análisis y diseño de sistemas.* (pp. 123-167). Religación Press. http://doi.org/10.46652/religacionpress.89.c84

3.1 Metodologías de Desarrollo de Software

3.1.1 Definiciones metodología de desarrollo de software.

Las metodologías de desarrollo de software abarcan un conjunto de enfoques y procedimientos de organización de empleados para concebir soluciones de software. El propósito de estas diversas metodologías radica en estructurar equipos de trabajo de manera eficiente para lograr una ejecución óptima de las funcionalidades de un programa.

Cuando se busca crear productos o respuestas para un cliente o mercado específico, es esencial considerar aspectos como los gastos, la programación, la complejidad, el equipo disponible, los idiomas empleados, entre otros. Estos elementos se integran en un enfoque de desarrollo que posibilita estructurar la mano de obra de manera altamente organizada (Santander, 2023).

3.1.2 Características metodología de desarrollo de software.

- Un proceso integral de ciclo de vida, abarcando tanto aspectos empresariales como técnicos
- Un conjunto exhaustivo de conceptos y modelos que mantengan una coherencia interna

- Un compendio de reglas y directrices
- Una descripción total de los elementos a desarrollar
- Una notación utilizada para trabajar, preferentemente respaldada por diversas herramientas CASE y diseñada para una experiencia óptima del usuario
- Un repertorio de técnicas con validación
- Un conjunto de métricas, junto con asesoramiento sobre calidad, estándares y enfoques de prueba
- Identificación de los roles dentro de la organización
- Orientación para la gestión de proyectos y aseguramiento de la calidad
- Asesoramiento sobre la administración de bibliotecas y la reutilización

3.1.3 Clasificación de las metodologías de desarrollo: estructurados, orientadas a objetos, ágiles.

Las estrategias empleadas en la creación de software se dividen en diversas corrientes principales, que incluyen los métodos estructurados, los orientados a objetos y las metodologías ágiles.

Según Maida y Pacienzia (2015) dicen que a continuación, se ofrece una descripción de cada una de estas categorías:

Metodologías Estructuradas:

Las metodologías estructuradas se fundamentan en un enfoque secuencial y jerárquico para crear software. Su enfoque se centra en dividir un proyecto en etapas bien definidas y lineales, destacando la importancia del análisis y diseño antes de la implementación. Ejemplos de estas metodologías son el modelo en cascada y el modelo en V.

Metodologías Orientadas a Objetos:

Las metodologías orientadas a objetos se centran en el diseño y desarrollo de software basado en objetos, los cuales son unidades autónomas que encapsulan datos y funcionalidades. Estas metodologías promueven la reutilización, modularidad y flexibilidad. Ejemplos de estas metodologías incluyen el Lenguaje de Modelado Unificado (UML) y los Métodos de Desarrollo de Software Orientado a Objetos (Object-Oriented Software Development Methods, OOSD).

Metodologías Ágiles:

Las metodologías ágiles son enfoques flexibles y colaborativos que priorizan la adaptación y la entrega continua de software funcional. Se basan en la interacción frecuente con los clientes, la comunicación entre los miembros del equipo y la capacidad de responder de manera ágil a los cambios. Ejemplos de estas me-

todologías son Scrum, Kanban, Extreme Programming (XP) y Lean. Cada una de estas categorías posee sus propias ventajas y desventajas. La elección de una metodología adecuada dependerá de factores como la naturaleza del proyecto, las preferencias del equipo, los plazos y la complejidad del software a desarrollar. Elegir la metodología correcta es crucial para maximizar la eficiencia y la calidad en el proceso de desarrollo.

Autoevalución 9

1. **¿Cuál es el propósito principal de las metodologías de desarrollo de software mencionadas en el texto?**
    1. Crear soluciones de software eficientes.
    2. Organizar equipos de trabajo de manera óptima.
    3. Maximizar los gastos en desarrollo de software.
    4. Utilizar múltiples idiomas en el desarrollo de software.
2. **¿Qué elementos se deben considerar al buscar crear productos o soluciones de software según el texto?**
    a. El clima.
    b. La historia de la empresa.
    c. Los gastos, la programación, la complejidad y otros factores.
    d. El costo de los servidores.
3. **¿Qué papel desempeñan las metodologías de desarrollo de software en la organización de equipos de trabajo?**
    a. Reducen la eficiencia de los equipos.
    b. Aumentan la complejidad del proceso de desarrollo.
    c. Estructuran la mano de obra de manera altamente organizada.

d. No tienen impacto en la organización de equipos.

4. **¿Cuál es uno de los aspectos clave mencionados en el texto que se deben considerar al elegir una metodología de desarrollo de software?**
    1. El idioma del cliente.
    2. La ubicación geográfica del equipo de desarrollo.
    3. La complejidad del proyecto.
    4. El nombre de la empresa.
5. **¿Cuál es la finalidad principal de las metodologías de desarrollo de software en relación con las funcionalidades de un programa?**
    1. Aumentar la complejidad de las funcionalidades.
    2. Lograr una ejecución óptima de las funcionalidades.
    3. Maximizar los costos de desarrollo.
    4. Ignorar las funcionalidades del programa.
6. **¿Cuál es el enfoque principal de las metodologías estructuradas?**
    1. Enfoque secuencial y jerárquico.
    2. Enfoque flexible y colaborativo.
    3. Enfoque centrado en objetos.
    4. Enfoque centrado en la adaptación continua.
7. **¿Qué importancia tienen el análisis y diseño en las metodologías estructuradas?**
    a. Mínima importancia.
    b. Importancia moderada.
    c. Importancia alta.
    d. No se menciona en el texto.
8. **¿Qué modelos son ejemplos de metodologías estructuradas?**
    1. Modelo en cascada y modelo en V.

2. Scrum y Kanban.
3. UML y OOSD.
4. Agile y Lean.

9. ¿En qué se centran las metodologías orientadas a objetos?

a. En el diseño secuencial.
b. En la adaptación continua.
c. En el desarrollo basado en objetos.
d. En la comunicación con el cliente.

10. ¿Qué promueven las metodologías orientadas a objetos?

1. La reutilización, modularidad y flexibilidad.
2. La entrega continua de software funcional.
3. La adaptación a cambios sin restricciones.
4. La comunicación entre los miembros del equipo.

11. ¿Cuál es un ejemplo de lenguaje de modelado orientado a objetos mencionado en el texto?

a. Scrum.
b. UML.
c. Kanban.
d. Lean.

12. ¿Qué priorizan las metodologías ágiles?

1. La entrega única de software.
2. La comunicación con el cliente.
3. La adaptación continua y la entrega frecuente de software funcional.
4. La planificación a largo plazo.

13. ¿En qué se basa la interacción frecuente en las metodologías ágiles?

1. En la comunicación escrita.
2. En la comunicación con otros equipos.

3. En la comunicación con el cliente.
4. No se basa en la interacción frecuente.

14. ¿Cuál de las siguientes no es una metodología ágil mencionada en el texto?

a. Scrum.
b. Kanban.
c. UML.
d. Extreme Programming (XP).

15. ¿Qué factores influyen en la elección de una metodología adecuada?

a. La preferencia del equipo.
b. El clima.
c. La ubicación geográfica.
d. Todas las anteriores.

3.2 Definiciones metodología de desarrollo de software

3.2.1 Ciclos de vida de Desarrollo de Software.

Concepto

El proceso de desarrollo de software a lo largo de su ciclo de vida (también reconocido como SDLC o Ciclo de Vida del Desarrollo de Sistemas) involucra las etapas necesarias para validar la creación del software, asegurando que este se ajuste a los requisitos establecidos y permitiendo la verificación de los procedimientos de desarrollo. Esto garantiza que los métodos utilizados sean apropiados (Bournissen, 1999).

La génesis de este enfoque surge del hecho de que corregir errores que se detectan en etapas avanzadas, durante la implementación, resulta sumamente costoso. Mediante el uso de enfoques metodológicos adecuados, es posible identificar problemas a tiempo, permitiendo que los programadores se concentren en la calidad del software y cumplan con los plazos y los costos preestablecidos (Bournissen, 1999).

Fases de desarrollo de software

La metodología de desarrollo de software se constituye como un enfoque estructurado para la ejecución, gestión y supervisión de un proyecto, con el fin de aumentar significativamente las probabilidades de éxito. Esta sistematización establece cómo dividir

un proyecto en componentes más pequeños para estandarizar su gestión.

En este contexto, una metodología de desarrollo de software engloba los procesos que deben ser seguidos de manera ordenada para idear, construir y mantener un producto de software, desde el instante en que se identifica la demanda del producto hasta que se logra el objetivo para el cual fue concebido.

De esta manera, las etapas del proceso de desarrollo de software son las siguientes:

Planificación

Antes de comenzar un proyecto de desarrollo de un sistema de información, es esencial realizar una serie de actividades que desempeñarán un papel fundamental en su éxito. Estas actividades son comúnmente referidas como la "fase inicial difusa" del proyecto, ya que no están restringidas por plazos específicos.

Algunas de las actividades en esta etapa engloban acciones como definir el alcance del proyecto, llevar a cabo un análisis de viabilidad, evaluar los riesgos asociados, estimar los costos involucrados en el proyecto, planificar su cronograma y distribuir los recursos para las diferentes fases del proyecto.

Análisis

Indudablemente, resulta necesario investigar con precisión

cuál es la función que el software debe desempeñar. En consecuencia, la etapa de análisis dentro del ciclo de vida del software representa el proceso mediante el cual se busca con precisión determinar lo que se necesita y obtener una comprensión total de los requisitos del sistema (las características específicas que el sistema debe tener).

Diseño

En esta fase, se investigan diversas posibilidades de cómo implementar el software que se va a crear y se selecciona la estructura básica del mismo. El proceso de diseño es complejo y su progreso debe basarse en un enfoque que involucre iteraciones.

Es viable que la solución inicial no resulte óptima, en cuyo caso es necesario pulirla. Sin embargo, existen catálogos de modelos de diseño que resultan extremadamente beneficiosos al captar los errores cometidos por otros, evitando así caer en los mismos obstáculos.

Implementación

En esta etapa, resulta fundamental elegir las herramientas apropiadas, configurar un ambiente de desarrollo que simplifique las tareas y decidir sobre un lenguaje de programación adecuado para el tipo de software que se va a desarrollar. Esta elección estará influenciada tanto por las decisiones de diseño adoptadas como por el entorno en el que el software estará funcionando.

Al codificar, es esencial evitar que el código se vuelva incomprensible, y esto se logra al seguir directrices como las siguientes:

- Evitar la creación de bloques de control que carezcan de estructura.
- Llevar a cabo una precisa identificación de las variables y definir su extensión.
- Elegir algoritmos y estructuras de datos apropiados para abordar el problema específico.
- Mantener la simplicidad en la lógica de la aplicación.
- Incluir una documentación y comentarios adecuados en el código de los programas.
- Enriquecer la legibilidad del código a través de la aplicación de reglas de formato previamente acordadas en el equipo de desarrollo.

También resulta crucial contemplar la adquisición de los recursos requeridos para el correcto desempeño del software, y a medida que se programa, crear casos de prueba que posibiliten la evaluación de su operatividad.

Pruebas

Dado que todos somos susceptibles a cometer errores, la etapa de pruebas en el ciclo de vida del software tiene como objetivo identificar los errores que puedan haber surgido en las fases pre-

vias para subsanarlos. Claro está, el objetivo óptimo es realizar esta corrección antes de que los usuarios finales lleguen a notarlos. Se considera que una prueba ha sido exitosa si logra identificar algún tipo de error.

Instalación o despliegue

El paso siguiente implica poner en marcha el software, por lo cual es esencial llevar a cabo una planificación del entorno considerando las interrelaciones existentes entre los distintos elementos que lo componen.

Es factible que algunos componentes funcionen sin inconvenientes de manera individual, pero al unirse, puedan generar dificultades. Por esta razón, es fundamental recurrir a combinaciones previamente probadas que no ocasionen conflictos de compatibilidad.

Uso y mantenimiento

Este constituye uno de los momentos de mayor relevancia en el ciclo de vida de desarrollo de software. Dado que el software no experimenta desgaste ni deterioro en su uso, su mantenimiento abarca tres aspectos distintos:

- Rectificación de los fallos identificados durante su período de utilización (mantenimiento correctivo).
- Ajuste para satisfacer nuevas necesidades emergentes

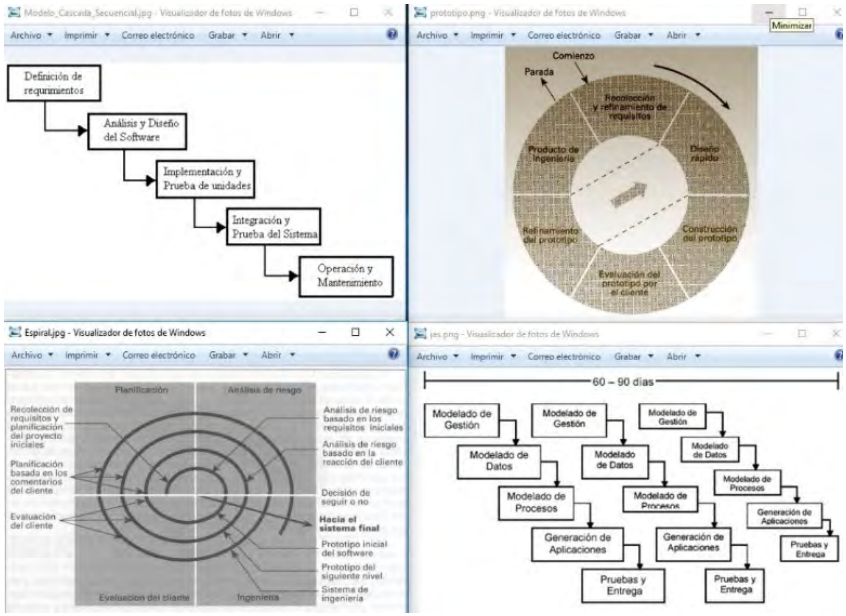
(mantenimiento adaptativo).

- Introducción de nuevas capacidades y funciones (mantenimiento perfectivo).

A pesar de que pueda parecer paradójico, cuanto mayor es la calidad del software, mayor inversión temporal debe destinarse a su mantenimiento. Esto se debe en gran medida al incremento en su uso (inclusive de maneras que no se habían anticipado), lo cual conlleva más propuestas de mejora y optimización (Cedeño, 2015).

3.2.2 Definición de paradigmas de proceso

Figura 7. Paradigmas en el desarrollo de software



Nota. El grafico 7 representa los paradigmas en el desarrollo de software

Dentro del campo de la Ingeniería de Software, un paradigma se define como un conjunto de técnicas, herramientas y procedimientos que se combinan con la finalidad de establecer un modelo.

En el ámbito de la Ingeniería de Software, cada metodología de desarrollo de software ofrece su propio enfoque único para el proceso de construcción del software. La Ingeniería de Software establece paradigmas de desarrollo estructurado como un punto de partida en un proyecto de software (Maida y Pacienzia, 2015).

Cuando ninguna de estas metodologías se adecua a la problemática en cuestión, los desarrolladores pueden verse en la necesidad de combinar paradigmas existentes o incluso crear uno nuevo.

Para abordar desafíos prácticos, el profesional de la ingeniería de software debe incorporar una estrategia que guíe el proceso, los métodos y el conjunto de herramientas utilizadas. Esta estrategia es comúnmente conocida como un modelo de proceso o paradigma en el campo de la ingeniería de software.

La elección de un modelo de proceso para la ingeniería de software depende de la naturaleza del proyecto y de la aplicación, así como de los métodos, herramientas, controles y entregables necesarios. Los modelos o paradigmas más comúnmente empleados en el desarrollo de software incluyen el enfoque en cascada, el enfoque de prototipos y el enfoque en espiral (Maida y Pacienzia, 2015).

Autoevaluación 10

1. **¿Qué significa el acrónimo SDLC en el contexto del desarrollo de software?**
   a. Sistema de Desarrollo de Lenguaje de Código.
   b. Software Development Life Cycle (Ciclo de Vida del Desarrollo de Software).
   c. Sistema de Diseño y Lenguaje de Código.
   d. Seguridad de Desarrollo de Lenguaje de Código.
2. **¿Por qué es importante seguir un enfoque metodológico adecuado en el desarrollo de software?**
   1. Para reducir el tamaño del proyecto.
   2. Para aumentar la complejidad del software.
   3. Para estandarizar la gestión y aumentar las probabilidades de éxito.
   4. Para acelerar el desarrollo.
3. **¿Qué actividades se llevan a cabo en la fase de Planificación del ciclo de vida del software?**
   1. Codificación y pruebas.
   2. Definición del alcance, análisis de viabilidad y evaluación de riesgos.
   3. Diseño de la interfaz de usuario.
   4. Instalación y despliegue.
4. **¿Cuál es el propósito de la fase de Análisis en el desarrollo de software?**
   a. Seleccionar herramientas de desarrollo.
   b. Determinar los requisitos del sistema.
   c. Diseñar la estructura del software.
   d. Codificar el programa.

5. **¿Qué se busca lograr en la fase de Diseño del ciclo de vida del software?**
    1. Determinar los requisitos del sistema.
    2. Seleccionar herramientas de desarrollo.
    3. Elegir la estructura básica del software.
    4. Identificar errores en el código.
6. **¿Qué se debe evitar al codificar el software durante la fase de Implementación?**
    1. Documentación y comentarios adecuados.
    2. Identificación precisa de variables y definición de su extensión.
    3. Creación de bloques de control con estructura.
    4. Simplificación de la lógica de la aplicación.
7. **¿Cuál es el objetivo de la fase de Pruebas en el ciclo de vida del software?**
    a. Identificar errores y corregirlos antes de que los usuarios los noten.
    b. Documentar el código fuente.
    c. Seleccionar el lenguaje de programación.
    d. Implementar el software.
8. **¿Qué aspecto se considera exitoso en una prueba de software?**
    1. Identificar errores.
    2. No encontrar ningún error.
    3. Encontrar errores que los usuarios también encuentren.
    4. Documentar el código fuente.

9. ¿Qué implica la fase de Instalación o despliegue en el ciclo de vida del software?

1. Poner en marcha el software y considerar interrelaciones entre componentes.
2. Identificar errores en el código fuente.
3. Seleccionar herramientas de desarrollo.
4. Codificar el software.

10. ¿Por qué el mantenimiento es una fase crucial en el ciclo de vida del desarrollo de software?

1. Porque el software se desgasta con el tiempo.
2. Porque el mantenimiento es más costoso que el desarrollo inicial.
3. Porque el software no experimenta desgaste y puede requerir correcciones, ajustes y mejoras.
4. Porque el mantenimiento no es necesario en el desarrollo de software de alta calidad.

11. ¿Cómo se define un paradigma en el contexto de la Ingeniería de Software?

1. Como un modelo de proceso.
2. Como un conjunto de técnicas de programación.
3. Como un enfoque único para el desarrollo de software.
4. Como un conjunto de herramientas de desarrollo.

12. ¿Por qué los desarrolladores de software pueden necesitar combinar paradigmas existentes o crear uno nuevo?

a. Para aumentar la complejidad del proyecto.
b. Cuando ninguna metodología existente se ajusta a la problemática en cuestión.
c. Para reducir los costos del desarrollo.
d. Para seguir las tendencias actuales en desarrollo de software.

13. ¿Qué se entiende por "modelo de proceso" o "paradigma" en la Ingeniería de Software?

1. Un conjunto de herramientas de desarrollo.
2. Una estrategia que guía el proceso, los métodos y las herramientas utilizadas.
3. Un lenguaje de programación.
4. Un conjunto de técnicas de programación.

14. ¿Cuál es un factor importante a considerar al elegir un modelo de proceso en la Ingeniería de Software?

1. La complejidad del código fuente.
2. La naturaleza del proyecto y de la aplicación, así como los métodos y herramientas necesarios.
3. La preferencia personal del desarrollador.
4. El tamaño del equipo de desarrollo.

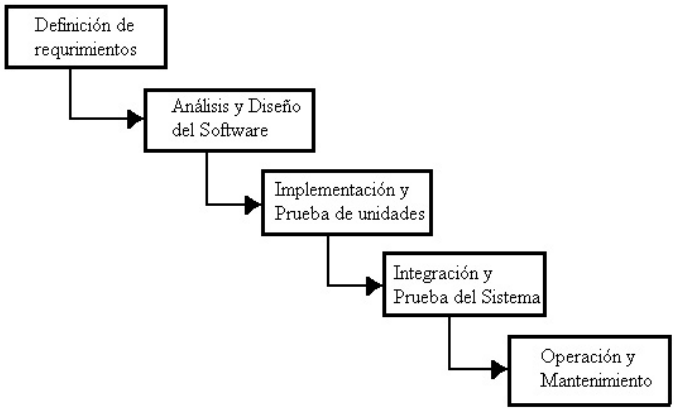
15. ¿Cuáles son algunos de los modelos o paradigmas más comúnmente empleados en el desarrollo de software mencionados en el texto?

a. Enfoque en criptografía, enfoque de redes neuronales, enfoque de inteligencia artificial.
b. Enfoque en cascada, enfoque de prototipos, enfoque en espiral.
c. Enfoque de diseño gráfico, enfoque de marketing, enfoque de ventas.
d. Enfoque de hardware, enfoque de software, enfoque de seguridad.

3.3 Clasificación de los paradigmas de proceso:

3.3.1 Modelo en cascada

Figura 8.Modelo Lineal Secuencial o de Cascada (Waterfall)



Nota. La figura 8 hace mención al modelo tradicional o en cascada, mostrando cada una de las etapas se desarrollan de manera consecutiva y secuencial.

Es un procedimiento de desarrollo secuencial en el que los pasos de progreso descienden de manera continua, siguiendo una secuencia similar al flujo de una cascada de agua. Esta secuencia abarca fases que van desde el análisis de requerimientos, el diseño, la implementación, las pruebas (validación), la integración y el mantenimiento.

Comúnmente, se atribuye la primera descripción formal del modelo en cascada a un artículo escrito por Winston Royce en 1970, aunque en dicho artículo Royce no empleó el término "cascada".

Este modelo representa el paradigma más antiguo y predominante durante la era del método estructurado. El número de fases propuestas puede variar según el proyecto en desarrollo, pero en este enfoque existen etapas comunes.

Dentro de este enfoque, las fases de desarrollo se consideran como procesos que ocurren en diferentes momentos en el tiempo, lo que implica que no pueden llevarse a cabo simultáneamente. Cada fase comienza después de que se haya completado la fase anterior, y para avanzar a la siguiente etapa, es necesario cumplir con todos los objetivos de la fase anterior.

Las etapas en este paradigma siguen una secuencia lineal. Cuando se detecta un error en alguna fase, generalmente es necesario retroceder hasta la fase inicial de análisis de requisitos del sistema. Aunque es posible retroceder a través de las etapas, esto conlleva un esfuerzo significativo y puede resultar en el fracaso del proyecto (Sulbarán, 2018).

Definición de los requisitos

Dentro de este procedimiento, se reconocen las demandas y condiciones del cliente en relación al software.

Según Sommerville (2011), nos señala las siguientes definiciones:

Análisis y Diseño: Durante la etapa de análisis, se evalúa la posibilidad del software tanto en términos técnicos como económicos, además se trazan los pasos y el financiamiento previsto. En la fase de diseño del software, el enfoque se centra en cuatro características clave de un programa: la organización de los datos, la arquitectura del software, las representaciones de la interfaz y los procedimientos detallados (algoritmos).

Codificación: El modelo diseñado previamente es transformado en un formato que la máquina pueda comprender. Esta tarea es realizada por el proceso de generación de código. Cuando el diseño se elabora de manera exhaustiva, la generación de código se ejecuta de manera automática.

Pruebas: El proceso de pruebas se focaliza tanto en los aspectos internos lógicos del software como en sus funciones externas. Estas pruebas tienen como objetivo identificar fallos y verificar que las entradas predefinidas generen resultados concretos que se alineen con los resultados esperados.

Mantenimiento: Es innegable que el software experimentará modificaciones después de su entrega al cliente. El proceso de mantenimiento implica volver a implementar todas las fases anteriores en un programa ya existente en lugar de uno nuevo.

3.3.2 Desarrollo incremental

Figura 9. Metodología de Desarrollo Incremental.

Nota. Con relación a la metodología de desarrollo incremental, es posible observar las etapas de su proceso de desarrollo en la Figura 9.

La metodología incremental inicia con un diseño inicial que abarca aspectos fundamentales, denominado esquema de descripción. A medida que avanza el proceso de desarrollo, se elaboran versiones progresivamente más avanzadas del sistema, culminando en una versión definitiva que cumple plenamente con las necesidades del usuario y satisface todos los requisitos de manejo y gestión de la información.

Tal como afirma León et al. (2021), que el enfoque incremental emplea secuencias de pasos lineales progresivos, a medida que

avanza en el calendario. Entre las actividades simultáneas se hace referencia a la especificación, en la que se establecen los requisitos y se recopila la información específica para ese incremento, junto con su análisis y diseño correspondiente. Posteriormente, se procede a la etapa de desarrollo en el lenguaje de programación elegido, y al final se verifica la congruencia de los resultados obtenidos con los requisitos iniciales del incremento. A medida que avanza, se van entregando versiones intermedias del sistema. Estas secuencias iterativas se repetirán hasta alcanzar un producto que satisfaga las exigencias del cliente.

Es relevante destacar que el enfoque de desarrollo incremental sirve de base para la mayoría de las metodologías ágiles de desarrollo de software. Es de importancia que los estudiantes que se encuentran en proceso de aprender el desarrollo de sistemas empleen esta metodología antes de recurrir directamente a enfoques ágiles, como una experiencia previa. Uno de los beneficios notables de la adopción de esta metodología es que los costos asociados con cambios en los requisitos, que son más significativos en metodologías tradicionales y rígidas como el enfoque en cascada, resultan mucho menores y se pueden incorporar en cada una de las etapas iterativas del sistema.

Comparado con la metodología en cascada, una ventaja evidente es la mayor facilidad para obtener retroalimentación del cliente. Esto se debe a que, durante la implementación de los incrementos, el usuario tiene la oportunidad de interactuar con el sistema en las versiones parciales y ofrecer comentarios y sugerencias antes de avanzar con los siguientes incrementos. Esto se

traduce en un sistema más elaborado y con un alto nivel de aceptación, tal como lo menciona (Sommerville, 2011).

El progreso gradual en la ampliación de funcionalidades de un sistema, en el contexto de la metodología incremental, implica que el líder del proyecto debe efectuar revisiones después de cada iteración. Esta revisión tiene como fin evaluar si el avance está en línea con el plan establecido o si requiere ajustes para cumplir con la meta principal del proyecto. Además, esta evaluación permite verificar si el cronograma planificado se ajusta a los avances logrados en cada incremento y si el proyecto general sigue su curso normalmente.

No obstante, surgen ciertos desafíos al emplear la metodología incremental. Uno de ellos es la tarea compleja de identificar los requisitos comunes entre los distintos incrementos que se llevarán a cabo durante el desarrollo del sistema. Esto puede llevar a duplicar esfuerzos o definir de manera superficial algunos de los requerimientos esenciales de la aplicación. Cuando se trata de reemplazar un sistema antiguo por uno nuevo, no se aconseja utilizar la metodología incremental. Esto se debe a que los usuarios podrían intentar aplicar todas las funcionalidades tan pronto como los primeros incrementos se presenten, lo que podría generar resistencia e insatisfacción con la nueva propuesta.

En muchas empresas u organizaciones, al momento de establecer contratos, se requiere especificar de manera anticipada y con un alto nivel de detalle las características y funciones del sistema. Sin embargo, en la metodología incremental, los detalles de las especificaciones se desarrollan a medida que avanzan los

incrementos del software. Por lo tanto, adaptar una nueva forma de contratación se vuelve complicado, especialmente en instituciones gubernamentales (Sommerville, 2011).

Desventajas del desarrollo incremental

El enfoque de desarrollo incremental, como cualquier modelo, presenta ventajas y desventajas, las cuales dependen del proyecto, el contexto, el equipo de trabajo, los acuerdos contractuales y otros factores. Hasta ahora, he resaltado los aspectos positivos del modelo de manera general, pero ninguna metodología es completamente infalible.

Un desafío principal se origina en la existencia de procedimientos burocráticos arraigados en las grandes organizaciones, lo que puede generar descoordinación entre estos procesos y un enfoque iterativo o ágil más flexible.

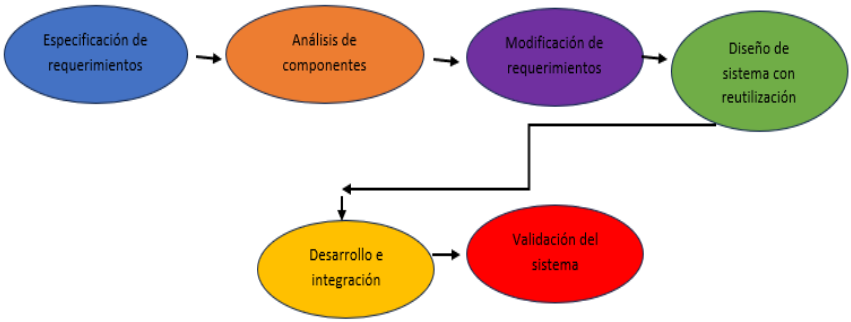
Los inconvenientes del desarrollo incremental se hacen más evidentes en sistemas amplios, complejos y de largo plazo, donde equipos diversos trabajan en distintas partes del sistema. Para sistemas de gran envergadura, es fundamental contar con una estructura arquitectónica sólida y definir con claridad las responsabilidades de los equipos involucrados en las diferentes áreas del sistema. Estas cuestiones deben ser planeadas de antemano en lugar de evolucionar de manera incremental.

A pesar de lo mencionado anteriormente, el concepto de presentar una versión inicial del producto con las funcionalida-

des esenciales y luego mejorarla progresivamente sigue siendo válido en el desarrollo incremental. Sin embargo, esta metodología no siempre es factible, ya que la implementación de nuevo software en un entorno de producción real puede afectar los procesos empresariales normales (Costanzo, 2023).

3.3.3 Ingeniería de software orientado a la reutilización y otros

Figura 10.Ingeniería de software orientada a la reutilización



Nota. Se presenta en la figura 10 indica un esquema que ilustra el modelo típico del proceso de desarrollo basado en reutilización.

En la mayoría de los proyectos de desarrollo de software, se produce cierto grado de reutilización de componentes de software. Este fenómeno ocurre de manera frecuente en un contexto informal, donde los profesionales involucrados en el proyecto identifican diseños o códigos similares a lo que se necesita y pro-

ceden a buscarlos, adaptarlos según sea necesario e integrarlos en sus sistemas.

Este tipo de reutilización no está vinculado a un proceso de desarrollo particular. Sin embargo, en el siglo XXI, los enfoques de desarrollo de software que hacen hincapié en la reutilización de componentes existentes son ampliamente adoptados. Estos enfoques se basan en una extensa colección de componentes de software reutilizables y en la incorporación de marcos de trabajo para ensamblar estos componentes. En algunas ocasiones, estos componentes pueden ser sistemas completos en sí mismos, como software comercial listo para usar (COTS), que mejoran funciones específicas, como procesadores de texto o hojas de cálculo.

Aunque las etapas iniciales de definición de requisitos y validación se asemejan a otros procesos de desarrollo de software en un enfoque orientado a la reutilización, las fases intermedias muestran notables distinciones.

En la investigación de Whitten et al. (2003), estas etapas comprenden:

Análisis de componentes: Una vez se cuenta con la descripción de los requisitos, se procede a buscar componentes que se ajusten a esa descripción para llevar a cabo la implementación. En la mayoría de los casos, no se encuentra una correspondencia exacta, y los componentes empleados solo abarcan una porción de la funcionalidad necesitada.

Modificación de requerimientos: En este punto del proceso, se examinan los requisitos empleando los detalles de los componentes identificados. Después, se ajustan para adecuarse a los componentes que están disponibles. En situaciones donde no es factible llevar a cabo las modificaciones necesarias, existe la posibilidad de volver a la fase de análisis de componentes para explorar en busca de enfoques alternativos.

Diseño de sistema con reutilización: En esta fase, se desarrolla la estructura conceptual del sistema o se hace uso de un marco conceptual existente. Los diseñadores analizan los componentes que están siendo reutilizados y ajustan el marco de trabajo en consecuencia. En casos en los que los componentes reutilizables no estén disponibles, es probable que sea necesario crear nuevo software para llenar esas brechas.

Desarrollo e integración: Luego, se procede con la creación del software que no puede ser obtenido de fuentes externas y se realiza la combinación de componentes y sistemas COTS para construir el nuevo sistema. En este enfoque, la integración del sistema puede estar incorporada en el proceso de desarrollo en lugar de ser una etapa separada y autónoma.

Existen tres categorías de elementos de software que pueden ser utilizados en un enfoque de reutilización en el desarrollo:

1. Servicios Web que se construyen de acuerdo con estándares específicos y se encuentran disponibles para ser llamados de forma remota.

2. Conjuntos de elementos que se crean como una entidad para ser incorporados en un entorno de componentes como .NET o J2EE.

3. Sistemas de software autónomos que se configuran para su utilización en un entorno específico.

La ingeniería de software centrada en la reutilización ofrece una ventaja evidente al reducir la necesidad de crear software nuevo, lo que a su vez conlleva una reducción en los costos y los riesgos. En general, también tiende a acelerar la entrega del software. No obstante, es inevitable que se deban hacer concesiones en algunos requisitos, lo que podría resultar en un sistema que no cumpla completamente con las necesidades de los usuarios. Además, existe una pérdida de control sobre la evolución del sistema, dado que las nuevas versiones de los componentes reutilizables no están bajo el control de la organización que los emplea (Sommerville, 2009).

Autoevaluación 11

1. **¿Qué caracteriza al modelo en cascada en el desarrollo de software?**
    a. Un enfoque de desarrollo paralelo.
    b. Un flujo continuo de pasos secuenciales.
    c. La retroalimentación constante con el cliente.
    d. La ausencia de fases de desarrollo.
2. **¿Quién es comúnmente atribuido como el autor de la primera descripción formal del modelo en cascada?**
    a. Winston Royce.
    b. Alan Turing.
    c. Richard Stallman.
    d. Linus Torvalds.
3. **¿Cómo se caracterizan las fases en el modelo en cascada en términos de secuencia temporal?**
    1. Se pueden llevar a cabo simultáneamente.
    2. Cada fase comienza antes de completar la fase anterior.
    3. No existe una secuencia lineal.
    4. Las fases no están relacionadas entre sí.
4. **¿Qué sucede en el modelo en cascada cuando se detecta un error en una fase particular?**
    a. Se ignora el error y se procede a la siguiente fase.
    b. Se retrocede hasta la fase inicial de análisis de requisitos del sistema.
    c. Se suspende el proyecto indefinidamente.
    d. Se comunica el error al cliente sin hacer correcciones.

5. **¿Cuál es uno de los propósitos de la fase de Pruebas en el modelo en cascada?**
    1. Evaluar la viabilidad económica del software.
    2. Transformar el diseño en un formato comprensible por la máquina.
    3. Identificar fallos y verificar que los resultados coincidan con los esperados.
    4. Realizar modificaciones en el software existente.
6. **¿Qué caracteriza al desarrollo incremental en el desarrollo de software?**
    1. Un enfoque rígido y secuencial.
    2. La elaboración de una versión definitiva desde el principio.
    3. La entrega de versiones progresivamente más avanzadas del sistema.
    4. La falta de retroalimentación del cliente.
7. **¿Cuál es una ventaja significativa del desarrollo incremental en comparación con el modelo en cascada?**
    a. Mayor rigidez en la gestión del proyecto.
    b. Facilita la obtención de retroalimentación del cliente.
    c. Requiere menos planificación inicial.
    d. Mayor costo asociado a cambios en los requisitos.
8. **¿Cuál es uno de los desafíos del desarrollo incremental mencionados en el texto?**
    1. Identificar los requisitos comunes entre los incrementos.
    2. Generar resistencia e insatisfacción con la nueva propuesta.
    3. Aplicar todas las funcionalidades tan pronto como los primeros incrementos se presenten.
    4. Adaptar una nueva forma de contratación en instituciones gubernamentales.

9. ¿En qué tipo de proyectos el desarrollo incremental puede ser menos adecuado?

a. En proyectos con un equipo diverso trabajando en distintas partes del sistema.
b. En proyectos pequeños y simples.
c. En proyectos donde no es necesario contar con retroalimentación del cliente.
d. En proyectos que no involucran software.

10. ¿Qué aspecto es fundamental para el éxito del desarrollo incremental en sistemas de gran envergadura?

1. No es necesario definir estructuras arquitectónicas sólidas.
2. Contar con un enfoque completamente burocrático.
3. Planificar de antemano las responsabilidades de los equipos involucrados.
4. Evolucionar de manera incremental en lugar de planificar.

11. ¿Qué caracteriza la reutilización de componentes de software en un contexto informal?

a. Se basa en un proceso de desarrollo específico.
b. Se enfoca en la creación de componentes nuevos desde cero.
c. Los profesionales buscan y adaptan diseños o códigos existentes.
d. Solo se aplica a proyectos pequeños.

12. ¿Cuál de las siguientes afirmaciones es cierta sobre los enfoques de desarrollo de software orientados a la reutilización?

1. No se basan en componentes de software reutilizables.
2. Utilizan exclusivamente componentes COTS.
3. Incorporan marcos de trabajo para ensamblar componentes reutilizables.
4. Se centran en la creación de software completamente nuevo.

13. ¿Cuál es una de las etapas intermedias en un enfoque orientado a la reutilización según el texto?

a. Definición de requisitos.
b. Análisis de componentes.
c. Validación.
d. Planificación del proyecto.

14. ¿Cuál es uno de los beneficios de la ingeniería de software centrada en la reutilización?

1. Reducción de costos y riesgos.
2. Pérdida de control sobre la evolución del sistema.
3. Mayor necesidad de crear software nuevo.
4. Retraso en la entrega del software.

15. ¿Qué categoría de elementos de software se menciona como parte de un enfoque de reutilización en el desarrollo?

a. Hardware incorporado en el sistema.
b. Sistemas de software autónomos.
c. Documentación técnica detallada.
d. Modelos de negocio.

3.4 Proceso de desarrollo de software

3.4.1 Introducción

El proceso de elaboración de software constituye una secuencia planificada y organizada de acciones y pasos destinados a concebir, diseñar, construir, evaluar, aplicar y mantener software de alto nivel. Este procedimiento está estructurado para dirigir de manera eficaz y eficiente la generación de soluciones informáticas que satisfagan los requerimientos del proyecto y las expectativas de los usuarios (Sulbarán, 2018).

La introducción al procedimiento de desarrollo de software conlleva la comprensión de su importancia en la creación de software funcional y confiable. En un contexto cada vez más tecnológico y orientado a la automatización, el proceso de desarrollo de software proporciona un marco organizativo para abordar las complejidades y desafíos inherentes a la construcción de aplicaciones y sistemas informáticos. Desde la etapa inicial de planificación hasta la ejecución y el mantenimiento continuo, este método orienta a los equipos de desarrollo a través de fases cruciales, incluyendo la definición de requisitos, el diseño, la programación, las pruebas y la entrega definitiva.

Esta presentación introductoria al proceso de desarrollo de software también hace hincapié en la necesidad de adaptarse a diferentes enfoques y metodologías según las particularidades del proyecto y las preferencias del equipo. Cada modelo de desa-

rrollo, ya sea el en cascada, el incremental, el ágil u otros, posee sus propias ventajas y restricciones, lo que pone de relieve la flexibilidad que se requiere en la gestión de proyectos de software. Además, este inicio destaca la importancia de establecer normas y buenas prácticas para asegurar la calidad del software resultante y la satisfacción de los usuarios finales.

En resumen, la introducción al proceso de desarrollo de software es fundamental para comprender la estructura y la finalidad de las actividades involucradas en la creación exitosa de software. Este procedimiento no solo aborda la fase técnica del desarrollo, sino que también contempla aspectos como la colaboración con los usuarios, la gestión de riesgos y la planificación de recursos. Como resultado, proporciona un marco sólido para alcanzar productos de software eficaces, confiables y alineados con las necesidades del mercado y los clientes (Martinez, 2015).

3.4.2 Estándares de desarrollo

Los estándares de desarrollo dentro del proceso de desarrollo de software comprenden conjuntos de directrices, prácticas y reglas instaurados con el propósito de asegurar la excelencia, uniformidad y eficacia en todas las fases del ciclo de vida del software. Estos estándares desempeñan un papel fundamental en orientar a los equipos de desarrollo y garantizar que el software elaborado cumpla con las condiciones establecidas por el proyecto, además de ser confiable, fácilmente mantenible y seguro (Martinez, 2015).

Algunos estándares en el proceso de desarrollo de software son:

Documentación: Establecimiento de estructuras y contenidos para la elaboración de documentación que abarca requisitos, especificaciones, diseño y guías de usuario.

Código fuente: Creación de normativas de diseño, convenciones de denominación y prácticas recomendadas para el código fuente del software.

Pruebas: Establecimiento de protocolos y enfoques para las pruebas, que engloban casos de prueba, criterios de aprobación y sistemas para registrar los resultados.

Gestión de configuración: Creación de flujos de trabajo para supervisar versiones, administrar modificaciones y monitorear la configuración del software.

Seguridad: Aplicación de lineamientos para enfrentar aspectos de seguridad, tales como la salvaguardia de información, la verificación de identidad y la prevención de debilidades en la protección.

Metodologías y modelos de desarrollo: Aplicación de estructuras como Scrum, Kanban, en cascada, desarrollo ágil, y otras similares, las cuales definen prácticas particulares para cada fase del procedimiento.

Cumplimiento normativo: Garantía de que el software se ajuste a las normativas y estándares particulares correspondientes al sector o al ámbito en el que se implemente.

Interoperabilidad: Establecimiento de normas y procedimientos con el propósito de asegurar una comunicación eficaz entre diversos sistemas y plataformas.

Documentación de usuarios: Elaboración de manuales y orientaciones para los usuarios, adhiriéndose a estructuras y enfoques predefinidos**.**

Auditoría y revisión: Creación de procedimientos para inspeccionar y confirmar la calidad del software durante su evolución.

La aplicación de estándares de desarrollo en el proceso de crear software ayuda a reducir fallos, fomentar la colaboración entre los integrantes del equipo, simplificar la comunicación con las partes interesadas y asegurar que el producto terminado sea seguro y cumpla con las anticipaciones. Estos estándares pueden ser adaptados a la organización en particular o estar fundados en regulaciones industriales ampliamente reconocidas, como las ISO/IEC 12207 para los procedimientos en el ciclo de vida del software o las ISO/IEC 25010 para medir la calidad del software (Martinez, 2015).

3.5 Conclusión

En conclusión, dentro de este capítulo hemos hablado de las metodologías de desarrollo de software, se han indagado descripciones que abarcan la totalidad del proceso, desde la concepción hasta la implementación de programas utilizando diferentes len-

guajes de programación. Estas metodologías presentan características particulares que guían su eficacia en proyectos. Han sido agrupadas en categorías de estructuradas, orientadas a objetos y ágiles, cada una con enfoques propios. Además, se han analizado detalladamente los ciclos de vida del desarrollo de software y se ha explorado en profundidad el concepto de paradigmas de proceso, que proporcionan marcos conceptuales para dirigir los proyectos.

Dentro de la categorización de los paradigmas de proceso, se han examinado el modelo en cascada con sus etapas secuenciales, el enfoque incremental basado en la liberación gradual de versiones, y la ingeniería de software orientada a la reutilización de componentes previamente construidos. También se han considerado otras aproximaciones. Se ha subrayado la importancia de establecer estándares en el proceso de desarrollo de software para garantizar la calidad y la confiabilidad de los productos resultantes. En resumen, la exploración de metodologías, definiciones y paradigmas de proceso en el desarrollo de software brinda una visión amplia y variada de los enfoques que afectan la gestión de proyectos, los ciclos de vida y la creación de software eficaz y confiable.

Autoevaluación 12

1. **¿Cuál es el objetivo principal del proceso de elaboración de software según el texto?**
   a. Generar software de bajo nivel.
   b. Concebir ideas para proyectos de software.
   c. Generar soluciones informáticas satisfaciendo requerimientos y expectativas.
   d. Evaluar aplicaciones existentes.
2. **¿Qué importancia tiene la introducción al proceso de desarrollo de software?**
   1. Ninguna importancia.
   2. Proporciona un marco organizativo para abordar desafíos tecnológicos.
   3. Define los requisitos técnicos de un proyecto.
   4. Establece las tareas de mantenimiento del software.
3. **¿Cuáles son algunas de las fases cruciales mencionadas en el texto?**
   a. Comunicación con usuarios y revisión de documentación.
   b. Diseño, pruebas y entrega definitiva.
   c. Comercialización y ventas.
   d. Reuniones de equipo y administración de recursos.
4. **¿Por qué es importante la flexibilidad en la gestión de proyectos de software?**
   1. Para evitar cualquier cambio en el proyecto.
   2. Para adaptarse a diferentes enfoques y metodologías.
   3. Para imponer un único modelo de desarrollo.
   4. Para acelerar el proceso de desarrollo.

5. **¿Cuál de los siguientes NO es un modelo de desarrollo de software mencionado en el texto?**
   a. En cascada.
   b. Incremental.
   c. Ágil.
   d. Continuo.
6. **¿Qué aspectos son contemplados en el proceso de desarrollo de software según el texto?**
   1. Colaboración con usuarios y gestión de riesgos.
   2. Administración financiera y ventas.
   3. Diseño gráfico y publicidad.
   4. Estándares de seguridad informática.
7. **¿Cuál es el propósito principal de establecer normas y buenas prácticas en el desarrollo de software?**
   1. Aumentar la complejidad del proyecto.
   2. Garantizar la satisfacción de los desarrolladores.
   3. Asegurar la calidad del software y la satisfacción de los usuarios finales.
   4. Acelerar el proceso de desarrollo.
8. **¿Qué tipo de productos de software busca alcanzar el proceso de desarrollo mencionado en el texto?**
   a. Productos económicos.
   b. Productos de entretenimiento.
   c. Productos eficaces, confiables y alineados con las necesidades del mercado y los clientes.
   d. Productos de alta complejidad.

9. ¿Cuál es uno de los elementos destacados en la introducción al proceso de desarrollo de software?

1. La exclusión de los usuarios en el proceso.
2. La importancia de evitar la planificación.
3. La gestión de recursos humanos.
4. La necesidad de adaptarse a diferentes enfoques.

10. ¿Qué importancia tiene la colaboración con los usuarios en el proceso de desarrollo de software?

a. Ninguna importancia.
b. Facilita la planificación del proyecto.
c. Ayuda a definir los requisitos y expectativas.
d. Retrasa el proceso de desarrollo.

11. ¿Qué función desempeñan los estándares de desarrollo en el proceso de desarrollo de software?

1. Establecer criterios de diseño.
2. Asegurar la seguridad del hardware.
3. Garantizar la calidad y eficacia en todas las fases del ciclo de vida del software.
4. Facilitar la comunicación con clientes.

12. ¿Qué aspectos abarcan los estándares de desarrollo relacionados con la documentación?

1. Especificaciones y guías de usuario.
2. Códigos de programación.
3. Pruebas de rendimiento.
4. Reglas de seguridad.

13. ¿Cuál es uno de los propósitos de los estándares de desarrollo en relación con la gestión de configuración?

a. Definir casos de prueba.
b. Supervisar versiones y gestionar modificaciones.
c. Crear flujos de trabajo para pruebas.
d. Prevenir debilidades en la protección.

14. ¿Qué tipo de estructuras y enfoques definen las metodologías y modelos de desarrollo mencionados en el texto?

1. Estructuras de diseño.
2. Pruebas de seguridad.
3. Prácticas particulares para cada fase del procedimiento.
4. Convenciones de denominación.

15. ¿Cómo contribuyen los estándares de desarrollo a la creación exitosa de software?

a. Reduciendo fallos y fomentando la colaboración.
b. Aumentando la complejidad del proyecto.
c. Limitando la comunicación con las partes interesadas.
d. Ignorando los estándares industriales reconocidos.

Referencias

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

Costanzo, M. (s.f). *Desarrollo incremental* [Blog]. https://mauricio.mwebs.com.uy/blog/qué-es-el-desarrollo-incremental/23

León Yacelga, A.R., Acosta Espinoza, J.L., & Díaz Vásquez, R.A. (2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Universidad Y Sociedad, 13*(5), 175-182. https://rus.ucf.edu.cu/index.php/rus/article/view/2223

Martinez, R.N. (2015). *El Proceso de Desarrollo de Software*. IT Campus Academy.

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

Santander Universidades. (2020, 21 de diciembre). *Metodologías de desarrollo de software: ¿qué son?* https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html

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

Sulbarán, H. (2014, 24 de septiembre). *Paradigmas en el desarrollo de software* [Blog]. https://acortar.link/VUTmOL

Sulbarán, I.. (2023, 27 de abril). Interfaz de usuario (ui): ejemplos y tipos. *Tiffin University*. https://global.tiffin.edu/noticias/interfaz-de-usuario-ui-ejemplos-y-tipos

§

Software development methodologies and processes
Metodologias e processos de desenvolvimento 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
This chapter focuses on the essential relevance of software development methodologies and processes to achieve reliable and high quality programs. Its central objective is to explore in detail a range of methodologies and paradigms that guide this process, outlining their fundamental characteristics and their classification into categories such as structured, object-oriented and agile. In addition, the chapter explores software development life cycles, the paradigms that guide project planning and execution, and classifies common process paradigms. Finally, it discusses the importance of the development process and the role of standards in ensuring consistency and quality in software projects. In summary, the chapter provides a solid understanding of the methodologies, paradigms, and processes that shape software creation in the software engineering industry.
Keywords: software development, projects, planning.

Resumo
Este capítulo se concentra na relevância essencial das metodologias e dos processos de desenvolvimento de software para obter software confiável e de alta qualidade. Seu objetivo central é explorar em detalhes uma série de metodologias e paradigmas que orientam esse processo, delineando suas características fundamentais e sua classificação em categorias como estruturada, orientada a objetos e ágil. Além disso, o capítulo explora os ciclos de vida do desenvolvimento de software, os paradigmas que orientam o planejamento e a execução do projeto e classifica os paradigmas de processos comuns. Por fim, ele discute a importância do processo de desenvolvimento e o papel dos padrões para garantir a consistência e a qualidade dos projetos de software. Em resumo, o capítulo oferece uma sólida compreensão das metodologias, dos paradigmas e dos processos que moldam a criação de software no setor de engenharia de software.
Palavras-chave: desenvolvimento de software, projetos, planejamento.

Capítulo 4

Especificación del software y enfoques ágiles

Mónica Elizabeth Páez Padilla
Diego Javier Portilla Martínez

Resumen

En este capítulo, se tratan asuntos fundamentales que abarcan desde la definición del software hasta la implementación de enfoques ágiles, con un enfoque particular en el marco de trabajo Scrum. Se destaca la relevancia de no confiar únicamente en metodologías y procesos para producir software de alta calidad, sino también en una precisa especificación y la aplicación de prácticas ágiles. El texto se enfoca en la etapa de especificación del software, incluyendo diseño, implementación, validación y evolución. También explora los enfoques ágiles en el desarrollo de software, detallando su idoneidad y el Manifiesto Ágil. Luego, se analizan los componentes clave de Scrum, incluyendo roles, responsabilidades, interacción, justificación de negocio y artefactos. Finalmente, se profundiza en los eventos esenciales de Scrum, incluyendo el Scrum Diario, la retrospectiva y el refinamiento del Product Backlog.
Palabras claves: Scrum, Product Backlog, Manifiesto Ágil, software.

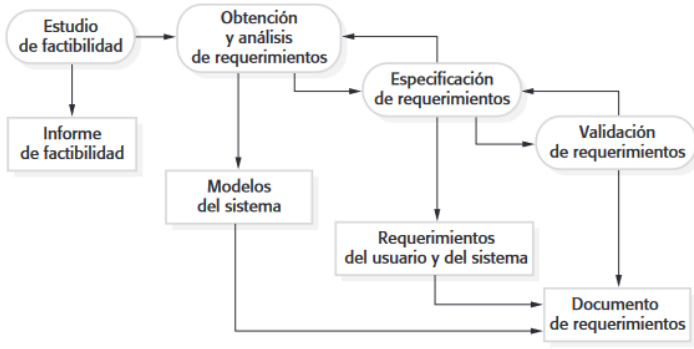
Paáz Padilla, M.E., y Portilla Martinez, D.J. (2023). Especificación del software y enfoques ágiles. En M.E. Páez Padilla (ed). *Análisis y diseño de sistemas.* (pp. 169-238). Religación Press. http://doi.org/10.46652/religacionpress.89.c85

4.1 Especificación del software

La elaboración de la especificación de software o la disciplina de ingeniería de requisitos comprende el procedimiento de comprender y definir los servicios necesarios del sistema, además de identificar las limitaciones en su funcionamiento y desarrollo. La ingeniería de requisitos es una fase especialmente crucial en el ciclo de desarrollo de software, ya que los fallos en esta etapa inevitablemente desencadenan problemas posteriores tanto en el diseño como en la implementación del sistema.

El proceso de ingeniería de requisitos (representado en la figura 11) se concentra en la elaboración de un documento de requisitos acordado que exprese las expectativas de los stakeholders del proyecto sobre lo que el sistema debería lograr. En general, estos requisitos se dividen en dos niveles de detalle. Los usuarios finales y los clientes solicitan un resumen de alto nivel de los requisitos, mientras que los desarrolladores del sistema requieren una descripción más detallada de los mismos (Sommerville, 2009).

Figura 11. Proceso de ingeniería de requerimientos.



Nota. Como se muestra en la figura anterior la especificación de requisitos de software consiste en una exposición exhaustiva del funcionamiento del sistema que será construido.

De acuerdo con (James, 2007) dentro del proceso de ingeniería de requisitos, se pueden identificar cuatro actividades principales:

Estudio de factibilidad: Se realiza una evaluación para determinar si las necesidades identificadas por el usuario son factibles de abordar con las tecnologías actuales de software y hardware. Esta evaluación considera si la propuesta del sistema conducirá a una relación costo-beneficio favorable desde una perspectiva empresarial y si su desarrollo puede ser gestionado dentro de las restricciones financieras existentes. Es esencial que esta evaluación de viabilidad sea rápida y económica. El resultado debe tener un impacto en la decisión de proceder o no a un análisis más detenido.

Obtención y análisis de requerimientos: Este procedimiento implica deducir los requisitos del sistema a través de la observación de sistemas ya en funcionamiento, conversaciones con usuarios y posibles proveedores, análisis de tareas y similares. Esto podría involucrar la creación de uno o varios modelos de sistemas y prototipos, con el propósito de aumentar la comprensión del sistema que se está por describir.

Especificación de requerimientos: Involucra la tarea de documentar la información recopilada durante la fase de análisis en un informe que define un conjunto de requerimientos. Este informe abarca dos categorías de necesidades: los requisitos del usuario, que ofrecen descripciones generales de los requisitos del sistema dirigidas al cliente y al usuario final del sistema; y los requisitos del sistema, que proporcionan una explicación detallada de las capacidades que se deben proporcionar.

Validación de requerimientos: Esta etapa se encarga de verificar que los requisitos sean lógicos, razonables y completos. En este proceso, es inevitable detectar errores en el documento de requisitos. Como resultado, es necesario ajustarlos para resolver dichos problemas.

Por supuesto, las fases en el proceso de requisitos no siguen una secuencia estrictamente lineal. El análisis de requisitos continúa durante la definición y especificación, y a lo largo del proceso, surgen nuevos requisitos; por lo tanto, las actividades de análisis, definición y especificación están interconectadas. En enfoques ágiles, como la programación extrema, los requisitos se desarrollan de manera incremental según las prioridades del

usuario, y la obtención de requisitos proviene de los usuarios que forman parte del equipo de desarrollo (Sommerville, 2011).

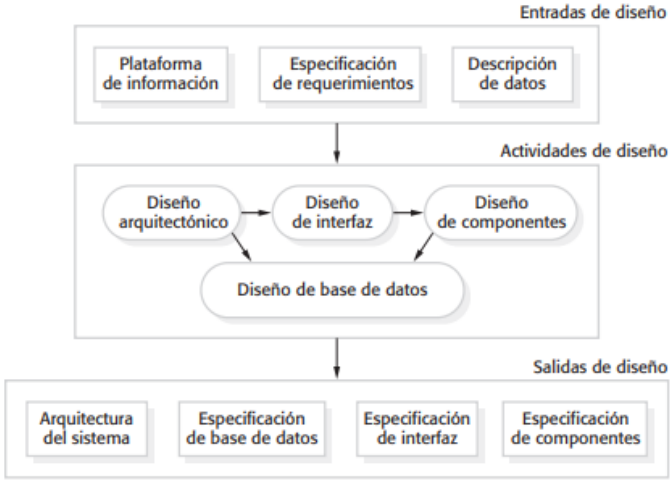
4.1.1 Diseño e implementación del software

La fase de desarrollo e implementación del software comprende el proceso de transformar una descripción del sistema en un sistema funcional. Siempre implica los procedimientos de diseño y programación del software y, en algunas ocasiones, puede requerir modificaciones en la descripción del software si se sigue un enfoque de desarrollo incremental (Sulbarán, 2018).

El diseño de software se define como una representación de la estructura del software a ser implementado, los modelos y estructuras de datos utilizados por el sistema, las conexiones entre los componentes del sistema y, ocasionalmente, los algoritmos empleados. Los diseñadores no llegan de inmediato a un diseño definitivo, sino que desarrollan el diseño de manera iterativa. Conforme avanzan en el diseño, añaden formalidad y detalles, al mismo tiempo que corrigen y ajustan diseños previos.

La figura 12 es un modelo general del proceso de diseño en la cual las etapas del proceso de diseño siguen un orden lineal, pero en la práctica, las actividades de diseño están estrechamente conectadas entre sí. En todos los procesos de diseño, es inevitable que se produzca retroalimentación entre las etapas, lo que da lugar a una revisión continua y a la mejora del diseño.

Figura 12. Modelo general del proceso de diseño.



Nota. La figura 12 representa un esquema conceptual de esta metodología, que muestra las aportaciones al proceso de diseño, las acciones llevadas a cabo durante el proceso y los documentos que se originan como resultados de dicho proceso.

La mayoría del software interactúa con otros sistemas de software, como sistemas operativos, bases de datos y middleware. Estos sistemas forman la "plataforma de software" en la que el software se ejecutará. La información acerca de esta plataforma es fundamental para el proceso de diseño, y los diseñadores deben determinar cómo integrarla con el entorno de software. La especificación de requerimientos describe la funcionalidad prevista del software, así como los requisitos de rendimiento y fiabilidad. Si el sistema debe trabajar con datos existentes, entonces la descripción de esos datos se incluirá en la especificación de la plataforma, de lo contrario, la organización de los datos se convertirá en un elemento de entrada para el proceso de diseño.

Las actividades en el proceso de diseño son diferentes según el tipo de sistema que se esté desarrollando. Por ejemplo, los sistemas en tiempo real necesitarán un diseño relacionado con la gestión del tiempo, mientras que los sistemas que no involucren bases de datos no requerirán un diseño de base de datos (Senn, 1997).

En la figura anterior, se representan cuatro tareas que podrían formar parte del proceso de diseño para sistemas de información:

En el libro Arquitectura de Sistemas de Información (Aguilar, 2005), afirma que:

Diseño arquitectónico: Se determina la configuración general del sistema, los elementos primordiales, sus conexiones y distribución.

Diseño de interfaces: Se establecen las interfaces entre los elementos del sistema. Estas descripciones de interfaz deben ser detalladas para que un componente pueda operar sin que otros requieran información sobre cómo está construido.

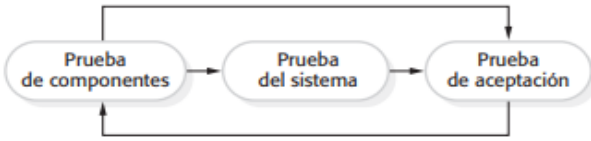
Diseño de componentes: Se enfoca en cada uno de los elementos del sistema y se elabora su funcionamiento. Esto puede implicar la creación de una descripción funcional o la lista de modificaciones en un componente reutilizado.

Diseño de base de datos: Se planifican las estructuras de datos del sistema y cómo se guardarán en una base de datos.

Estas acciones resultan en diversos productos de diseño, que también se ilustran en la figura. La forma y el nivel de detalle de estos productos pueden variar, desde documentos minuciosos hasta diagramas. En metodologías ágiles, estos productos pueden ser directamente integrados en el código del programa.

Los métodos de diseño estructurado se desarrollaron en las décadas de 1970 y 1980, lo que condujo al surgimiento del UML y al enfoque de diseño orientado a objetos. Estos métodos implican la creación de modelos gráficos y, en muchos casos, la generación de código a partir de estos modelos. El Desarrollo Basado en Modelos (MDD) representa una evolución de estos métodos, donde se generan modelos de software en varios niveles de abstracción. El diseño de programas para implementar el sistema se deriva de los procesos de desarrollo del sistema. Las herramientas de desarrollo de software pueden crear una estructura básica del programa a partir del diseño, y luego los programadores completan los detalles (Sommerville, 2011).

Figura 13. Etapas de Pruebas.



Nota. La depuración implica establecer hipótesis sobre el comportamiento del programa y ponerlas a prueba para identificar y corregir errores. Herramientas interactivas son útiles para depurar, mostrando valores de variables y rastreo de instrucciones ejecutadas.

La programación es una tarea individual y no sigue un proceso uniforme. Algunos programadores empiezan trabajando en componentes que conocen bien y luego avanzan hacia los menos familiares, mientras que otros siguen un enfoque contrario. A medida que desarrollan el código, las pruebas revelan fallos que deben ser corregidos, en un proceso conocido como depuración.

4.1.2 Validación de Software

La validación de software, o en un sentido más amplio, su verificación y validación (V&V), se lleva a cabo para demostrar que un sistema cumple con sus especificaciones y las expectativas del cliente. La principal técnica empleada en la validación es realizar pruebas del programa, donde el sistema se ejecuta utilizando datos de prueba simulados. Además de las pruebas, la validación también puede involucrar actividades de verificación, como inspecciones y revisiones en cada etapa del proceso de desarrollo de software, desde la definición de requisitos por parte del usuario hasta el desarrollo del programa. Debido a que las pruebas son predominantes, la mayoría de los costos asociados con la validación se incurren durante la implementación y en las fases posteriores (Lewis, 2011).

A menos que se trate de programas pequeños, no se debe realizar pruebas en los sistemas como una entidad única. En la figura 13 se representa un proceso de pruebas en tres etapas, donde en primer lugar se prueban los componentes individuales del sistema, luego se procede con las pruebas del sistema en su

conjunto y, finalmente, se realizan pruebas con los datos reales del cliente. Idealmente, los defectos en los componentes se descubren tempranamente en el proceso, mientras que los problemas de interconexión se identifican al integrar el sistema. Sin embargo, a medida que se encuentran defectos, es necesario depurar el programa, lo que podría requerir repetir otras fases del proceso de pruebas. Los errores en los componentes individuales pueden manifestarse durante las pruebas del sistema. Por lo tanto, este proceso es iterativo, con información que fluye desde etapas posteriores hacia las fases iniciales del proceso.

En palabras de Massol y Husted (2003), las etapas en el proceso de pruebas incluyen:

Prueba de desarrollo: Los desarrolladores del sistema realizan pruebas en los elementos que componen el sistema. Cada elemento se somete a pruebas de forma individual, sin que otros elementos del sistema estén presentes. Estos elementos pueden ser unidades simples, como funciones o clases de objetos, o grupos coherentes de estas entidades. Generalmente, se emplean herramientas de automatización de pruebas, como JUnit que facilitan la repetición de las pruebas de los componentes cuando se introducen nuevas versiones de estos.

Pruebas del sistema: Los elementos del sistema se agrupan para constituir el sistema en su totalidad. Esta acción tiene como finalidad detectar posibles errores que puedan surgir de interacciones no anticipadas entre los componentes, así como problemas de conectividad entre los mismos. Además, se busca verificar que el sistema cumple con sus requisitos tanto funcionales como no

funcionales, y se evalúan las propiedades emergentes del sistema. En sistemas de gran envergadura, este proceso puede constar de múltiples etapas, en las cuales los componentes se organizan en subsistemas que son evaluados de manera individual antes de ser integrados en el sistema definitivo.

Pruebas de aceptación: Esta etapa representa el último paso en el proceso de pruebas antes de que el sistema se considere apto para su uso operativo. En esta fase, el sistema se somete a pruebas utilizando datos proporcionados directamente por el cliente del sistema, en lugar de emplear datos simulados para las pruebas. Las pruebas de aceptación tienen la capacidad de poner de manifiesto errores y omisiones en la definición de los requisitos del sistema, ya que los datos reales ponen a prueba el sistema de manera diferente en comparación con los datos de prueba simulados. Además, estas pruebas permiten identificar problemas relacionados con los requisitos, en situaciones en las que las capacidades del sistema no satisfacen eficazmente las necesidades del usuario o cuando el rendimiento del sistema resulta insuficiente.

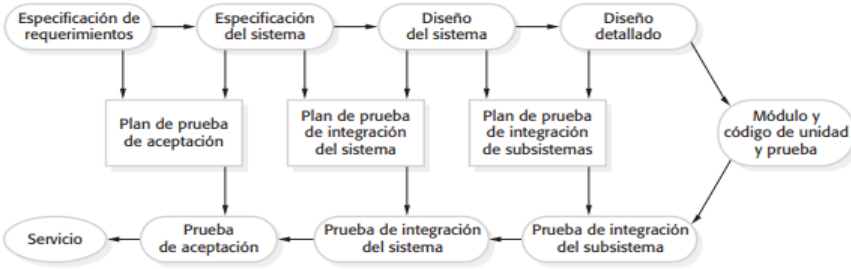
En general, los procesos de desarrollo y pruebas de componentes están estrechamente relacionados. Los programadores crean sus propios conjuntos de datos de prueba y evalúan el código incrementalmente a medida que lo desarrollan. Este enfoque tiene sentido desde un punto de vista económico, ya que el programador tiene un conocimiento profundo del componente y es el más adecuado para crear casos de prueba.

En un enfoque de desarrollo incremental, cada fase adicional debe someterse a pruebas a medida que se diseña, y estas

pruebas se basan en los requisitos específicos de esa fase. En la programación extrema, las pruebas se elaboran juntamente con los requisitos antes de iniciar el desarrollo. Este enfoque ayuda a los revisores y a los desarrolladores a comprender los requisitos y garantiza que no haya demoras en la creación de casos de prueba.

En situaciones en las que se utiliza un proceso de desarrollo planificado, como en el desarrollo de sistemas críticos, las pruebas se llevan a cabo mediante un conjunto de planes de prueba predefinidos. Un equipo de revisores independientes trabaja de acuerdo con estos planes, que se generan a partir de las especificaciones y el diseño del sistema (Aguilar, 2005).

Figura 14. Probando fases en un proceso de software dirigido por un plan.



Nota. La figura anterior ilustra la relación entre los planes de prueba en las etapas de desarrollo y pruebas, a menudo conocida como el "modelo V" de desarrollo (puede girarse para formar una "V" y así distinguirlo).

A veces, las pruebas de aceptación se denominan "pruebas alfa". Los sistemas personalizados se desarrollan exclusivamen-

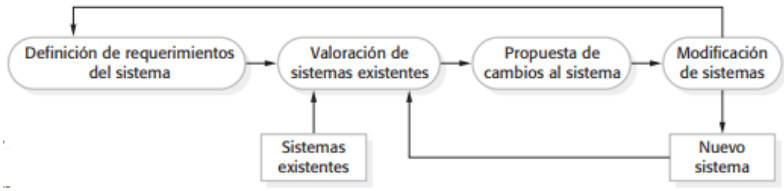
te para un cliente. El proceso de prueba alfa continúa hasta que tanto el desarrollador como el cliente están de acuerdo en que el sistema entregado cumple satisfactoriamente con los requisitos.

Una vez que un sistema se considera un producto de software, a menudo se recurre a un proceso de prueba conocido como "prueba beta". Esto implica ofrecer el sistema a algunos usuarios potenciales que están dispuestos a probarlo. Estos usuarios proporcionan comentarios sobre cualquier problema a los desarrolladores. Este proceso de retroalimentación somete el producto a un uso real y permite detectar errores que los constructores del sistema no anticiparon. Después de esta fase de retroalimentación, el sistema se ajusta y se lanza nuevamente, ya sea para más pruebas beta o para su lanzamiento al público en general.

4.1.3 Evolución de Software

La flexibilidad inherente a los sistemas de software es una de las principales razones por las cuales el software se está incorporando cada vez más en sistemas extensos y complejos. Una vez que se toma la decisión de fabricar hardware, efectuar modificaciones en su diseño conlleva un costo significativamente alto. Sin embargo, en cualquier momento durante o después del proceso de desarrollo del sistema, es posible realizar cambios en el software. Incluso las modificaciones sustanciales resultan más económicas que las equivalentes en el hardware del sistema. A lo largo de la historia, ha existido una clara separación entre el proceso de desarrollo de software y el proceso de evolución del software, que se refiere al mantenimiento de este.

Figura 15. Evolución del sistema.



Nota. La figura anterior muestra el proceso evolutivo, en el cual el software se modifica de manera constante a lo largo de su ciclo de vida, de acuerdo con los cambiantes requerimientos y necesidades del cliente.

El desarrollo de software se considera una actividad creativa en la que se crea un sistema de software a partir de una idea inicial y mediante un proceso de trabajo. No obstante, a veces, el mantenimiento del software se percibe como tedioso y carente de interés. A pesar de que, en la mayoría de los casos, los costos de mantenimiento superan con creces los costos iniciales de desarrollo, algunas veces los procesos de mantenimiento se consideran menos desafiantes que la creación original del software. La distinción tradicional entre desarrollo y mantenimiento está perdiendo relevancia. Es altamente improbable que cualquier sistema de software sea completamente innovador, por lo que es más apropiado ver el desarrollo y el mantenimiento como un continuo. En lugar de considerarlos como procesos independientes, tiene más sentido concebir la ingeniería de software como un proceso integrado (James, 2007).

Autoevaluación 13

1. **¿Cuál es el propósito de la actividad de «Estudio de factibilidad» en la ingeniería de requisitos?**
    a. Definir los requisitos del usuario.
    b. Documentar los requisitos del sistema.
    c. Evaluar si las necesidades son factibles y rentables.
    d. Verificar que los requisitos sean lógicos.
2. **¿Qué implica la actividad de «Obtención y análisis de requerimientos» en la ingeniería de requisitos?**
    1. Documentar los requisitos del sistema.
    2. Verificar que los requisitos sean lógicos.
    3. Deducir los requisitos del sistema a través de la observación y análisis.
    4. Evaluar la viabilidad financiera del proyecto.
3. **¿Qué tipo de necesidades abarca la actividad de «Especificación de requerimientos» en la ingeniería de requisitos?**
    1. Requisitos de usuario y requisitos de hardware.
    2. Requisitos de software y requisitos de seguridad.
    3. Requisitos del usuario y requisitos del sistema.
    4. Requisitos de rendimiento y requisitos de mantenimiento.
4. **¿Cuál es el propósito de la actividad de «Validación de requerimientos» en la ingeniería de requisitos?**
    a. Evaluar la viabilidad del proyecto.
    b. Documentar los requisitos del sistema.
    c. Verificar que los requisitos sean lógicos y razonables.

d. Identificar las necesidades de hardware.

5. **¿Cómo se relacionan las actividades de análisis, definición y especificación de requisitos en el proceso de requisitos?**
    1. Son actividades completamente independientes.
    2. Siguen una secuencia estrictamente lineal.
    3. Están interconectadas, y el análisis de requisitos continúa durante la definición y especificación.
    4. Se llevan a cabo en etapas separadas del proyecto.
6. **¿Cuál es el propósito del diseño de software en el proceso de desarrollo?**
    a. Documentar los problemas en el sistema.
    b. Transformar una descripción del sistema en un sistema funcional.
    c. Evaluar la viabilidad financiera del proyecto.
    d. Generar código directamente desde la descripción del sistema.
7. **¿Qué tipo de sistemas pueden requerir un diseño relacionado con la gestión del tiempo en el proceso de diseño?**
    1. Sistemas de información.
    2. Sistemas en tiempo real.
    3. Sistemas de base de datos.
    4. Sistemas de desarrollo incremental.
8. **¿Qué tarea del proceso de diseño se enfoca en establecer las interfaces entre los elementos del sistema?**
    a. Diseño arquitectónico.
    b. Diseño de componentes.

c. Diseño de interfaces.
d. Diseño de base de datos.

9. ¿Qué método de diseño implica la creación de modelos gráficos y, en algunos casos, la generación de código a partir de estos modelos?

a. Diseño estructurado.
b. Desarrollo Basado en Modelos (MDD).
c. Desarrollo orientado a objetos.
d. Desarrollo incremental.

10. ¿Qué proceso se lleva a cabo cuando los programadores corrigen los fallos que se revelan durante las pruebas del código?

1. Depuración.
2. Diseño de componentes.
3. Diseño de base de datos.
4. Evaluación de la viabilidad.

11. ¿Cuál es una de las principales ventajas del software en comparación con el hardware en términos de flexibilidad?

1. El software es más económico de producir.
2. El software no requiere mantenimiento.
3. Las modificaciones en el software son más económicas que en el hardware.
4. El software es más rápido en términos de procesamiento.

12. ¿Qué se entiende por evolución del software en términos de ingeniería de software?

a. La creación inicial de un sistema de software.
b. La fase de desarrollo de un sistema de software.
c. El proceso de mantenimiento y actualización del software.
d. La retirada de un sistema de software obsoleto.

13. ¿Cuál es la principal diferencia entre el desarrollo de software

y el mantenimiento del software?

1. El desarrollo de software es más económico que el mantenimiento.
2. El mantenimiento del software se considera menos desafiante.
3. El desarrollo de software es una actividad creativa, mientras que el mantenimiento es tedioso.
4. No hay diferencias significativas entre el desarrollo y el mantenimiento.

14. ¿Por qué la distinción tradicional entre desarrollo y mantenimiento del software está perdiendo relevancia?

1. Porque el mantenimiento del software se ha vuelto más caro que el desarrollo.
2. Porque los procesos de desarrollo y mantenimiento son independientes.
3. Porque la mayoría de los sistemas de software son completamente innovadores.
4. Porque se considera más apropiado ver la ingeniería de software como un proceso integrado.

15. ¿Cuál es una razón importante para considerar la evolución del software como un proceso continuo?

a. Para reducir los costos de mantenimiento.
b. Para evitar la creación de software completamente innovador.
c. Porque el mantenimiento del software es más económico que el desarrollo.
d. Para que el desarrollo y el mantenimiento estén mejor separados.

16. ¿Por qué la flexibilidad inherente al software es una ventaja en sistemas extensos y complejos?

a. Porque el software es más barato que el hardware.
b. Porque el software no requiere mantenimiento.
c. Porque las modificaciones en el software son más económicas que en el hardware.
d. Porque el software es más rápido que el hardware.

17. ¿Qué se entiende por evolución del software en términos de ingeniería de software?

1. La creación inicial de un sistema de software.
2. La fase de desarrollo de un sistema de software.
3. El proceso de mantenimiento y actualización del software.
4. La retirada de un sistema de software obsoleto.

18. ¿Cuál es una de las razones por las que a veces el mantenimiento del software se percibe como tedioso?

1. Porque es más económico que el desarrollo original del software.
2. Porque los costos de mantenimiento son significativamente más bajos que los del desarrollo.
3. Porque no es necesario realizar cambios en el software con el tiempo.
4. Porque se considera menos desafiante que la creación original del software.

19. ¿Por qué la distinción tradicional entre desarrollo y mantenimiento del software está perdiendo relevancia?

a. Porque el mantenimiento del software se ha vuelto más caro que el desarrollo.
b. Porque los procesos de desarrollo y mantenimiento son independientes.

c. Porque la mayoría de los sistemas de software son completamente innovadores.
d. Porque se considera más apropiado ver la ingeniería de software como un proceso integrado.

20. ¿Qué ventaja principal ofrece la flexibilidad del software en comparación con el hardware?

1. El software es más económico de producir.
2. El software permite realizar cambios más económicos.
3. El software no requiere mantenimiento.
4. El software se crea de manera más rápida.

4.2 Introducción a los enfoques ágiles

La estrategia ágil en el ámbito del desarrollo de software tiene como objetivo principal la entrega continua de sistemas de software operativos mediante iteraciones rápidas.

No obstante, la expresión "metodología ágil" puede ser engañosa, ya que sugiere que el enfoque ágil es la única manera de encarar el desarrollo de software. En realidad, la metodología ágil no proporciona instrucciones precisas sobre cómo llevar a cabo el desarrollo de software. Más bien, representa una mentalidad orientada a la colaboración y a los flujos de trabajo, y establece un conjunto de principios que dirigen nuestras decisiones en lo que respecta a qué hacer y cómo hacerlo.

Específicamente, las metodologías ágiles de desarrollo de software se centran en proporcionar fragmentos funcionales de software en un corto período de tiempo para mejorar la satisfacción del cliente. Estas metodologías adoptan enfoques flexibles y fomentan la colaboración en equipo para lograr mejoras continuas. En resumen, el desarrollo ágil de software implica que equipos pequeños y autónomos de desarrolladores y representantes de negocios se reúnan regularmente en persona a lo largo del ciclo de vida del desarrollo de software. La metodología ágil aboga por una aproximación simplificada a la documentación de software y acepta los cambios que puedan surgir en diversas etapas del ciclo de vida en lugar de resistirse a ellos (Hat, 2023).

4.2.1 ¿Cuándo y por qué enfoques ágiles?

El sentido común, que rara vez es común, sobre todo entre profesionales que han invertido años en trabajar con sistemas, estructuras y protocolos predefinidos tras finalizar sus estudios, se convierte en un componente esencial dentro del nuevo enfoque ágil para gestionar proyectos. Aunque se etiqueta como "metodología ágil", esta perspectiva no impone un conjunto rígido de reglas para la ejecución de proyectos. En lugar de ello, representa un modo de pensar acerca de la colaboración y los flujos de trabajo, estableciendo un conjunto de valores que guían las decisiones sobre qué hacer y cómo llevarlo a cabo.

Las metodologías ágiles para el desarrollo de software buscan entregar segmentos operativos de sistemas en un tiempo breve, otorgando prioridad a la satisfacción del cliente. Estas metodologías adoptan enfoques adaptables y trabajo en equipo para lograr mejoras en curso. Se centran en equipos autoorganizados que se reúnen con regularidad a lo largo del ciclo de vida del desarrollo. La metodología ágil favorece la simplificación de la documentación y la aceptación de cambios en lugar de resistirlos.

Las metodologías ágiles fomentan tres objetivos esenciales en los proyectos: proporcionar valor, minimizar desperdicios y mejorar continuamente. Aunque estos principios parezcan evidentes, a menudo son pasados por alto en la gestión convencional de proyectos. También se vinculan con el ciclo de Deming (Planificar, Hacer, Verificar y Actuar) y promueven una mentalidad de mejora continua.

El Triángulo de Hierro, que engloba Costo, Tiempo, Alcance y Calidad, juega un papel crucial en las metodologías ágiles. Aquí, los aspectos de Costo y Tiempo se mantienen constantes, permitiendo que el Alcance sea el elemento variable para lograr resultados de alta calidad.

Se menciona el "Cono de Incertidumbre", que apunta a que la incertidumbre disminuye conforme avanza el proyecto. Al combinar estos conceptos, como si estuvieran mezclando ingredientes en un recipiente, se enfatiza la importancia de tener objetivos claros, un enfoque iterativo e incremental y la capacidad de adaptarse en la gestión de proyectos.

En síntesis, el enfoque ágil aporta claridad a conceptos que a menudo son pasados por alto, promoviendo una mentalidad colaborativa, adaptable y orientada a la mejora constante en la gestión de proyectos (Dremyn, 2020).

4.2.2 Manifiesto y Principios Agiles

Después de examinar los valores propuestos por las metodologías ágiles, exploraremos los 12 principios en los que se basan (Sentrio, 2021).

1. Satisfacer al cliente mediante la entrega temprana y continua

El primer principio del Manifiesto Ágil se enfoca en la satisfacción del cliente y destaca cómo una entrega temprana y constante de software valioso es esencial para lograrla. Esto aumenta

las probabilidades de cumplir con las demandas de los clientes y acelera el retorno de la inversión.

En un contexto en constante cambio, retrasar la entrega de software resulta insatisfactorio para los clientes. Dado que los usuarios utilizan el software en una amplia variedad de actividades y buscan cambios inmediatos, la demora no es una opción viable. Por lo tanto, el equipo de desarrollo debe realizar entregas más ágiles que agreguen valor a los usuarios.

Además, las entregas frecuentes y rápidas permiten que los clientes reciban valor en menor tiempo y con mayor frecuencia. Esto también facilita la retroalimentación temprana por parte del cliente, lo que disminuye la posibilidad de tener que realizar cambios significativos en etapas posteriores.

2. Aprovechar el cambio como ventaja competitiva

El segundo principio de la declaración promueve la adaptación a los cambios en beneficio del cliente, permitiendo ajustes en los requisitos incluso en las etapas finales de un proyecto.

En las prácticas tradicionales de gestión de proyectos, introducir cambios en los requisitos al final del proceso solía implicar un aumento del alcance y, por ende, un incremento de los costos. Sin embargo, en los enfoques ágiles, los equipos reconocen la potencial utilidad de estos cambios para los clientes y responden de manera eficaz a ellos.

Dada la naturaleza en constante evolución del entorno, resulta sumamente desafiante prever con exactitud los requisitos definitivos de un software. Destinar recursos a un producto que, al ser entregado a los usuarios, ya no resulta relevante para ellos carece de sentido para una empresa. Por lo tanto, aceptar y abrazar los cambios proporcionará al cliente una ventaja competitiva al abordar las necesidades actuales de sus usuarios.

3. Entregar valor frecuentemente

El tercer principio del Manifiesto Ágil profundiza en la noción de entrega continúa presentada en el primer principio fundamental. Específicamente, se refiere a la importancia de proporcionar actualizaciones más pequeñas del software en intervalos más cortos.

Estas entregas de menor escala requieren un menor tiempo de planificación y disminuyen la probabilidad de errores en su desarrollo. Además, un aumento en la frecuencia de entregas conlleva una mayor retroalimentación por parte del cliente, lo que previene la necesidad de solicitar cambios significativos a los desarrolladores en etapas posteriores.

Este principio no solo sigue siendo relevante en la actualidad, sino que ha ganado más importancia en los últimos años. Las versiones se liberan semanalmente o incluso diariamente.

4. Cooperación negocio-desarrolladores durante todo el proyecto

Este principio aboga por abolir las divisiones que existen entre los equipos de negocio y desarrollo de software, con el propósito de mejorar la comprensión y la colaboración recíproca, en aras de lograr resultados más efectivos.

Históricamente, los responsables de negocios y los desarrolladores operaban en esferas separadas. Otros roles intervenían para traducir las ideas de los primeros a un lenguaje comprensible por los segundos. Con este principio, los promotores del Manifiesto Ágil buscan fomentar la interacción diaria entre ambos grupos, abordando cualquier malentendido previamente, compartiendo retroalimentación de manera mutua y, en última instancia, alineando sus intereses.

5. Construir proyectos en torno a individuos motivados

El quinto principio de Agile aboga por el fortalecimiento de la motivación entre los miembros del equipo de desarrollo, para que puedan ejecutar los proyectos de manera óptima.

La segunda parte de este enunciado se enfoca en un aspecto crucial: la confianza en el equipo para generar software de alta calidad de manera autónoma, contando con el entorno y el apoyo adecuados. Si los integrantes del equipo no participan en las decisiones de los proyectos en los que trabajan, no podrán conectar con el propósito de estos, lo que resultará en una disminución de su compromiso y rendimiento.

6. Utilizar la comunicación cara a cara

El sexto principio del manifiesto aborda la comunicación óptima para lograr proyectos exitosos: el diálogo en persona. Entre todas las modalidades de comunicación disponibles, el cara a cara es la más eficaz, ya que disminuye considerablemente los tiempos de respuesta y las posibilidades de malentendidos.

Sin embargo, la naturaleza de nuestro entorno laboral ha cambiado significativamente desde 2001. El trabajo a distancia se ha generalizado y han surgido numerosas herramientas que facilitan la comunicación y la colaboración a distancia. Esto no significa que este principio haya perdido relevancia en la actualidad, sino que ha evolucionado.

7. Software funcionando como medida de progreso

El séptimo principio ágil hace hincapié en que la medida fundamental para evaluar el progreso de un proyecto en las organizaciones debe ser la presencia de software en funcionamiento. Las actividades realizadas por el equipo de desarrollo que no contribuyen a la creación de un producto que responda a las demandas del cliente apenas o nada aportan en términos del verdadero avance del proyecto.

No importa cuántos errores el equipo haya corregido o cuánto código haya escrito si no se ha trabajado en pos de obtener un software que cumpla con las necesidades del cliente. En la actualidad, este principio tiene un alcance más amplio. Incluso si el

software funciona correctamente, si no se ha entregado al cliente, el equipo de desarrollo no ha logrado avanzar. No ha generado valor para el cliente hasta que el producto final esté en sus manos.

8. Promover y mantener un desarrollo sostenible

La octava premisa del manifiesto sostiene que en la operativa ágil se busca optimizar los métodos de trabajo para evitar excesos y lograr entregar con frecuencia al mercado soluciones de software que generen valor para los usuarios, sin necesidad de exponerse a esfuerzos desmesurados. En otras palabras, todas las partes involucradas en el proceso de desarrollo de software deben mantener un ritmo que sea viable para todos, evitando tensiones o presiones exageradas.

Por lo general, este principio se relaciona con la aplicación a equipos de desarrollo que están sobrecargados al proporcionar nuevas funcionalidades a los usuarios. Sin embargo, también puede aplicarse en sentido contrario. Si el equipo de desarrollo está liberando un número excesivo de nuevas funcionalidades para los usuarios, se deberá moderar la producción o invertir esfuerzos en capacitar a los usuarios en la utilización de estas características.

9. La excelencia técnica mejora la agilidad

El noveno enunciado del manifiesto enfatiza que otorgar importancia a los aspectos técnicos durante el proceso de desarrollo de un producto de software contribuye a la agilidad. Resulta más

factible realizar actualizaciones posteriores en el software si se ha construido de manera meticulosa y cuenta con un diseño sólido, en comparación con una construcción descuidada.

Con frecuencia, las organizaciones priorizan el time-to-market de sus productos por encima de la calidad del código. Esta perspectiva es comprensible, ya que la excelencia técnica no impacta directamente a los usuarios finales ni genera beneficios tangibles para el negocio. Sin embargo, si se relega, acabará afectando la velocidad, los plazos de entrega y la capacidad de mejorar el producto ante nuevas necesidades. En última instancia, se sacrificará la agilidad.

Un indicador estrechamente relacionado con la falta de priorización de un código de alta calidad es la deuda técnica. En este artículo, explicamos en qué consiste y por qué es importante considerarla en tus proyectos**.**

10. La simplicidad es fundamental

El décimo principio del Manifiesto Ágil presenta un enfoque altamente pragmático: adoptar la forma más simple de abordar una situación. El cliente no recompensa por el esfuerzo invertido, sino por la entrega de una solución que satisfaga sus necesidades.

Algunas estrategias para evitar esfuerzos superfluos incluyen automatizar tareas manuales, eliminar procedimientos redundantes y hacer uso de bibliotecas preexistentes. En última instancia, se trata de dirigir el tiempo y la energía del equipo hacia acciones que verdaderamente generen valor.

11. Equipos auto-organizados para generar más valor

La undécima premisa establece que los equipos que disfrutan de la libertad y confianza adecuadas son los que logran los resultados más sobresalientes.

Este principio guarda una estrecha relación con algunos de los conceptos presentados anteriormente en el manifiesto. Para lograr una comunicación y colaboración efectivas entre el negocio y los desarrolladores, para emplear el software en funcionamiento como medida de avance y para cimentar nuestros proyectos en torno a individuos motivados, resulta fundamental permitir que los equipos de desarrollo operen sin un control excesivo y se organicen internamente en todos los aspectos del proceso de desarrollo de software.

12. Reflexión y ajustes frecuentes del trabajo de los equipos

El último principio de Agile aborda la noción de mejora constante. Los equipos deben llevar a cabo revisiones periódicas de su trabajo con el propósito de ajustarlo y elevar su eficacia. Se trata de un concepto esencial que motiva a individuos, equipos y organizaciones a aspirar al éxito mediante la búsqueda continua de mejoras en lugar de conformarse.

4.2.3 Valores y Pilares de Scrum

Valores de la metodología Scrum

No podrás adquirir una comprensión completa de Scrum sin asimilar los cinco valores fundamentales de Scrum, los cuales están definidos en la Guía de Scrum: (Martins, 2023).

1.- Compromiso: El equipo Scrum opera como una entidad unificada, y la confianza entre sus miembros es esencial. Los integrantes del equipo Scrum se comprometen con el sprint durante su duración y se dedican a una mejora constante con el propósito de encontrar la solución óptima.

2.-Valentía: En el transcurso de un sprint Scrum, es posible que el equipo se enfrente a retos complejos que no tengan respuestas precisas. Los equipos Scrum poseen la valentía para plantear preguntas desafiantes y abiertas, y para responder con sinceridad con el objetivo de alcanzar la solución más adecuada.

3.-Enfoque: En cada sprint de Scrum, el equipo Scrum ejecutará tareas seleccionadas de una lista de pendientes del producto. La atención del equipo Scrum se concentra en las labores elegidas de dicha lista, las cuales resultarán en la entrega de sus productos al final de cada sprint.

4.-Actitud receptiva: No todo marchará sin contratiempos en el contexto de Scrum. Los miembros del equipo Scrum deben estar dispuestos a acoger nuevas ideas y oportunidades que contribuyan a su aprendizaje individual y a la mejora del producto o proceso.

5.-Respeto: La colaboración es un elemento esencial para comprender la esencia de Scrum, y para fomentar una colaboración sólida en el equipo, los integrantes deben tratarse entre sí con respeto, además de mostrar consideración hacia el Scrum Master y el proceso Scrum en sí.

Pilares de Scrum

La estructura de la metodología Scrum, como discutimos previamente, se cimienta en tres fundamentos: los eventos, los roles y los artefactos. Además, esta metodología opera en ciclos de trabajo llamados "sprints", los cuales tienen una duración predefinida que suele oscilar entre una semana y un mes. ¿Y por qué se establece un límite de un mes para la duración de los sprints? Esto se debe a que, al extender este período, aumenta el riesgo de cambios tanto en los requisitos del desarrollo como en el contexto en el que se desenvuelve. Al finalizar cada sprint, se genera un incremento que se suma al producto desarrollado hasta ese punto.

Cuando comprendes la esencia de Scrum, te das cuenta de que al iniciar un sprint es posible que tengas un conocimiento limitado, pero tienes la flexibilidad de adaptar tus procedimientos y necesidades en función de la información que vayas obteniendo durante el proceso del sprint (Martins, 2023).

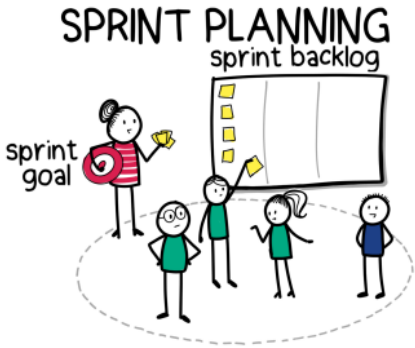
Eventos de Scrum

Entonces, ¿qué implica Scrum en realidad? ¿Cómo se desenvolverá tu equipo al adoptar Scrum? Así es como se desarrolla el proceso de Scrum:

1.-Organiza tu trabajo pendiente: Al iniciar un sprint de Scrum, el líder del equipo (también conocido como Scrum Master) determinará qué trabajo seleccionar de la lista de tareas pendientes, es decir, las labores que requieren ejecución. Para garantizar un sprint de Scrum efectivo, es esencial que el trabajo pendiente relacionado con el producto esté meticulosamente documentado en un solo lugar. Considera la posibilidad de utilizar una herramienta de gestión de proyectos para centralizar toda esta información.

2.-Sprint Planning: Lleva a cabo una reunión de planificación del sprint. Antes de iniciar el sprint de Scrum, es vital definir en qué se enfocará el equipo. Durante esta sesión de planificación del sprint, se evaluará qué parte del trabajo pendiente será el foco principal durante este sprint de Scrum en particular. Para comenzar, puedes utilizar nuestra plantilla gratuita para la planificación de Sprint.

Figura 16. Sprint Planning.



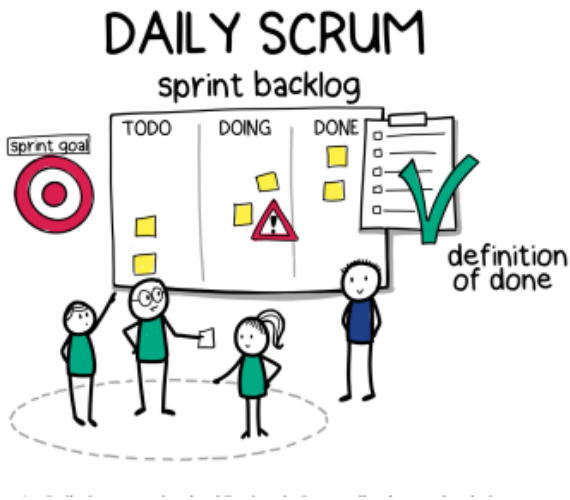
Nota. Durante el Sprint Planning se eligen los elementos requeridos del Product Backlog para alcanzar el Sprint Goal.

3.- Comienza tu sprint de Scrum: Por lo general, un sprint tiene una duración de dos semanas, aunque puedes optar por Sprint más cortos o largos según lo que resulte más adecuado para tu equipo. A lo largo del sprint, el equipo se dedicará a trabajar en las tareas pendientes establecidas durante la reunión de planificación del sprint.

4.- Daily Stand Up: Programa reuniones diarias de seguimiento de Scrum. La Reunión Diaria de Pie es un encuentro diario destinado al equipo de desarrollo, con una duración de 15 minutos. Estas reuniones diarias de seguimiento brindan la oportunidad de informar sobre las labores en progreso y abordar cualquier inconveniente inesperado que haya surgido. El objetivo primordial de estas reuniones es planificar el trabajo de las próximas horas y evaluar el progreso de las tareas. Si deseas optimizar

la efectividad de estas reuniones diarias de seguimiento, puedes utilizar nuestra plantilla gratuita diseñada para tal fin.

Figura 17. Daily Scrum.



Nota. La Daily Scrum es donde el Equipo de Desarrollo planea el trabajo colaborativo de las próximas 24 horas para lograr el Sprint Goal.

5.- Sprint Review: Exhibe tus logros durante la evaluación del sprint. Una vez que hayas completado el sprint de Scrum, tu equipo debe congregarse para llevar a cabo una evaluación del sprint. La revisión del sprint se extenderá durante un máximo de 4 horas en el caso de sprints de un mes. En este intervalo, tu equipo Scrum expondrá el trabajo que ha sido considerado como "Completado" para que los participantes lo aprueben o inspeccionen.

Figura 18. Sprint Review.

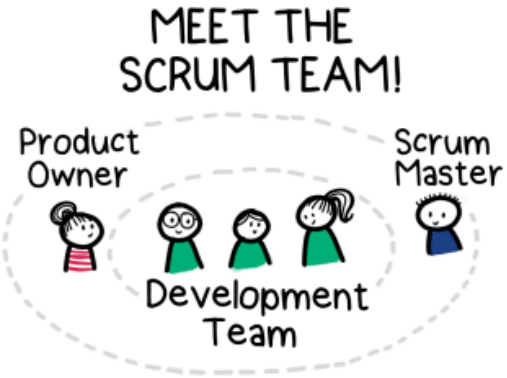
Nota. El propósito de la Sprint Review es inspeccionar el trabajo que se ha hecho hasta la fecha y decidir cuáles serán los siguientes pasos en base a lo que hemos aprendido

6. Sprint Retrospective: Dialoga y contempla durante la revisión retrospectiva del sprint. Al concluir cada sprint, dedica un momento para analizar cómo se desenvolvió y qué áreas podrían ser potenciadas en el futuro. Mantén en mente que en Scrum se abraza la idea de una mejora constante, por lo tanto, no dudes en experimentar con nuevos procedimientos o redefinir estrategias que consideras menos efectivas en el próximo sprint (James, 2007).

Roles de Scrum

Para comprender a fondo qué implica Scrum, es esencial tener un conocimiento sólido de los roles principales que la metodología Scrum distingue, ya que estos desempeñan un papel fundamental en la implementación efectiva de este enfoque ágil:

Figura 19. Roles de Scrum.



Nota. Tres roles para equilibrar tres perspectivas: valor, calidad y proceso.

a. **Product Owner o responsable del producto:** Esta es la figura responsable del backlog del producto en espera (product backlog). Está estrechamente relacionado con las necesidades de los usuarios y se dedica a transmitir la perspectiva del usuario tanto a su equipo como a otros líderes implicados. Los efectivos encargados de produc-

tos aportan claridad acerca de qué debe ser abordado a continuación debido a su relevancia. En última instancia, deberían ser los encargados de decidir cuándo algo está preparado para ser entregado (preferiblemente, con una tendencia hacia entregas frecuentes).

b. **Scrum Master:** El Scrum Master es aquel individuo encargado de liderar los diversos acontecimientos de Scrum. Puedes concebirlo como el administrador del proyecto y el facilitador de Scrum. El Scrum Master tiene la responsabilidad de impulsar las reuniones diarias de seguimiento y coordinar las sesiones de planificación, revisión y evaluación retrospectiva del sprint.

c. **Equipo Scrum:** El grupo Scrum abarca a todos aquellos involucrados en el sprint en curso. Los integrantes del equipo deben asumir la responsabilidad de su propia organización y trabajar en colaboración para alcanzar el objetivo esencial de Scrum, que es la búsqueda constante de mejoras.

Artefactos de Scrum

Los elementos fundamentales de Scrum son esenciales para la comprensión de esta metodología. En el contexto de Scrum, un artefacto representa una construcción que generas, como una herramienta destinada a solventar un desafío. Se identifican tres elementos cruciales que definen la esencia de Scrum: el backlog

de producto, el backlog de sprint y el incremento del producto (Sulbarán, 2018).

a. Product Backlog, el trabajo pendiente del producto

El Backlog de Producto constituye el componente de Scrum que recopila el inventario de tareas por realizar. Quien supervisa este backlog, también conocido como la pila de producto, es el propietario del producto, quien se encarga de jerarquizar los elementos en esta lista. Es crucial entender que la presencia de elementos en la lista de tareas pendientes no implica necesariamente que el equipo los abordará; en lugar de eso, los elementos en el Backlog de Producto representan opciones que el equipo podría considerar durante un sprint de Scrum. Los responsables del proyecto tienen la responsabilidad de ajustar y actualizar de forma constante el trabajo pendiente en función de nueva información y la lista de requisitos obtenida de los clientes, el mercado o el equipo del proyecto.

b. Sprint Backlog, el trabajo pendiente del sprint

El Backlog de Sprint, también denominado pila de sprint, consiste en el conjunto de tareas por abordar durante el sprint en cuestión. En otras palabras, representa la serie de labores o elementos a los cuales tu equipo se ha comprometido durante el sprint de Scrum. Estos elementos son extraídos de la lista de tareas pendientes del producto durante la reunión de planificación del sprint y se incorporan al plan de trabajo específico para el sprint de tu equipo, si este existe.

Si bien es posible que no todas las tareas pendientes se culminen durante cada sprint, es poco común que añadas nuevas tareas a la lista de trabajo pendiente una vez que el sprint ha comenzado. Si observas que esto ocurre con frecuencia, dedica más tiempo a la fase de planificación del sprint para tener una comprensión más sólida de qué labores abordarás durante el transcurso del sprint.

c. Incremento del producto

El producto incrementado es lo que proporcionarás al concluir cada sprint. Este componente de Scrum puede englobar desde un nuevo producto o función, hasta mejoras o correcciones de errores, según lo requiera tu equipo. La presentación del incremento está programada durante la evaluación del sprint. En ese instante, la entrega del incremento estará sujeta a la opinión de los participantes de Scrum, en función de si consideran que está "Completado" o no.

4.2.4 Marco Cinefyn – Tipos de Dominio

Una manera potencialmente más comprensible de ilustrar el contexto en el cual Scrum resulta más efectivo está relacionada con la perspectiva sobre la complejidad del Marco Cynefin o Modelo Cynefin, introducido en el año 2000 por Dave Snowden.

El Marco o Modelo Cynefin contrasta las atribuciones de cinco categorías distintas de complejidad: simple, complicado, complejo, caótico y el área central de desorden. Vamos a explorar cada uno de estos dominios (Labs, 2019).

1. Dominio Simple: Dentro del Marco Cynefin, el Ámbito Simple es donde se trata con situaciones de simplicidad. Dave Snowden caracteriza este dominio como aquel en el que es sencillo discernir las causas y sus consecuencias. En general, la respuesta correcta es evidente, ampliamente reconocida y no objeto de debate. En este dominio, prevalecen las mejores prácticas y soluciones establecidas para problemas conocidos. Los procesos más efectivos aquí son aquellos que establecen una secuencia lógica de pasos y se ejecutan de manera reiterada. Ejemplos de esta categoría incluyen la producción en serie de un mismo artículo o la instalación uniforme de un sistema en diversos clientes. Si bien Scrum podría ser aplicable en este contexto, los procedimientos que involucran pasos claramente definidos resultan considerablemente más eficientes.

2. Dominio Complicado: Otro de los ámbitos contemplados en el Marco Cynefin es el Ámbito Complicado. En esta categoría se abordan problemáticas de complejidad, destacan las mejores prácticas y la intervención de expertos. Aquí, existe una multiplicidad de soluciones correctas para un mismo problema, aunque se requiere la intervención de expertos para identificarlas. Un caso típico en este contexto podría ser la resolución de problemas de rendimiento en software o bases de datos, la sincronización de semáforos en una intersección de tres avenidas o la optimización de la distribución logística de productos. Aunque es posible aplicar Scrum, no necesariamente sería la opción más eficaz para abordar estas situaciones. Aquí, un enfoque que involucre a expertos en el campo, quienes analicen la situación, exploren varias alternativas y propongan soluciones basadas en las mejores prác-

ticas, suele ser más adecuado. Una práctica común en este ámbito es el mantenimiento de sistemas y el soporte técnico.

3. Dominio Complejo: Dentro del Modelo Cynefin de Dave Snowden encontramos el Ámbito Complejo. Nos hallamos en este dominio cuando nos enfrentamos a desafíos complejos, cuyos resultados se tornan más impredecibles. Para las situaciones que abarca este ámbito, no existen ni mejores ni buenas prácticas predefinidas.

Dentro del Ámbito Complejo, no hay ni mejores ni buenas prácticas establecidas. Simplemente, no se puede prever de antemano si una solución específica será efectiva. Solo es posible evaluar los resultados y ajustarse en consecuencia. Este es el terreno de las prácticas emergentes. Las soluciones que se descubren rara vez pueden aplicarse de manera idéntica a otros problemas similares, con resultados idénticos. Para operar en la complejidad, es necesario crear entornos que permitan la experimentación y minimicen el impacto del fracaso. Se requiere una alta dosis de creatividad, innovación, interacción y comunicación. El desarrollo de nuevos productos o la incorporación de funciones adicionales en productos existentes representa un ámbito complejo en el que Scrum se emplea con frecuencia para actuar, evaluar y ajustar las prácticas emergentes de un equipo de trabajo.

4.- Dominio Caótico: Dentro del Marco Cynefin, en el Ámbito Caótico nos encontramos con situaciones caóticas que exigen una respuesta inmediata. Nos hallamos en una crisis y es necesario tomar medidas inmediatas para restablecer algún grado de orden. Pongamos como ejemplo un escenario en el que el siste-

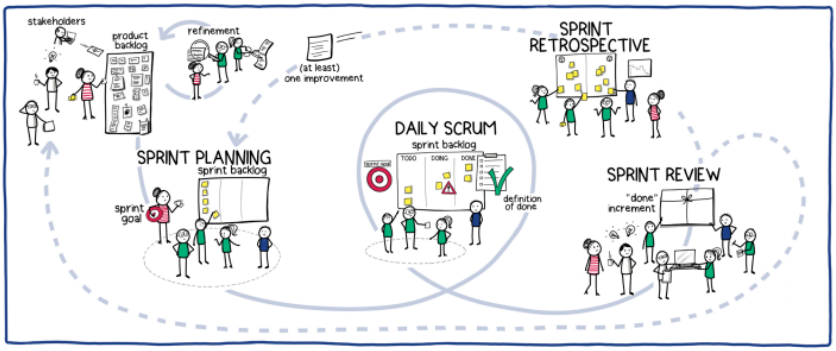
ma de programación de vuelos en un aeropuerto muy concurrido deja de funcionar. En este contexto, no sería apropiado aplicar Scrum; en su lugar, se requiere una acción inmediata, alguien debe asumir el control y sacar la situación del caos. Por ejemplo, resolver el problema de inmediato (sin importar el enfoque técnico), y luego, una vez fuera del caos, evaluar y aplicar una solución más sólida si es necesario. Este ámbito es donde la improvisación impera.

5. Dominio Desordenado: Según Dave Snowden, ingresamos al territorio del desorden cuando carecemos de claridad sobre en qué dominio nos encontramos. En el Modelo Cynefin, esta región se cataloga como un espacio arriesgado debido a la dificultad para evaluar las situaciones y establecer un enfoque de actuación. En tales circunstancias, es común que las personas interpreten las situaciones y tomen acciones basadas en preferencias personales. El mayor riesgo en el ámbito del desorden radica en actuar de manera incongruente con las necesidades para resolver problemas específicos. Un ejemplo de esto se observa en el campo del desarrollo de software, donde a menudo se emplea un enfoque secuencial y una planificación detallada basada en las mejores prácticas de la industria, lo cual es apropiado para situaciones simples. Sin embargo, cuando se aplica de manera incorrecta en situaciones complejas, como las que caen en el ámbito del desorden, puede resultar ineficaz. En estas circunstancias, la principal prioridad debería ser salir de la zona de desorden hacia un dominio más definido, y luego ajustar nuestras acciones de acuerdo con las demandas de ese nuevo dominio.

4.3. Elementos del marco de trabajo Scrum

Una cosa es expresar la necesidad de tener claridad en el trabajo que llevas a cabo para un producto, de revisar dicho trabajo con regularidad y de basar las modificaciones en los hallazgos resultantes de esta revisión. Otra cuestión es implementar todos estos aspectos de manera tangible y concreta. Esto es precisamente lo que convierte a Scrum en un marco de trabajo; presenta cinco eventos que se repiten para trabajar en tres artefactos, establece tres roles para brindar respaldo y abarca diversos principios y reglas que amalgaman todo en un conjunto coherente y cohesionado.

Figura 20. Marco de Trabajo Scrum.



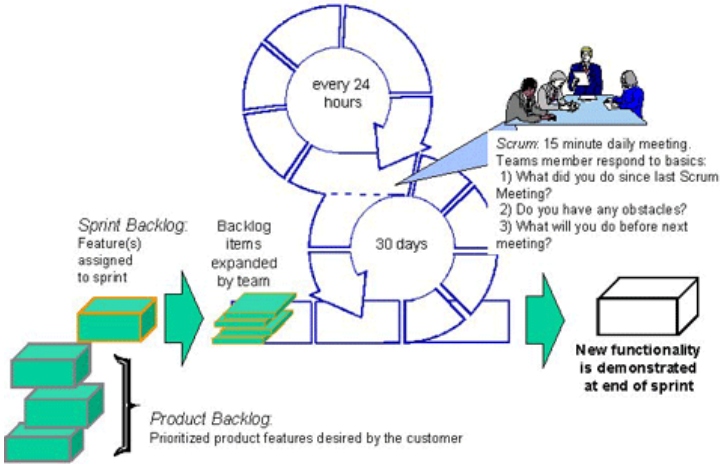
Nota. Scrum es un marco de trabajo para el Control del Proceso Empírico.

4.3.1 Justificación de Negocio

Es esencial para una organización comprender el concepto y el propósito de la Justificación de Negocio en el contexto de los proyectos Scrum. Antes de embarcarse en cualquier proyecto, resulta crucial que la organización realice una adecuada Justificación de Negocio y desarrolle una Declaración de la Visión del Proyecto que sea factible. Esto brinda la oportunidad a las partes clave o los responsables de tomar decisiones de evaluar si hay una necesidad real de que la empresa introduzca un cambio o un nuevo producto o servicio. Además, proporciona el razonamiento necesario para avanzar con el proyecto. Esta práctica también facilita al Product Owner la creación de un Backlog de Producto Priorizado, que tenga en cuenta las expectativas empresariales de la Alta Dirección y las partes interesadas relevantes (Overeem & Verwijs, 2020).

4.3.2 Sprint Backlogs

Figura 21. Sprint Backlogs.



Nota. El Sprint Backlog es la suma de el Objetivo del Sprint, los elementos del Product Backlog elegidos para el Sprint, más un plan de acción de cómo crear el Incremento de Producto.

Es uno de los tres artefactos esenciales de Scrum y se crea durante la reunión de Sprint Planning. Se trata de un plan elaborado por los Desarrolladores, diseñado para guiar su trabajo durante el Sprint.

Usualmente, el equipo descompone el trabajo en elementos conocidos como Elementos del Backlog del Sprint (SBI). Estos elementos pueden representar tareas específicas que el equipo debe llevar a cabo, componentes intermedios que se combinan para formar una entrega o cualquier otra unidad de trabajo que ayu-

de al equipo a comprender cómo alcanzar el Objetivo del Sprint dentro del propio Sprint (Garcia, 2020).

El compromiso del Sprint Backlog: El Objetivo del Sprint

Como sucede con todos los elementos en Scrum, el Sprint Backlog también incluye un compromiso específico. El compromiso asociado al Sprint Backlog es conocido como el Objetivo del Sprint, y se establece durante la reunión de Sprint Planning.

El Objetivo del Sprint representa el propósito central del Sprint. Aunque este compromiso es responsabilidad de los Desarrolladores, ofrece una cierta flexibilidad en cuanto a la naturaleza exacta del trabajo que se necesita para alcanzarlo.

Además, el Objetivo del Sprint contribuye a generar coherencia y concentración, fomentando así que el Equipo Scrum colabore en lugar de emprender iniciativas de manera aislada.

Ejemplo de Sprint Backlog

Aunque la guía Scrum no especifica un método específico para implementar este artefacto, considero que un enfoque recomendado podría ser la implementación de un tablero Kanban.

El tablero Kanban es una herramienta compuesta por columnas que representan el estado actual de una tarea y filas que reflejan diferentes categorías de actividades (por ejemplo, tareas desglosadas de las Historias de Usuario).

Cada tablero Kanban generalmente incluye al menos tres columnas con estados básicos:

"To Do" / Por hacer (donde se inicia una tarea).

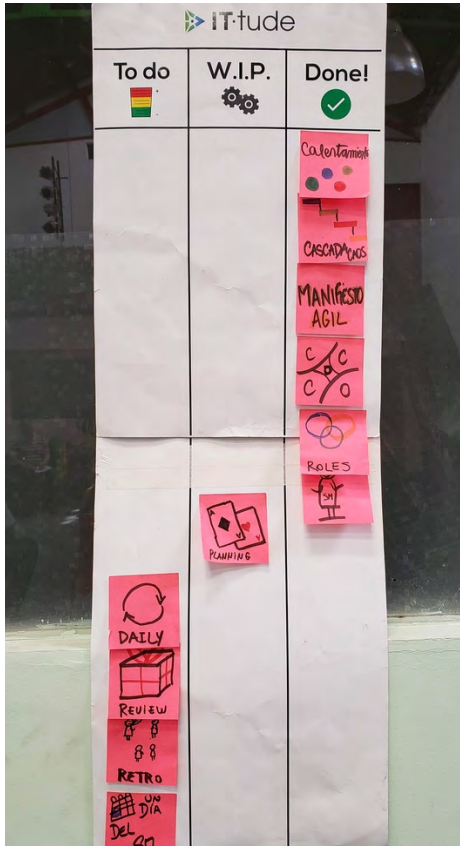
"W.I.P" / Trabajo en proceso (donde se trabaja activamente en la tarea).

"Done" / Terminado (donde se completa la tarea).

Aunque personalmente favorezco la opción de tener una representación física de este artefacto para fomentar la comunicación cara a cara, en la actualidad, existen soluciones digitales de tableros Kanban muy efectivas, como Trello, que son especialmente útiles para equipos que trabajan de forma remota.

Además, es posible agregar columnas adicionales a este tablero, como, por ejemplo "QA" (En etapa de pruebas).

Figura 22. Sprint Backlogs.



Nota. Ejemplo de implementación en un tablero Kanban.

Autoevaluación 14

1. **¿Cuál es el objetivo principal de la estrategia ágil en el desarrollo de software?**
    1. Desarrollar software con documentación detallada.
    2. Realizar proyectos de desarrollo a largo plazo.
    3. Entregar sistemas de software operativos de forma continua mediante iteraciones rápidas.
    4. Cumplir rigurosamente un conjunto de reglas predefinidas.
2. **¿Qué representa principalmente la metodología ágil en el desarrollo de software?**
    a. Un conjunto rígido de reglas para ejecutar proyectos.
    b. Una aproximación simplificada a la documentación de software.
    c. La resistencia a los cambios en el ciclo de vida del desarrollo.
    d. La entrega de software en grandes entregas finales.
3. **¿Cuáles son los tres objetivos esenciales que fomentan las metodologías ágiles en proyectos?**
    1. Establecer un cronograma estricto, minimizar costos y documentar exhaustivamente.
    2. Proporcionar valor, minimizar desperdicios y mejorar continuamente.
    3. Cumplir con los requisitos del cliente, evitar cambios y mantener una documentación detallada.
    4. Maximizar el alcance, minimizar los recursos y acelerar el proceso.

4. **¿Qué representa el Triángulo de Hierro en las metodologías ágiles?**
    a. Un conjunto de reglas estrictas para la gestión de proyectos.
    b. La priorización de la documentación sobre otros aspectos.
    c. Costo, Tiempo, Alcance y Calidad como elementos constantes.
    d. Un enfoque rígido en la entrega de proyectos.
5. **¿Cómo se relaciona el «Cono de Incertidumbre» con las metodologías ágiles?**
    1. Indica que la incertidumbre aumenta a medida que avanza el proyecto.
    2. Se utiliza para fijar objetivos claros en la gestión de proyectos.
    3. Sugiere que la documentación detallada es esencial para reducir la incertidumbre.
    4. Muestra que la gestión ágil minimiza la incertidumbre en el desarrollo de software.
6. **¿Cuál es el primer principio del Manifiesto Ágil?**
    a. Aprovechar el cambio como ventaja competitiva.
    b. Satisfacer al cliente mediante la entrega temprana y continua.
    c. Construir proyectos en torno a individuos motivados.
    d. Utilizar la comunicación cara a cara.
7. **¿Qué representa el segundo principio del Manifiesto Ágil?**
    1. Aprovechar el cambio como ventaja competitiva.
    2. Satisfacer al cliente mediante la entrega temprana y continua.
    3. Entregar valor frecuentemente.
    4. Cooperación negocio-desarrolladores durante todo el proyecto.

8. ¿Cuál es el tercer principio del Manifiesto Ágil?

1. Aprovechar el cambio como ventaja competitiva.
2. Satisfacer al cliente mediante la entrega temprana y continua.
3. Entregar valor frecuentemente.
4. Construir proyectos en torno a individuos motivados.

9. ¿Qué enfatiza el cuarto principio del Manifiesto Ágil?

1. La comunicación cara a cara es esencial para el éxito.
2. La colaboración entre negocio y desarrollo.
3. La importancia de mantener un desarrollo sostenible.
4. La simplicidad en la toma de decisiones.

10. ¿Cuál es el quinto principio del Manifiesto Ágil?

a. Utilizar la comunicación cara a cara.
b. La comunicación efectiva es fundamental.
c. Software funcionando como medida de progreso.
d. Promover y mantener un desarrollo sostenible.

11. ¿Qué se destaca en el sexto principio del Manifiesto Ágil?

1. La importancia de la comunicación a través de herramientas digitales.
2. La necesidad de documentar detalladamente todos los procesos.
3. La excelencia técnica como base para la agilidad.
4. La simplicidad en la toma de decisiones.

12. ¿Cuál es el séptimo principio del Manifiesto Ágil?

a. La colaboración negocio-desarrolladores durante todo el proyecto.
b. Software funcionando como medida de progreso.
c. La excelencia técnica mejora la agilidad.
d. La simplicidad es fundamental.

13.¿Qué se resalta en el octavo principio del Manifiesto Ágil?

a. La necesidad de desarrollar rápidamente sin preocuparse por la calidad.
b. La importancia de mantener un desarrollo sostenible.
c. La comunicación efectiva es fundamental.
d. La simplificación de los procesos de desarrollo.

14.¿Cuál es el noveno principio del Manifiesto Ágil?

a. La comunicación cara a cara es esencial para el éxito.
b. La simplicidad en la toma de decisiones.
c. La excelencia técnica mejora la agilidad.
d. Equipos auto-organizados para generar más valor.

15.¿Qué se destaca en el décimo principio del Manifiesto Ágil?

a. La importancia de mantener la documentación detallada.
b. La necesidad de automatizar todas las tareas.
c. La simplicidad en la toma de decisiones.
d. La forma más simple de abordar una situación.

16.¿Cuál es uno de los eventos principales en Scrum?

1. Sprint Review
2. Justificación de Negocio
3. Documentación de Proyecto
4. Reunión de Estrategia

17.¿Qué es esencial para una organización antes de embarcarse en un proyecto Scrum?

1. Reunión de Estrategia
2. Sprint Backlog
3. Justificación de Negocio
4. Backlog de Producto Priorizado

18.¿Cuál es uno de los artefactos clave en Scrum?

a. Reunión de Estrategia
b. Declaración de la Visión del Proyecto

c. Sprint Backlog
d. Justificación de Negocio

19. ¿Cuál de los siguientes no es un evento en Scrum?

1. Sprint Review
2. Sprint Backlog
3. Daily Standup
4. Sprint Planning

20. ¿Qué rol es responsable de crear un Backlog de Producto Priorizado?

a. Scrum Master
b. Product Owner
c. Desarrollador
d. Alta Dirección

4.4 Eventos de Scrum

Scrum incluye cuatro reuniones distintivas, cada una con un objetivo específico: la planificación del sprint, la reunión diaria de Scrum, la revisión del sprint y la retrospectiva del sprint. La reunión diaria de Scrum se realiza a diario por la mañana, mientras que las otras reuniones tienen lugar una vez en cada iteración del ciclo.

La palabra "sprint", al igual que "scrum", proviene del ámbito deportivo, y es una metáfora sumamente adecuada. No se trata de carreras de larga distancia, sino de esfuerzos intensos con un objetivo claro durante un período breve.

La duración óptima de un sprint dependerá de las circunstancias. Aunque algunas opiniones favorezcan sprints muy cortos, también es posible establecerlos de un mes. De hecho, la duración puede variar, incluso en el mismo proyecto. Esto depende del propósito de la iteración. Por ejemplo, si es necesario presentar una nueva funcionalidad en un evento dentro de dos semanas, se podría ajustar la duración del sprint aunque el equipo generalmente opere con sprints de 3 semanas. La clave radica en comprender que estos sprints subdividen el desarrollo en pequeñas metas. Se trata de una técnica para gestionar el avance constante. El período de tiempo debe ser lo bastante corto como para permitir una evaluación frecuente de nuestro progreso en la dirección correcta y determinar si se requieren ajustes (Domínguez Coutiño, 2012).

Figura 23. Eventos de Scrum.

4.4.1 Scrum Diario

El Daily Scrum, que es uno de los cinco eventos esenciales en Scrum, tiene una duración de 15 minutos y está diseñado para que los Desarrolladores se sincronicen entre sí.

Esta reunión diaria se efectúa en un horario y locación constantes, con el propósito de simplificar la situación. Durante ella, se busca lograr transparencia e inspeccionar el trabajo realizado, a fin de crear una oportunidad para ajustarse de cara al día siguiente.

¿Cuál es el objetivo de la Daily Scrum?

El propósito de la Daily Scrum es examinar el avance en dirección al Objetivo del Sprint y ajustar el Sprint Backlog según sea necesario. Su enfoque está en la sincronización de los Desarrolladores, planificando las labores para las próximas 24 horas.

Mediante esto, se optimiza la colaboración y el rendimiento del equipo al inspeccionar el progreso desde la última Reunión Daily Scrum y al proyectar el trabajo que resta en el Sprint.

Los Desarrolladores utilizan la Daily Scrum para evaluar su avance en relación con el Sprint Goal, y para evaluar cómo se desarrolla esa tendencia en términos de culminar las tareas del Sprint actual.

Dado que esta reunión posee un tiempo limitado de 15 minutos, cada individuo debe resumir de manera concisa lo que considera más relevante para lograr la sincronización con sus colegas.

¿Quién debe asistir al Daily Scrum?

La Daily Scrum es un evento destinado a los Desarrolladores. En el caso de que el Product Owner o el Scrum Master estén involucrados en la ejecución de elementos del Sprint Backlog, se integran como Desarrolladores. Si hay otras personas presentes, el Scrum Master se asegura de que no interfieran con el propósito de la reunión

.

Individuos externos al equipo de Desarrolladores pueden asistir a esta reunión diaria de sincronización si son invitados por ellos. Los Desarrolladores podrían considerar que en Scrum existe un principio de transparencia. Sin embargo, la presencia de personas ajenas puede influir en la dinámica de la reunión. De todas formas, solamente los Desarrolladores participan activamente en la sesión. Esto abarca tanto al Product Owner como a cualquier otra persona que no forme parte del equipo de Desarrolladores.

Beneficios del Daily Scrum

Optimización: La reunión diaria del Daily Scrum maximiza las probabilidades de que los Desarrolladores alcancen el Sprint Goal.

Reducción de tiempo perdido: Los Desarrolladores exponen los obstáculos diariamente, minimizando la pérdida de tiempo.

Mejora de la coordinación: Facilita la identificación de oportunidades para coordinarse, ya que todos están al tanto de las tareas de los demás.

Fomento del intercambio de conocimientos: El Daily Scrum señala áreas de desconocimiento y posibilita el enriquecimiento del equipo.

Consolidación de la cultura: Esto se logra mediante rituales compartidos y la participación activa.

Incremento de la productividad: Proporciona eficiencia al proceso, resultando en un aumento de la productividad.

Buenas prácticas del DAILY SCRUM

Los equipos Scrum altamente efectivos utilizan esta ocasión para actualizar su pizarra Kanban, también reconocida como "pizarra Scrum".

El equipo actualizará el estado más reciente de las tareas en la pizarra, con el propósito de hacer visibles los avances y obstáculos, y así determinar cómo pueden cooperar entre sí para concluir PRIORITARIAMENTE los elementos que contribuyen en mayor medida al Sprint Goal.

También pueden emplear parte del tiempo durante esta reunión de sincronización para actualizar su gráfico de Burndown, con el propósito de evaluar qué tan cerca o lejos están de alcanzar el Sprint Goal y poder ajustar su plan con base en esa información.

En cada iteración, se generará un nuevo gráfico, y a lo largo del proyecto, se pueden identificar tendencias y los principales obstáculos que contribuyen a la demora del proyecto.

Es esencial considerar que cuando los miembros del equipo exponen sus impedimentos, es natural que surja el deseo de profundizar en ellos y buscar posibles soluciones.

Recuerda, el Daily Scrum es una reunión destinada a replanificar y tomar decisiones, no para resolver problemas directa-

mente. Para abordar esos problemas, los involucrados pueden acordar reunirse después de la reunión diaria o en un horario específico del día para encontrar soluciones, sin consumir el tiempo de los demás. En la siguiente reunión diaria, podrán informar sobre la decisión que tomaron o cómo resolvieron el problema (Garcia, 2020).

4.4.2 Revisión de Sprint

El término "sprint review" se refiere a la reunión llevada a cabo con la intención de examinar los logros alcanzados por el equipo Scrum después de un sprint. Esto posibilita, asimismo, la evaluación del avance del desarrollo en relación con el logro del objetivo establecido.

Aunque comúnmente se piensa que el sprint review se centra en la presentación del producto, esta representa únicamente una porción reducida del evento en sí. Exploraremos con mayor detalle este aspecto en futuras secciones al analizar cómo se desarrolla la reunión (Aguirre, 2022).

Sprint review: objetivos

Dentro del marco de Scrum, se realiza una evaluación del proyecto con respecto al propósito del sprint, el cual ha sido establecido durante la sesión de planificación del sprint.

Al concluir la revisión del sprint, los objetivos principales que se buscan son los siguientes:

- Examinar el avance alcanzado en el incremento.
- Ajustar el product backlog en caso de ser requerido.

Figura 24. Sprint review.

Nota. La figura muestra cuando el equipo revisa los elementos de trabajo que completaron durante el sprint o la iteración.

4.4.3 Refinamiento del Product Backlog

Dentro de Scrum, el proceso de mejora del Backlog es una actividad constante en la cual el Product Owner y el Equipo de Desarrollo trabajan juntos para garantizar que los elementos en el Product Backlog:

- Son comprensibles de manera uniforme por todos los participantes (comprensión colectiva),

- Cuentan con una evaluación de su nivel de complejidad y trabajo necesario (relativos) para su ejecución, y
- Se organizan en función de su prioridad en relación con el valor empresarial y el esfuerzo involucrado.

En resumen, el proceso de refinar el backlog implica establecer un conocimiento compartido acerca de las funcionalidades que el producto abarcará y aquellas que no, evaluar el grado de esfuerzo necesario para llevar a cabo su implementación, y determinar el orden secuencial en que se abordarán estos aspectos.

¿Por qué es importante el refinamiento en el backlog?

Examinemos nuevamente los propósitos del proceso de mejora del backlog expuestos en la sección previa.

Si no se establece un entendimiento común, existe la posibilidad de implementar algo erróneo, malgastar esfuerzos y verse obligado a retrabajar la implementación para ejecutarlo de manera adecuada.

La falta de estimación para cada elemento impide considerar el "costo" de los elementos, con el riesgo de sobrevalorar los elementos de alto valor y gran costo, mientras que se subestiman aquellos de menor valor y costo.

Si no se organiza el producto backlog en un orden descendente de prioridad, existe la posibilidad de enfocarse en elementos de menor importancia y pasar por alto aquellos que son esenciales.

Otras razones por las que es importante el refinamiento del backlog:

Incrementa la efectividad de la sesión de Planificación del Sprint debido a que gran parte de las incógnitas ya han sido resueltas.

Preserva la claridad, orden y pertinencia del Backlog del Producto, evitando que se abrume en una lista de tareas en constante crecimiento.

Capitaliza los beneficios de trabajar en equipo para elaborar con mayor profundidad las historias de usuario y los problemas identificados.

Fomenta la construcción de un conocimiento mutuo tanto en el Equipo Scrum como en los involucrados externos.

¿Cuánto tiempo debe durar el refinamiento del backlog?

La Guía de Scrum no establece una duración específica para el proceso de Refinamiento del Backlog. Simplemente menciona que generalmente no debe exceder el 10% de la capacidad del equipo de desarrollo. Dado que el Product Owner no forma parte del Equipo de Desarrollo, puede invertir el tiempo necesario y solicitar asistencia de otros miembros del Equipo Scrum. La conversión de una historia en un Spike es una estrategia para dejar esto explícito y prevenir que esa cuota del 10% se consuma (Work, 2023).

4.4.4 Conclusión

En este capítulo, hemos explorado una amplia gama de temas esenciales en el desarrollo de software y en la adopción de enfoques ágiles. Desde la Especificación del software hasta el Refinamiento del Product Backlog, hemos abordado áreas críticas como el Diseño e implementación del software, la Validación de Software y la Evolución de Software.

Hemos comprendido cómo la Especificación del software desempeña un papel fundamental al recopilar y analizar requisitos, y cómo el Diseño e implementación del software transforma estos requisitos en soluciones técnicas concretas. Exploramos los pilares de los enfoques ágiles, aprendiendo del Manifiesto y Principios Ágiles, y profundizamos en el marco de trabajo Scrum, donde se desglosaron los Valores, Pilares, Roles y Elementos esenciales.

Además, se subrayó la importancia de la Justificación de Negocio y se exploraron los Artefactos y Eventos de Scrum, como los Sprint Backlogs y las Reuniones de Revisión, que permiten una adaptación constante. En resumen, este capítulo ofrece una visión completa para diseñar, desarrollar y evolucionar software de manera eficaz, resaltando la relevancia de los enfoques ágiles y Scrum en la consecución de la agilidad, colaboración y entrega continua de valor en el desarrollo de productos.

Autoevaluación 15

1. **¿Qué evento de Scrum se lleva a cabo al final de cada Sprint y se centra en mostrar lo que se ha logrado durante ese Sprint?**
    1. Sprint Planning
    2. Sprint Backlog
    3. Daily Standup
    4. Sprint Review
2. **¿Cuál de los siguientes eventos de Scrum es una reunión diaria de seguimiento de 15 minutos?**
    a. Sprint Planning
    b. Sprint Review
    c. Daily Standup
    d. Sprint Retrospective
3. **¿Cuál es el propósito principal de la reunión de Sprint Planning en Scrum?**
    a. Revisar el trabajo completado en el Sprint anterior.
    b. Definir los roles y responsabilidades del equipo.
    c. Seleccionar las tareas que se abordarán en el próximo Sprint.
    d. Evaluar el rendimiento del Scrum Master.
4. **¿Cuál de los siguientes eventos de Scrum se enfoca en la mejora continua y la adaptación de procesos?**
    1. Sprint Planning
    2. Sprint Review
    3. Daily Standup
    4. Sprint Retrospective

5. **¿Qué evento de Scrum se refiere a la reunión donde el equipo de Scrum analiza su trabajo y busca oportunidades para mejorar?**
    a. Sprint Planning
    b. Sprint Review
    c. Daily Standup
    d. Sprint Retrospective
6. **¿Cuál es el otro nombre comúnmente utilizado para la reunión diaria de Scrum?**
    1. Sprint Planning
    2. Daily Review
    3. Daily Standup
    4. Sprint Retrospective
7. **¿Cuál es el propósito principal de la reunión diaria de Scrum?**
    a. Tomar decisiones importantes sobre el proyecto.
    b. Revisar y actualizar el Backlog de Producto.
    c. Identificar problemas y coordinar el trabajo del equipo.
    d. Realizar una revisión detallada de los entregables del Sprint.
8. **¿Cuál es la duración recomendada para la reunión diaria de Scrum?**
    a. 1 hora
    b. 30 minutos
    c. 15 minutos
    d. 2 horas

9. ¿Qué pregunta típicamente se hace durante la reunión diaria de Scrum?

1. ¿Qué hizo usted durante el fin de semana?
2. ¿Qué obstáculos enfrentó ayer?
3. ¿Cuándo se completará el proyecto?
4. ¿Qué se espera del próximo Sprint?

10. ¿Cuál es el objetivo principal de limitar la duración de la reunión diaria de Scrum a 15 minutos?

1. Para que el equipo tenga tiempo para socializar.
2. Para evitar discusiones largas y centrarse en la coordinación.
3. Para cumplir con una regla arbitraria de tiempo.
4. Para dar tiempo a los gerentes para tomar decisiones.

11. ¿Qué es el «Refinamiento del Product Backlog" en Scrum?

a. Una reunión diaria para revisar el progreso del equipo.
b. Un evento en el que se crea el Product Backlog.
c. El proceso de mejorar y aclarar elementos del Product Backlog.
d. Una técnica para priorizar las historias de usuario.

12. ¿Quién es responsable de liderar el proceso de refinamiento del Product Backlog?

a. El Scrum Master.
b. El Product Owner.
c. El equipo de desarrollo.
d. La alta dirección de la empresa.

13. ¿Cuál es uno de los objetivos principales del refinamiento del Product Backlog?

1. Crear un plan detallado para el próximo Sprint.
2. Priorizar los elementos del Product Backlog en orden alfabético.
3. Asegurarse de que los elementos del Product Backlog sean comprensibles y estimables.
4. Evaluar el desempeño del Scrum Master.

14. ¿Cuándo suele llevarse a cabo el refinamiento del Product Backlog?

1. Al final de cada Sprint.
2. Durante la reunión de planificación del Sprint.
3. Durante la reunión diaria de Scrum.
4. De manera continua a lo largo del Sprint.

15. ¿Qué beneficio se obtiene al refinar el Product Backlog de forma continua durante el Sprint?

1. Se evitan las reuniones innecesarias.
2. Se facilita la identificación de nuevos elementos para el Sprint.
3. Se mantiene una visión clara de los elementos a trabajar en futuros Sprints.
4. Se reduce la necesidad de contar con un Scrum Master.

Referencias

Aguirre, M.F. (2022, 21 de 01). ¿Qué es y cómo llevar a cabo un sprint review? *Appvizer.* https://www.appvizer.es/revista/organizacion-planificacion/gestion-proyectos/sprint-review

Domínguez Coutiño, L.A. (2012). *Análisis de Sistemas de Información.* Red Tercer Milenio

Dremyn, P. (2020, 12 de octubre). El enfoque ágil y porqué debes usarlo. *Linkedin*. https://es.linkedin.com/pulse/el-enfoque-ágil-y-porqué-debes-usarlo-paul-dremyn

Garcia, M. (2020). ¿Qué es el sprint backlog? *IT.tude*. https://ittude-agile.com/b/scrum/que-es-el-sprint-backlog/

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

León Yacelga, A.R., Acosta Espinoza, J.L., & Díaz Vásquez, R.A. (2021). Aplicación de la metodología incremental en el desarrollo de sistemas de información. *Universidad Y Sociedad, 13*(5), 175-182. https://rus.ucf.edu.cu/index.php/rus/article/view/2223

Martins, J. (2023, 19 de junio). Scrum: conceptos clave y cómo se aplica en la gestión de proyectos. *Asana*. https://asana.com/es/resources/what-is-scrum

Overeem, B., & Verwijs, C. (2020). *Scrum: Un Marco de Trabajo para Reducir el Riesgo y Entregar Valor Antes.* The Liberators.

SENTRIO. (2021, 13 de octubre). *Metodologías Agile: los 4 valores y 12 principios del 'Manifiesto Ágil'*. https://acortar.link/Eko9Zt

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

Sulbarán, H. (2014, 24 de septiembre). *Paradigmas en el desarrollo de software* [Blog]. https://acortar.link/VUTmOL

Sulbarán, I.. (2023, 27 de abril). Interfaz de usuario (ui): ejemplos y tipos. *Tiffin University*. https://global.tiffin.edu/noticias/interfaz-de-usuario-ui-ejemplos-y-tipos

§

Software Specification and Agile Approaches
Especificação de software e abordagens ágeis

Mónica Elizabeth Páez Padilla
Instituto de Educación Superior Nelson Torres | Cayambe | Ecuador
https://orcid.org/0009-0006-1030-1394
monica.paez@intsuperior.edu.ec
Diego Javier Portilla Martínez
Investigador independiente | Cayambe | Ecuador
https://orcid.org/0009-0008-7295-6227
dijapm@gmail.com

Abstract
In this chapter, fundamental issues ranging from software definition to the implementation of agile approaches are discussed, with a particular focus on the Scrum framework. It highlights the relevance of relying not only on methodologies and processes to produce high quality software, but also on accurate specification and the application of agile practices. The text focuses on the software specification stage, including design, implementation, validation and evolution. It also explores agile approaches to software development, detailing their suitability and the Agile Manifesto. Next, the key components of Scrum are discussed, including roles, responsibilities, interaction, business justification, and artifacts. Finally, it delves into the essential Scrum events, including the Daily Scrum, the retrospective and the refinement of the Product Backlog.
Keywords: Scrum, Product Backlog, Agile Manifesto, software.

Resumo
Este capítulo aborda questões fundamentais que vão desde a definição de software até a implementação de abordagens ágeis, com foco especial na estrutura Scrum. Ele destaca a relevância de contar não apenas com metodologias e processos para produzir software de alta qualidade, mas também com uma especificação precisa e a aplicação de práticas ágeis. O texto concentra-se no estágio de especificação do software, incluindo design, implementação, validação e evolução. Ele também explora abordagens ágeis para o desenvolvimento de software, detalhando sua adequação e o Manifesto Ágil. Em seguida, são discutidos os principais componentes do Scrum, incluindo funções, responsabilidades, interação, justificativa comercial e artefatos. Por fim, são abordados os eventos essenciais do Scrum, incluindo o Diário do Scrum, a retrospectiva e o refinamento do Backlog do Produto.
Palavras-chave: Scrum, Product Backlog, Manifesto Ágil, software.

ANEXOS

Anexo 1

Solucionario Evaluación 1

Respuestas

1. a
2. b
3. c
4. c
5. b
6. c
7. 1
8. b
9. 2
10. 2
11. 2
12. 2
13. c
14. 1
15. b
16. 2
17. 4
18. b
19. 3
20. 2

Anexo 2

Solucionario Evaluación 2

Respuestas

1. b
2. a
3. b
4. b
5. b
6. b
7. c
8. b
9. b
10. b

Anexo 3

Solucionario Evaluación 3

Respuestas

1. a
2. c
3. b
4. b
5. a
6. c
7. a
8. c
9. c
10. c
11. a
12. b
13. a
14. b
15. b

Anexo 4

Solucionario Evaluación 4

Respuestas

1. 3
2. 2
3. a
4. d
5. b
6. 2
7. c
8. 3
9. 2
10. a
11. 3
12. b
13. 1
14. 1
15. 2
16. c
17. b
18. 1
19. 3
20. C

Anexo 5

Solucionario Evaluación 5

Respuestas

1. a
2. 2
3. a
4. 3
5. b
6. 1
7. 2
8. 3
9. b
10. c
11. c
12. c
13. c
14. c
15. c

Anexo 6

Solucionario Evaluación 6

Respuestas

1. 2
2. b
3. 1
4. c
5. 2
6. b
7. 2
8. b
9. 3
10. c
11. 2.
12. b
13. 2
14. 3
15. C

Anexo 7

Solucionario Evaluación 7

Respuestas

1. c
2. d
3. 2
4. 3
5. c
6. 3
7. d
8. a
9. 3
10. C
11. 2
12. c
13. b
14. 3
15. 3

Anexo 8

Solucionario Evaluación 8

Respuestas:

1. c
2. b
3. c
4. d
5. a
6. c
7. b
8. b
9. c
10. c
11. b
12. b
13. b
14. b
15. b
16. d
17. b
18. b
19. a
20. b

Anexo 9

Solucionario Evaluación 9

Respuestas:

1. a
2. c
3. c
4. c
5. b
6. a
7. c
8. a
9. c
10. a
11. b
12. c
13. c
14. c
15. a

Anexo 10

Solucionario Evaluación 10

Respuestas:

1. b
2. 3
3. 2
4. b
5. 3
6. 3
7. a
8. 1
9. 1
10. 3
11. 3
12. b
13. 2
14. 2
15. b

Anexo 11

Solucionario Evaluación 11

Respuestas:

1. b
2. a
3. 2
4. b
5. 3
6. 3
7. b
8. 1
9. a
10. 3
11. c
12. 3
13. b
14. 1
15. b

Anexo 12

Solucionario Evaluación 12

Respuestas:

1. c
2. 2
3. b.
4. 2
5. d
6. 1
7. 3
8. c.
9. 4
10. c
11. 3
12. 1
13. b
14. 3
15. A

Anexo 13

Solucionario Evaluación 13

Respuestas:

1. c.
2. 3
3. 3
4. c.
5. 3
6. b.
7. 2
8. c.
9. b.
10. 1
11. 3
12. c.
13. c.
14. d.
15. a.
16. c.
17. 3
18. 4
19. d.
20. 2

Anexo 14

Solucionario Evaluación 14

Respuestas:

1. 3
2. b
3. 2
4. c
5. 1
6. b
7. 1
8. 3
9. 2
10. d
11. 1
12. b
13. b
14. c
15. d
16. 1
17. 3
18. c

19. 2
20. b

Anexo 15

Solucionario Evaluación 15

Respuestas:

1. 4
2. c
3. c
4. 4
5. d
6. 3
7. c.
8. c
9. 2
10. 2
11. c.
12. b
13. 3
14. 4
15. 3

Religación

Press

Ideas desde el Sur Global

Religación Press
Ideas desde el Sur Global

Instituto Nelson Torres

ISBN: 978-9942-642-43-1