INSTITUTO SUPERIOR TECNOLÓGICO TENA
Tecnología, Innovación y Desarrollo

DESARROLLO DE SOFTWARE

Instrumento para facilitar el proceso de enseñanza-aprendizaje de la asignatura

GUÍA GENERAL DE ESTUDIO DE LA ASIGNATURA 20240003

CALIDAD DE SOFTWARE

Período académico
Quinto

Abril - 2024

Ing. MARTHA JANINA DUARTE MORA, MG.

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA – CALIDAD DE SOFTWARE

INSTITUTO SUPERIOR TECNOLÓGICO TENA

Carrera de Tecnología Superior en Desarrollo de Software

ISTT DSW Primera Edición – Tena, abril 2024

SIN ISBN
Instituto Superior Tecnológico Tena
Km. 1 1/2 Vía Tena - Archidona
Tena, Ecuador

AUTOR(ES) - REDACCIÓN Y FORMULACIÓN DE CONTENIDOS

Mg. Martha Janina Duarte Mora
Profesor del Instituto Superior Tecnológico Tena

REVISIÓN DE PARES

Lcdo. Segundo Calisto Rochina Chileno
Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango
Ing. Agustín Gonzalo Guanipatin Ramirez

Comisión de revisión técnica de guías de estudio del Instituto Superior Tecnológico Tena

APROBACIÓN

Mg. Danilo Alexander Zamora Núñez
Coordinador de Investigación, Desarrollo Tecnológico e Innovación

Impreso y hecho en Ecuador.

TABLA DE CONTENIDO

GUIA GENERAL DE ESTUDIO DE LA ASIGNATURA

DATOS GENERALES DE LA ASIGNATURA							
Carrera	Tecnología Superior en Desarrollo de Software			**Nombre asignatura**		Calidad de Software	
Modalidad	Presencial			**Campo de Formación**		Adaptación e Innovación Tecnológica	
Jornada	Matutina y Nocturna			**Unidad de Organización Curricular**		Profesional	
Período académico	Quinto			**Código de la asignatura**		DS525	
Distribución de horas en las actividades de aprendizaje				**N° Total de horas de la asignatura**		148	
N° de horas Docencia	**72**	**N° de horas Aprendizaje Práctico Experimental**				**N° de horas Autónomo**	**40**
		En contacto con docente	**8**	**Autónomo**	**28**		

PRERREQUISITOS Y CORREQUISITOS			
Prerrequisitos de la asignatura		**Correquisitos de la asignatura**	
Asignatura	**Código**	**Asignatura**	**Código**
Desarrollo de Aplicaciones Móviles	DS422		

DESCRIPCIÓN DE LA ASIGNATURA

La asignatura, tiene como propósito desarrollar en el estudiante la capacidad de aplicar los conceptos, métodos, técnicas y estándares de calidad de software.

OBJETIVO GENERAL

Contribuir a la formación de recursos humanos calificados para aplicar conceptos, metodologías y herramientas necesarias para el desarrollo e implementación de aplicaciones de software de calidad.

CONTRIBUCIÓN DE LOS RESULTADOS DE APRENDIZAJE DE LA ASIGNATURA AL PERFIL DE EGRESO DE LA CARRERA

Resultados de aprendizaje de la asignatura	Resultados de aprendizaje del perfil de egreso de la carrera	Contribución (alta – media – baja)
• Aplica estándares de calidad en el desarrollo y evaluación de soluciones informáticas.	• Realiza pruebas que garanticen la calidad del software.	
• Comprende los elementos principales del sistema de calidad basado en la norma ISO 9000:2000.		Alta
• Comprende las principales características de los procesos del ciclo de vida del estándar ISO/IEC 12207.	• Aplica estándares de calidad en el desarrollo y evaluación de soluciones informáticas.	Alta
• Comprende las principales características de los procesos del ciclo de vida del estándar ISO/IEC 15504.		
• Comprende el ámbito del proceso y técnicas de pruebas de software.		

CONTENIDOS DE LA ASIGNATURA (descripción mínima de contenidos de la asignatura)

Unidad 1: Introducción a la Calidad, Métricas

- Fundamentos de la calidad
- Definiciones de calidad.
- Métricas de calidad.
- Modelos de Métricas
- Clasificación de las Métricas
- Medición de los costos de la mala calidad.
- Evolución del enfoque de la calidad.

Unidad 2: Gestión y enfoques de la calidad

- Principios de la gestión de la calidad total
- Cliente, empleados, procesos,
- Sistema integrado, enfoque estratégico, Mejora continua, Toma de decisiones basada en información, Comunicaciones.
- Autores de la calidad
- Enfoques de calidad
- Examen del Primer Parcial
- Estándares de Calidad-Modelos y normas de calidad
- Familia ISO 9000
- - ISO 9001
- - ISO 9004
- Familia de ISO 25000
  - ISO/IEC 2500n.
  - ISO/IEC 2501n
  - ISO/IEC 2502n
  - ISO/IES 12207
  - ISO/IEC 29110

Unidad 3: Diagramas UML y Seguridad de Software

- Métricas para Diagramas UML
- Introducción
- Tipos de UML
- Plataformas para diagramas UML
- Práctica en la plataforma Lucidchart
- Pruebas de Software
- Técnicas de pruebas
- Procesos de pruebas
- Herramientas de automatización
- Seguridad de software
- Tipos de seguridad de software
- Seguridad en el ciclo de desarrollo de software
- Confiabilidad, disponibilidad e integridad en el Desarrollo de Software.
- Gestión de pruebas completas de seguridad de software.
- Examen del segundo parcial

ESTRATEGIAS METODOLÓGICAS Y RECURSOS DIDÁCTICOS

ESTRATEGIAS METODOLÓGICAS	HABILIDADES BLANDAS	FINALIDAD
Activas para la enseñanza y aprendizaje	**Valores vinculados a la autonomía del sujeto**: confianza, crítica y autocrítica, honestidad, integridad	• Generar confianza/ Promover el pensamiento crítico. • Permite a los estudiantes cumplir un rol activo dentro de su formación. • Construye una sociedad participante.

Aprendizaje y trabajo cooperativo	**Valores elementales de convivencia y civilidad:** crítica y autocrítica, tolerancia, empatía, respeto, justicia, lealtad, paciencia	• Promover un ambiente de colaboración/ trabajo en equipo/ Saber escuchar/Promover el pensamiento crítico/ fomentar el liderazgo/ adaptabilidad. • Mantener una comunicación abierta con el equipo/ tolerancia a los errores, aceptar y aprender de las críticas. • Fomentar el sentido de pertenencia
Aprendizaje individual	**Valores vinculados a la autonomía del sujeto:** responsabilidad, honestidad, integridad, efectividad, autonomía	• Facilitar la asimilación del contenido por parte del estudiante/ Plantear preguntas para promover la comunicación efectiva /Promover el pensamiento crítico • Lectura comprensiva para fijar contenidos/ Promover el pensamiento crítico

RECURSOS DIDÁCTICOS	
MATERIALES CONVENCIONALES	*Material impreso: libros, folletos, fotocopias, periódicos, etc.*
	Tableros didácticos: pizarra
MATERIALES AUDIOVISUALES	*Imágenes fijas proyectables (fotos): diapositivas y fotografías.*
	Materiales audiovisuales (video): películas y videos
NUEVAS TECNOLOGÍAS	*Programas informáticos: procesador de palabras, hojas de cálculo, presentaciones*
	Servicios telemáticos: páginas web, plataforma EVA, correo electrónico, chats, CANVA, GENEALLY

BIBLIOGRAFÍA

Bibliografía Básica de la Asignatura:	**Físico**	**Digital**
Mario, G. (2016). *Calidad de sistemas de información*, 4ta Edición. Editorial Adriana Gutiérrez, España. ISBN: 978-958-778-156-4. **Código de inventario en Biblioteca ISTT-DS-0111**	X	
Guillermo, P. (2016). Calidad en el Desarrollo de Software, 2da Edición. Editorial Alfa Omega, Argentina. ISBN: 978-958-762-916-3. **Código de inventario en Biblioteca ISTT-DS-0084**	X	
Bibliografía de consulta de la Asignatura:	**Físico**	**Digital**
Silvia A. & Coral C. (2022). *Calidad y sostenibilidad de sistemas de información en la práctica*, 1ra Edición, Editorial RAMA S.A, España. ISBN: 9788418971815, 8418971819. https://www.google.com.ec/books/edition/Calidad_y_sostenibilidad_de_sistemas_de/E864EAAAQBAJ?hl=es-419&gbpv=0		X
Coral C. (2010). *Calidad Del Producto Y Proceso Software*, 1ra Edición. Editorial RAMA. ISBN: 978- 9788478979615 https://www.google.com.ec/books/edition/Calidad_del_producto_y_proceso_software/M4h1WAvbgqQC?hl=es-419&gbpv=0		X

DESCRIPTIVA DE LAS COMPETENCIAS DE LA GUÍA DE CALIDAD DE SOFTWARE

La asignatura "Calidad de Software" desarrolla en los estudiantes competencias que les permiten adquirir conocimientos teóricos y prácticos relacionados con la calidad en el desarrollo y gestión de software. Estas competencias están alineadas con la formación integral en el ámbito tecnológico y abordan aspectos como la implementación de métricas, la gestión de calidad, el uso de diagramas UML y la seguridad en software. A continuación, se describen las competencias generales de la guía y las específicas correspondientes a cada unidad.

Competencias Generales

Los estudiantes estarán en capacidad de comprender y aplicar principios, enfoques y herramientas relacionados con la calidad de software. Esto incluye la capacidad de identificar métricas clave, gestionar proyectos desde la perspectiva de la calidad, y garantizar la seguridad del software durante todo el ciclo de vida del desarrollo. Además, se fomenta la habilidad para trabajar en equipos multidisciplinarios, comunicarse efectivamente y tomar decisiones éticas y responsables en contextos profesionales.

Competencias Específicas

Unidad 1: Introducción a la Calidad, Métricas

En esta unidad, los estudiantes desarrollarán competencias para comprender los conceptos básicos de calidad en el desarrollo de software. Esto incluye la identificación y definición de métricas que permitan medir el desempeño y la eficiencia del software. Los estudiantes serán capaces de analizar indicadores clave y establecer criterios objetivos para evaluar la calidad, asegurando que los productos tecnológicos cumplan con estándares internacionales. Además, aprenderán a implementar herramientas y técnicas para recopilar, analizar y utilizar datos de manera efectiva en el contexto del control y aseguramiento de la calidad.

Unidad 2: Gestión y Enfoques de la Calidad

Los estudiantes adquirirán competencias en la planificación y gestión de proyectos de software desde una perspectiva orientada a la calidad. Esto incluye el aprendizaje de enfoques modernos, como la integración de metodologías ágiles y tradicionales en el aseguramiento de la calidad. También serán capaces de identificar riesgos asociados al desarrollo de software y proponer estrategias para mitigarlos. Por otro lado, los estudiantes comprenderán la importancia de las auditorías y evaluaciones periódicas como herramientas clave para garantizar que los proyectos cumplan con los objetivos establecidos.

Unidad 3: Diagramas UML y Seguridad de Software

En esta unidad, los estudiantes desarrollarán competencias en el uso de diagramas UML como herramienta para modelar sistemas y garantizar una comprensión clara de los requisitos y procesos. Serán capaces de interpretar y crear diagramas estructurales y de comportamiento, asegurando la alineación entre el diseño y los objetivos de calidad. Adicionalmente, los estudiantes obtendrán habilidades para evaluar y mejorar la seguridad en software, identificando vulnerabilidades y aplicando prácticas recomendadas para proteger los sistemas frente a amenazas internas y externas.

UNIDAD 1: Introducción a la Calidad, Métricas

- Fundamentos de la calidad
- Definiciones de calidad.
- Métricas de calidad.
- Modelos de Métricas
- Clasificación de las Métricas
- Medición de los costos de la mala calidad.
- Evolución del enfoque de la calidad.

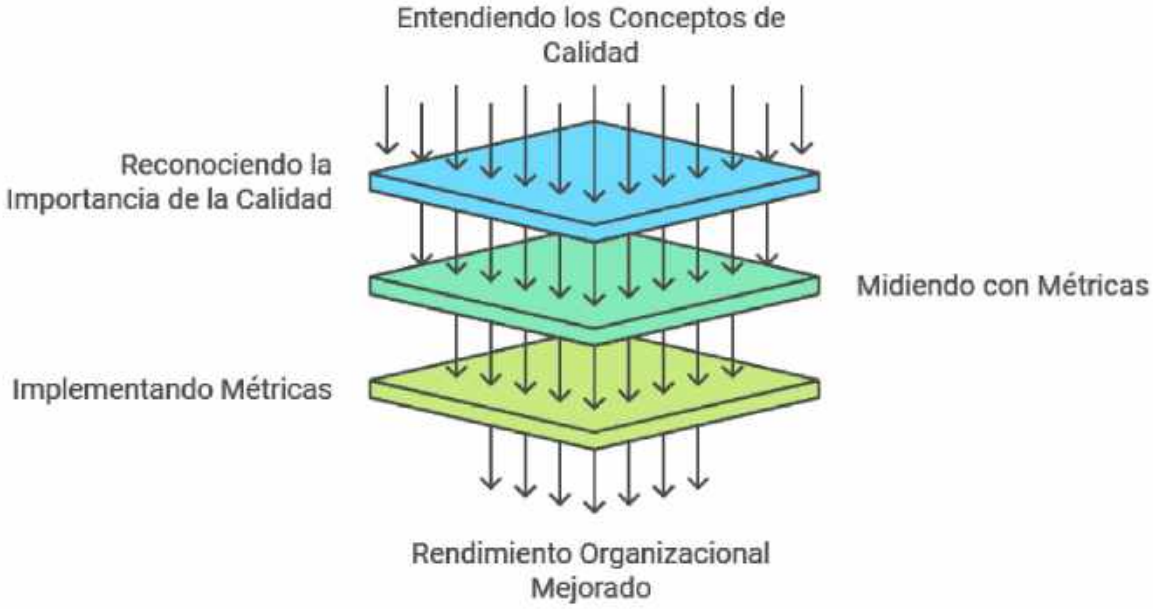
UNIDAD 1

Resultado de Aprendizaje

1. **Comprensión de conceptos fundamentales:**
   El estudiante será capaz de definir y explicar los conceptos básicos relacionados con la calidad del software, incluyendo sus dimensiones principales, como funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad.
2. **Identificación de métricas de calidad:**
   El estudiante podrá seleccionar e interpretar métricas clave para medir y evaluar la calidad del software en función de estándares internacionales, asegurando la alineación con los objetivos del proyecto.
3. **Análisis y aplicación de métricas:**
   Los estudiantes serán capaces de aplicar técnicas de medición para analizar el desempeño del software y tomar decisiones basadas en datos cuantitativos, utilizando herramientas tecnológicas adecuadas.
4. **Relación entre requisitos y calidad:**
   El estudiante desarrollará la habilidad de relacionar los requisitos funcionales y no funcionales con los indicadores de calidad, asegurando que los objetivos del proyecto sean medibles y alcanzables.
5. **Adopción de estándares internacionales:**
   El estudiante comprenderá la importancia de los estándares y normativas internacionales, como ISO/IEC 25010, y será capaz de integrar estos lineamientos en los procesos de desarrollo de software.

DIAGRAMA DE APRENDIZAJE

Mejorando el Rendimiento Organizacional a través de Métricas de Calidad



SÍNTESIS

En este documento se presenta una introducción a los conceptos fundamentales de la calidad y las métricas asociadas. La calidad es un aspecto crucial en cualquier organización, ya que determina la satisfacción del cliente y la eficiencia de los procesos. Las métricas, por su parte, son herramientas que permiten medir y evaluar la calidad de productos y servicios, facilitando la toma de decisiones informadas. A lo largo de este documento, se explorarán los principios de la calidad, la importancia de las métricas y cómo estas pueden ser implementadas para mejorar el rendimiento organizacional.

UNIDAD 1: INTRODUCCIÓN A LA CALIDAD

1.1. Conceptos y Generalidades

La calidad es un concepto multidimensional que se refiere al grado en que un conjunto de características inherentes de un producto, servicio o proceso cumple con los requisitos establecidos, ya sean estos normativos, funcionales o relacionados con las expectativas de los usuarios. Comprender y gestionar la calidad es esencial para garantizar la satisfacción del cliente, la eficiencia operativa y el cumplimiento de estándares internacionales.

Existen diversos enfoques para definir y evaluar la calidad, cada uno con su propia perspectiva y aplicación práctica. Entre los más destacados se encuentran:

- **Enfoque basado en el cliente**

  Este enfoque se centra en la percepción y satisfacción del cliente como principales indicadores de calidad. La calidad, desde esta perspectiva, es subjetiva y depende de la capacidad del producto o servicio para satisfacer las expectativas y necesidades de los usuarios finales. Un producto de alta calidad, según este enfoque, no solo cumple con los requisitos funcionales, sino que también genera una experiencia positiva y valor agregado para el cliente.

- **Enfoque basado en el proceso**

  Desde esta perspectiva, la calidad se evalúa en función de la eficacia y eficiencia de los procesos empleados para crear bienes o servicios. Un proceso bien diseñado y ejecutado garantiza que los resultados cumplan consistentemente con los estándares esperados. Este enfoque es clave para identificar y eliminar ineficiencias, reducir errores y mejorar la productividad, lo que resulta en una mejora continua de los resultados organizacionales.

- **Enfoque basado en la conformidad**

  Este enfoque define la calidad como el grado de cumplimiento con normas, especificaciones técnicas y estándares predefinidos. Es ampliamente utilizado en sectores regulados, donde la conformidad con requisitos específicos es crucial para garantizar la seguridad, la funcionalidad y la legalidad de los productos o servicios. Este enfoque

proporciona una medida objetiva de calidad, permitiendo evaluaciones precisas y repetibles.

Cada uno de estos enfoques ofrece una perspectiva complementaria sobre la calidad, y su aplicación puede variar según las necesidades y objetivos de la organización. Una gestión efectiva de la calidad implica integrar estos enfoques de manera estratégica, asegurando que se cumplan tanto los estándares técnicos como las expectativas del cliente y las metas organizacionales.

Este análisis brinda una base sólida para comprender la importancia de la calidad y los métodos para evaluarla, facilitando así la implementación de prácticas que promuevan la excelencia y la sostenibilidad en cualquier contexto organizacional.

Orígenes de la Calidad

Este documento explora la evolución de la calidad a lo largo de la historia, desde sus inicios en la artesanía hasta su transformación en la era digital. A través de diferentes períodos, se analizan los métodos de inspección y gestión de calidad que han surgido, destacando la influencia de figuras clave y el desarrollo de estándares internacionales. La calidad ha pasado de ser un concepto meramente artesanal a convertirse en un elemento fundamental en la producción y los servicios modernos, adaptándose a las necesidades cambiantes de la sociedad y la tecnología.

1. Inspección en la Artesanía

- En la antigüedad, la calidad era asegurada por artesanos que inspeccionaban su propio trabajo.
- Ejemplos: Construcción de las pirámides en Egipto, producción de armaduras en la Edad Media.

2. Era de la Revolución Industrial

- **Sistemas de Inspección:**
  - Durante la Revolución Industrial, la producción en masa llevó a la necesidad de sistemas formales de inspección.
  - Se implementaron controles de calidad al final del proceso de producción para identificar y corregir defectos.
  - Ejemplo: La fabricación de armas de fuego y maquinaria en el siglo XIX.

3. Siglo XX: Introducción de la Gestión de Calidad

- **Décadas de 1920 y 1930:**
  - **Walter A. Shewhart:**
    - Introdujo el Control Estadístico de Calidad (SQC).
    - Desarrollo de gráficos de control para monitorear la variabilidad en los procesos de producción.
  - **W. Edwards Deming y Joseph Juran:**
    - Promovieron la idea de que la calidad debería ser gestionada y no solo inspeccionada.
    - Enfoque en la mejora continua y la participación de todos los empleados en el proceso de calidad.

4. Post-Segunda Guerra Mundial: Japón y la Calidad Total

- **Revolución de Calidad en Japón:**

- Influencia de Deming y Juran en la industria japonesa.
- Introducción del concepto de Calidad Total (Total Quality Management, TQM).
- Enfoque en la mejora continua (Kaizen), involucrando a todos los niveles de la organización.
- Ejemplo: Toyota y el Sistema de Producción Toyota.

5. Décadas de 1970 y 1980: Internacionalización de la Calidad

- **Modelos de Calidad:**
  - Desarrollo de estándares internacionales como ISO 9000.
  - Creación de modelos de excelencia como el Premio Deming en Japón y el Malcolm Baldrige National Quality Award en EE.UU.
  - Enfoque en la satisfacción del cliente y la gestión de procesos.

6. Años 1990 y 2000: Enfoque en la Calidad del Software

- **Calidad en el Desarrollo de Software:**
  - Introducción de modelos de calidad específicos para software como CMMI (Capability Maturity Model Integration).
  - Desarrollo de normativas ISO/IEC 90003 e ISO/IEC 12207 para procesos de ingeniería de software.
  - Aumento del enfoque en la seguridad del software y la gestión de riesgos.

7. Siglo XXI: Calidad y Transformación Digital

- **Calidad en la Era Digital:**
  - Integración de tecnologías digitales en los sistemas de gestión de calidad.
  - Uso de big data y análisis predictivo para mejorar la calidad.
  - Enfoque en la experiencia del cliente y la calidad de servicio.
  - Aumento de la importancia de la sostenibilidad y la responsabilidad social corporativa en las estrategias de calidad.

Actividades de Aprendizaje

1. Lectura y Análisis de Artículos:

- Lectura de artículos sobre la evolución histórica de la calidad.

- Análisis de casos de estudio sobre la implementación de sistemas de calidad en diferentes industrias.

Definiciones Clásicas de Calidad

1. Joseph M. Juran

- Definición: Calidad es "adecuación al uso".

- Explicación: Juran enfatiza que la calidad debe ser vista desde la perspectiva del cliente y que un producto o servicio de calidad debe cumplir con las necesidades y expectativas de sus usuarios.

2. W. Edwards Deming

- Definición: Calidad es "el grado predecible de uniformidad y confiabilidad a bajo costo y adecuado a las necesidades del mercado".

- Explicación: Deming destaca la importancia de la consistencia y la fiabilidad en la calidad, así como la necesidad de reducir costos y satisfacer las necesidades del mercado.

3. Philip B. Crosby

- Definición: Calidad es "conformidad con los requisitos".

- Explicación: Crosby subraya que la calidad se logra cuando los productos o servicios cumplen con los requisitos especificados, sin desviaciones.

4. Armand V. Feigenbaum

- Definición: Calidad es "la corrección de problemas y su eliminación en el punto de origen".

- Explicación: Feigenbaum introduce el concepto de la calidad total, donde todos los departamentos y empleados de una organización son responsables de la calidad y deben trabajar para eliminar problemas en su origen.

Definiciones Modernas y Estándares Internacionales

1. ISO 9000:2015

- Definición: Calidad es "el grado en el que un conjunto de características inherentes cumple con los requisitos".

- Explicación: Esta definición se centra en la capacidad de un producto o servicio para cumplir con los requisitos establecidos, ya sean explícitos o implícitos.

2. ISO/IEC 25010:2011 (SQuaRE)

- Definición: Calidad es "la capacidad de un producto de software para satisfacer las necesidades y expectativas de los usuarios".

- Explicación: En el contexto del software, esta definición destaca la importancia de la satisfacción del usuario y la adecuación del producto a sus necesidades específicas.

3. American Society for Quality (ASQ)

- Definición: Calidad es "una medida de la excelencia de un producto o servicio".

- Explicación: La ASQ define la calidad en términos de excelencia, evaluando qué tan bien un producto o servicio cumple con los estándares y expectativas.

4. ISO 9001:2015

- Definición: Calidad es "la capacidad de un conjunto de características inherentes de un producto, sistema o proceso para cumplir con los requisitos de los clientes y otras partes interesadas".

- Explicación: Esta definición enfatiza la importancia de cumplir no solo con los requisitos del cliente, sino también con los de otras partes interesadas, como reguladores y la sociedad en general.

5. Genichi Taguchi

- Definición: Calidad es "la pérdida mínima impartida a la sociedad desde el momento en que el producto es enviado".

- Explicación: Taguchi introduce la idea de que la calidad debe minimizar el impacto negativo en la sociedad, incluyendo aspectos como el desperdicio y los costos adicionales.

6. David A. Garvin

- Definición: Calidad es "una combinación de ocho dimensiones: rendimiento, características, confiabilidad, conformidad, durabilidad, servicio, estética y calidad percibida".

- Explicación: Garvin proporciona una definición multidimensional de la calidad, reconociendo que diferentes aspectos contribuyen a la percepción general de calidad por parte del consumidor.

Actividades para los Estudiantes

1. Debate en Clase:

- Objetivo: Fomentar la discusión y el pensamiento crítico sobre las diferentes definiciones de calidad.

- Instrucciones: Organizar un debate donde los estudiantes defiendan la definición de calidad asignada a su grupo, argumentando por qué es la más efectiva o relevante en el contexto actual.

Métricas de Calidad

Objetivo

- Este módulo tiene como objetivo introducir y explorar las métricas de calidad más relevantes en el desarrollo de software y en la gestión de calidad en general. Se busca proporcionar las herramientas necesarias para medir y evaluar la calidad en distintos niveles, ya sea de producto, proceso o proyecto, con el fin de asegurar que se cumplan los estándares de calidad establecidos y se logren los objetivos organizacionales.

Contenidos

Las métricas de calidad son fundamentales para la toma de decisiones informadas y para la mejora continua en cualquier organización. Estas métricas permiten medir el rendimiento, identificar oportunidades de mejora, y asegurar la alineación de los proyectos con las expectativas de los clientes y los objetivos estratégicos de la organización.

Las métricas se pueden clasificar en tres grandes categorías: métricas de producto, métricas de proceso y métricas de proyecto. A continuación, se detallan cada uno de estos tipos de métricas.

1. Métricas de Producto

Las métricas de producto se centran en las características intrínsecas del software o producto que se está desarrollando. Son fundamentales para asegurar que el producto final cumpla con los estándares y expectativas de los usuarios finales.

Métricas de Tamaño:

- **Líneas de Código (LOC):** Esta métrica se utiliza para medir el tamaño de un software contando el número de líneas de código que lo componen. Es una métrica útil para estimar

la complejidad del proyecto, la cantidad de esfuerzo necesario para su desarrollo y los costos asociados. Sin embargo, esta métrica puede no reflejar completamente la calidad, ya que un software eficiente puede ser relativamente pequeño en líneas de código, mientras que uno grande podría estar sobrecargado de código innecesario.

- **Puntos de Función:** Los puntos de función miden la funcionalidad de un software desde la perspectiva del usuario. Se basa en la cantidad de entradas, salidas, consultas, archivos internos y archivos de interfaz que el sistema maneja. Esta métrica es útil porque mide la calidad en términos de lo que el software hace, no solo en términos de su tamaño, y es independiente del lenguaje de programación o de la plataforma utilizada.

Métricas de Complejidad:

- **Complejidad Ciclomática:** Mide la complejidad del flujo de control en un programa. Calcula el número de caminos lineales independientes a través del código, lo que puede ser un indicador de cuán difícil es de entender, probar y mantener el software. A mayor complejidad ciclomática, mayor es el riesgo de errores y de fallos durante el ciclo de vida del software.
- **Índice de Complejidad Halstead:** Esta métrica se calcula a partir del número de operadores y operandos en un programa, y mide la complejidad del software desde el punto de vista de la teoría de la información. Halstead sugiere que a mayor complejidad, mayor será el esfuerzo requerido para entender y modificar el código. Este índice también puede ayudar a prever la probabilidad de defectos en el software.

Métricas de Calidad del Código:

- **Densidad de Defectos:** Mide el número de defectos (errores) encontrados en el código por unidad de medida del software (como líneas de código). Una alta densidad de defectos indica que el software tiene una gran cantidad de errores y puede necesitar más pruebas y refactorización. Esta métrica es clave para evaluar la calidad del producto y detectar áreas de mejora.
- **Cobertura de Pruebas:** Se refiere al porcentaje del código que es ejecutado durante las pruebas automatizadas. Una alta cobertura de pruebas indica que el software ha sido exhaustivamente probado, lo que reduce el riesgo de errores no detectados. Sin embargo, tener una alta cobertura no siempre garantiza una alta calidad si las pruebas no están bien diseñadas o no cubren casos críticos.

1.2.1 Métricas de Proceso

Las métricas de proceso se utilizan para evaluar la eficiencia y efectividad de los procesos que se utilizan para desarrollar el software. Estas métricas son esenciales para el control de calidad y la mejora continua.

Métricas de Desempeño:

- **Tiempo de Ciclo:** Mide el tiempo total necesario para completar una tarea o proceso desde su inicio hasta su finalización. Este es un indicador crucial de la eficiencia operativa, ya que permite identificar cuellos de botella o demoras en el flujo de trabajo. La reducción del tiempo de ciclo contribuye a la mejora de la productividad y a la optimización de recursos.
- **Productividad del Equipo:** Mide la cantidad de trabajo completado por el equipo de desarrollo en un tiempo determinado. Se puede medir en términos de líneas de código, puntos de función o cualquier otra unidad de trabajo que se considere adecuada. Es un indicador útil para evaluar la eficiencia del equipo y para identificar oportunidades de mejora en su rendimiento.

Métricas de Eficiencia:

- **Defectos por Unidad de Producción:** Relaciona el número de defectos encontrados en el software con la cantidad de trabajo producido (por ejemplo, el número de líneas de código o puntos de función). Esta métrica ayuda a identificar la calidad del producto durante la fase de producción y a enfocar los esfuerzos de corrección en las áreas más propensas a generar errores.
- **Eficiencia de Costos:** Compara el costo de producción con la cantidad de trabajo realizado. Ayuda a evaluar la eficiencia económica de los proyectos y a identificar áreas donde los costos pueden ser reducidos sin sacrificar la calidad del producto.

Métricas de Mejora Continua:

- **Velocidad de Implementación de Mejoras:** Mide el tiempo necesario para implementar mejoras en el proceso de desarrollo, como la adopción de nuevas herramientas, prácticas ágiles o técnicas de programación. La velocidad de implementación es crucial para evaluar la agilidad de un equipo en la mejora continua.

- **Índice de Implementación de Sugerencias:** Mide el porcentaje de sugerencias de mejora que han sido implementadas con éxito. Un alto índice sugiere que el equipo está comprometido con la mejora continua y está dispuesto a adaptarse a nuevas ideas para mejorar la calidad y la eficiencia.

1.2.3. Métricas de Proyecto

Las métricas de proyecto ayudan a medir el progreso, los riesgos y la satisfacción del cliente a lo largo del ciclo de vida del proyecto.

Métricas de Planificación:

- **Desviación del Cronograma:** Compara el tiempo estimado para completar una tarea con el tiempo real utilizado. Esta métrica es clave para la gestión de plazos y la identificación de retrasos en el proyecto.
- **Cumplimiento del Presupuesto:**Compara el presupuesto inicial con los costos reales del proyecto. Permite identificar si el proyecto está siendo gestionado de manera eficiente en términos financieros.

Métricas de Riesgo:

- **Número de Riesgos Identificados:** Mide la cantidad de riesgos que han sido identificados durante las distintas fases del proyecto. Esta métrica ayuda a mantener un control proactivo sobre los riesgos potenciales.
- **Severidad de los Riesgos:** Mide el impacto potencial de los riesgos en el proyecto. Al clasificar los riesgos en términos de su severidad, los equipos pueden priorizar sus esfuerzos para mitigar los riesgos más graves.

3.3. Métricas de Satisfacción del Cliente:

- **Encuestas de Satisfacción:** Las encuestas permiten evaluar la satisfacción de los clientes con el producto o servicio entregado. Los resultados son esenciales para comprender si se han cumplido las expectativas del cliente y para mejorar futuras versiones del producto.
- **Índice de Retorno del Cliente:** Mide cuán frecuentemente los clientes regresan para utilizar el producto o servicio nuevamente. Este índice es un buen indicativo de la lealtad del cliente y la calidad percibida del producto.

Importancia de las Métricas en la Gestión de Calidad

Las métricas de calidad no solo permiten evaluar la calidad de los productos y procesos, sino que también facilitan la gestión efectiva de proyectos y la satisfacción del cliente. A continuación, se detallan los beneficios de utilizar métricas de calidad en la gestión organizacional:

1. Medición y Evaluación:

- **Objetividad:** Las métricas proporcionan una base cuantitativa y objetiva para evaluar el rendimiento, lo que elimina la subjetividad y mejora la toma de decisiones.
- **Benchmarking:** Las métricas permiten comparar el rendimiento con estándares industriales, competidores y mejores prácticas, ayudando a identificar áreas de mejora y establecer metas alcanzables.

2. Control y Monitoreo:

- **Seguimiento en Tiempo Real:** Las métricas de calidad permiten monitorear el desempeño del proyecto en tiempo real, asegurando que se estén cumpliendo los objetivos y detectando problemas a tiempo.
- **Identificación de Desviaciones:** Facilitan la identificación temprana de desviaciones respecto al plan, lo que permite corregir a tiempo problemas que podrían escalar en el futuro.

3. Mejora Continua:

- **Análisis de Tendencias:** Las métricas proporcionan datos históricos que permiten analizar tendencias y patrones de desempeño, ayudando a identificar áreas de mejora y optimización.
- **Implementación de Cambios:**

Basadas en datos, las métricas permiten hacer cambios y ajustes informados para optimizar procesos y productos, lo que incrementa la eficiencia a largo plazo.

4. Gestión de Riesgos:

- **Mitigación Preventiva:** Las métricas de calidad ayudan a identificar riesgos y problemas potenciales antes de que ocurran, lo que permite tomar medidas preventivas para reducir su impacto.
- **Evaluación de Impacto:** Las métricas permiten evaluar el impacto potencial de los riesgos y priorizar las acciones correctivas según su gravedad.

5. Satisfacción del Cliente:

- **Cumplimiento de Expectativas:** Las métricas garantizan que los productos y servicios cumplan con las expectativas del cliente, lo que mejora la satisfacción y la fidelidad.
- **Retroalimentación Constante:** Las métricas proporcionan información valiosa para la retroalimentación continua, lo que permite hacer mejoras y ajustes de manera proactiva en función de las necesidades cambiantes del cliente.

Este enfoque detallado ofrece una comprensión exhaustiva de las métricas de calidad, proporcionando las bases necesarias para su implementación efectiva en la gestión de proyectos, procesos y productos.

UNIDAD 2: GESTIÓN Y ENFOQUES DE LA CALIDAD

- Principios de la gestión de la calidad total
- Cliente, empleados, procesos,
- Sistema integrado, enfoque estratégico, Mejora continua, Toma de decisiones basada en información, Comunicaciones.
- Autores de la calidad
- Enfoques de calidad
- Estándares de Calidad-Modelos y normas de calidad
- Familia ISO 9000
- - ISO 9001
- - ISO 9004
- Familia de ISO 25000

Resultado de Aprendizaje

El estudiante será capaz de comprender y aplicar los principios fundamentales de la gestión de la calidad total, integrando clientes, empleados y procesos en un sistema organizado y estratégico. Identificará los enfoques clave de la calidad, como la mejora continua, la toma de decisiones informadas y las comunicaciones efectivas. Además, reconocerá las contribuciones de los principales autores en la calidad, explicando sus teorías y su impacto en las prácticas actuales. Podrá analizar y diferenciar enfoques y metodologías de calidad, así como interpretar y aplicar los estándares internacionales, incluyendo la familia ISO 9000 e ISO 25000, especialmente en el contexto de la mejora de la calidad en el desarrollo de software.

DIAGRAMA DE APRENDIZAJE

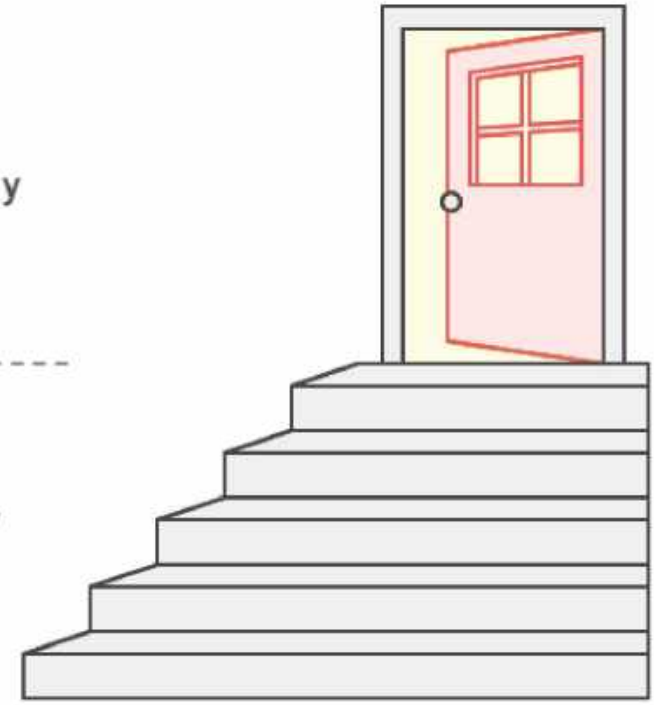
Mejorar la calidad del software a través de los principios de gestión de la calidad total.

Lograr la excelencia en la calidad del software

Implementar estándares ISO y mejora continua.

Falta de gestión de calidad estructurada

Procesos desorganizados y estándares poco claros.



SINTESIS

Esta unidad proporciona una guía comprensiva sobre los principios fundamentales de la gestión de la calidad total, enfocándose en la integración de clientes, empleados y procesos en un sistema organizado y estratégico. Se abordarán enfoques clave de la calidad, la mejora continua, la toma de decisiones informadas y las comunicaciones efectivas. Además, se explorarán las contribuciones de autores destacados en el ámbito de la calidad y se analizarán las metodologías y estándares internacionales relevantes, con especial énfasis en la familia ISO 9000 e ISO 25000 en el contexto del desarrollo de software.

.

2.1 Principios de la Gestión de Calidad Total (TQM)

Enfoque en el cliente

El cliente define lo que significa calidad para el producto. En el desarrollo de software, los equipos deben trabajar estrechamente con los usuarios para garantizar que se cumplan sus expectativas.

- **Ejemplo real:**

  En el desarrollo de una aplicación bancaria, los clientes desean funciones como transferencias rápidas, historial de transacciones y seguridad. El equipo realiza entrevistas y sesiones de "prueba de concepto" con los usuarios finales para definir prioridades. Esto se traduce en historias de usuario en metodologías ágiles.

Compromiso de los empleados

El éxito de la calidad depende del compromiso de todos los niveles de la organización.

- **Ejemplo práctico:**

  En un proyecto de desarrollo de un ERP (Sistema de Planificación de Recursos Empresariales), cada desarrollador tiene asignada una tarea específica, pero todos participan en reuniones diarias para identificar problemas y proponer soluciones conjuntas.

Enfoque basado en procesos

La calidad mejora cuando cada actividad se considera parte de un proceso mayor.

- **Aplicación en software:**

  Al implementar una herramienta CRM, el flujo de trabajo incluye las fases de diseño, desarrollo, pruebas y despliegue. Cada paso se mide para identificar cuellos de botella.

Sistema integrado de gestión

La integración de sistemas asegura que los objetivos de calidad estén alineados con los objetivos estratégicos.

- **Ejemplo práctico:**

Un equipo usa Jenkins para la integración continua y JIRA para gestionar incidencias. Estos sistemas están interconectados para automatizar la asignación de tareas cuando una compilación falla.

Mejora continua

La mejora es un esfuerzo constante.

- **Ejemplo real:**

  En un equipo de desarrollo ágil, después de cada sprint se realiza una retrospectiva. Identifican que los tiempos de compilación son largos y deciden implementar un servicio en la nube para acelerar el proceso.

Toma de decisiones basada en hechos

Los datos deben respaldar las decisiones de calidad.

- **Ejemplo práctico:**

  Un equipo utiliza Google Analytics para evaluar el rendimiento de una aplicación web. Los datos muestran que los usuarios abandonan el sitio debido a tiempos de carga elevados, lo que lleva a optimizar el backend.

2.2 Autores de la Calidad y su Relevancia en Software

1. W. Edwards Deming: Ciclo PDCA

El ciclo PDCA (Planificar, Hacer, Verificar, Actuar) es útil para iteraciones en software.

- **Ejemplo práctico:**

  Un equipo adopta PDCA para desarrollar un módulo de pagos en línea:

  - **Planificar:** Analizan los requisitos y diseñan el módulo.
  - **Hacer:** Desarrollan y prueban el código.
  - **Verificar:** Realizan pruebas funcionales.
  - **Actuar:** Ajustan el código según los resultados de las pruebas.

2. Joseph M. Juran: Trilogía de Juran

Incluye planificación, control y mejora de calidad.

- **Aplicación real:**

Durante el desarrollo de una plataforma educativa, planifican las funcionalidades básicas, controlan los defectos durante la implementación y mejoran la interfaz basándose en el feedback de los usuarios.

3. Philip Crosby: Calidad es Gratis y Cero Defectos

Enfatiza que prevenir defectos es menos costoso que corregirlos.

- **Ejemplo práctico:**

  Usar herramientas de análisis estático de código, como SonarQube, permite detectar defectos antes de la integración.

2.3. Estándares de Calidad: Modelos y Normas

Familia ISO 9000

Estos estándares proporcionan directrices para implementar un Sistema de Gestión de Calidad (SGC).

ISO 9001: Gestión de la Calidad

ISO 9001 establece los requisitos para un SGC y es ampliamente utilizado en software.

- **Ejemplo real:**

  Una empresa desarrolla un software médico y sigue ISO 9001 para documentar requisitos de usuario, definir procesos de validación y asegurar trazabilidad en cada cambio. Usan herramientas como GitLab para rastrear cambios en el código.

ISO 9004: Mejora del Desempeño

Esta norma complementa ISO 9001 y se centra en la sostenibilidad y mejora continua.

- **Ejemplo práctico:**

  Una empresa de videojuegos realiza análisis de satisfacción del cliente después de cada lanzamiento y adapta su estrategia de marketing y diseño.

Familia de Normas ISO 25000 (SQuaRE)

La familia ISO/IEC 25000 (Systems and software Quality Requirements and Evaluation, o SQuaRE) se enfoca en la calidad del software a lo largo de todo su ciclo de vida. Esta familia proporciona un marco que permite definir, medir y evaluar la calidad de los productos de software mediante un enfoque estandarizado y orientado a resultados.

Componentes Principales de la Familia ISO 25000

1. **ISO/IEC 25010**: Es la norma central, ya que establece un modelo de calidad para productos de software. Este modelo se divide en dos perspectivas:
   - **Modelo de calidad del producto**: Evalúa características como funcionalidad, confiabilidad, usabilidad, eficiencia de desempeño, mantenibilidad, portabilidad, compatibilidad y seguridad.

- **Modelo de calidad en uso**: Se centra en el impacto del software sobre el usuario final y evalúa características como efectividad, eficiencia, satisfacción y libertad de riesgo.

2. **ISO/IEC 25012**: Define atributos de calidad para los datos utilizados o generados por el software, como precisión, completitud, consistencia y accesibilidad.
3. **ISO/IEC 25020-25029**: Proporcionan guías y métricas para la medición de calidad en software. Estas normas incluyen directrices para evaluar y certificar la calidad del producto en función de los atributos establecidos en la ISO/IEC 25010.
4. **ISO/IEC 25040-25045**: Describen procesos y métodos para la evaluación sistemática de la calidad del software. Incluyen herramientas prácticas para asegurar que el software cumpla con los requisitos establecidos.

Uso de la Familia ISO 25000 en el Ciclo de Vida del Software

La implementación de las normas ISO 25000 en el ciclo de vida del software sigue un enfoque estructurado, asegurando que la calidad sea gestionada desde la concepción del producto hasta su retiro. A continuación, se detallan las principales fases donde estas normas se aplican:

1. **Definición de Requisitos de Calidad**:
   - Durante la etapa de planificación y diseño, se utilizan las normas de SQuaRE para definir criterios de calidad específicos que deben cumplir los productos de software.
   - El modelo ISO/IEC 25010 guía a los equipos de desarrollo en la identificación de objetivos como la usabilidad, confiabilidad o eficiencia, según las necesidades del cliente.
2. **Diseño y Desarrollo**:
   - Los estándares de calidad se incorporan en las fases iniciales del diseño del software para evitar fallos futuros.
   - Por ejemplo, la característica de **seguridad** definida en la ISO/IEC 25010 se puede implementar mediante protocolos de autenticación y manejo seguro de datos desde la fase de codificación.
3. **Evaluación del Software**:
   - Durante las fases de pruebas y validación, las normas ISO/IEC 25040 y 25045 proporcionan guías para llevar a cabo evaluaciones formales.
   - Se emplean métricas específicas para medir atributos como el rendimiento, la fiabilidad o la compatibilidad.
4. **Implementación y Mantenimiento**:
   - En la etapa de entrega e implementación, las normas aseguran que el software cumpla con las expectativas de los usuarios y pueda ser integrado sin problemas en el entorno operativo.
   - Las auditorías regulares basadas en la familia SQuaRE permiten identificar áreas de mejora durante el mantenimiento.
5. **Revisión y Mejora Continua**:

- La mejora continua es un principio clave de la familia ISO 25000. Al recopilar métricas y feedback de los usuarios, se pueden actualizar los procesos de desarrollo para futuros proyectos.

Beneficios de ISO 25000 en el Desarrollo de Software

- **Garantía de calidad**: Los estándares aseguran que el software cumple con criterios internacionales reconocidos.
- **Satisfacción del cliente**: Se centra en las necesidades del usuario final, asegurando un producto funcional y fiable.
- **Optimización de recursos**: La implementación temprana de estándares evita retrabajos y reduce costos.
- **Competitividad en el mercado**: Los productos que cumplen con la ISO 25000 tienen mayor aceptación en mercados globales.

Ejemplo Práctico: Uso de ISO/IEC 25010 en una Aplicación Web

Una empresa que desarrolla una aplicación de comercio electrónico puede utilizar el modelo ISO/IEC 25010 de la siguiente manera:

1. **Definir requisitos funcionales**: La aplicación debe permitir a los usuarios realizar compras seguras, gestionar sus perfiles y recibir notificaciones.
2. **Establecer requisitos de seguridad**: Se implementan medidas como cifrado de datos y autenticación de usuarios.
3. **Evaluar usabilidad**: Se realizan pruebas con usuarios para garantizar que la interfaz sea fácil de navegar.
4. **Medir el rendimiento**: Se monitorea el tiempo de carga de las páginas web y el manejo de múltiples solicitudes simultáneas.
5. **Asegurar mantenibilidad**: El código se documenta adecuadamente para facilitar futuras actualizaciones.

En conclusión, la familia ISO 25000 ofrece un marco robusto para gestionar y evaluar la calidad del software, asegurando que este sea confiable, eficiente y adecuado para su propósito. Su implementación no solo mejora los procesos internos, sino que también aumenta la confianza del cliente y la competitividad en el mercado.

(ISO/IEC, 2011)

UNIDAD 3: Diagramas UML y Seguridad de Software

3.1 Métricas para Diagramas UML

3.1.1 Introducción

3.1.2 Tipos de UML

3.1.3 Plataformas para diagramas UML

3.2 Práctica en la plataforma Lucidchart

3.3 Pruebas de Software

3.3.1 Técnicas de pruebas

3.3.2 **Procesos de** pruebas

3.3.3 Herramientas de automatización

3.4 Seguridad de software

3.4.1 Tipos de seguridad de software

3.4.2 Seguridad en el ciclo de desarrollo de **software**

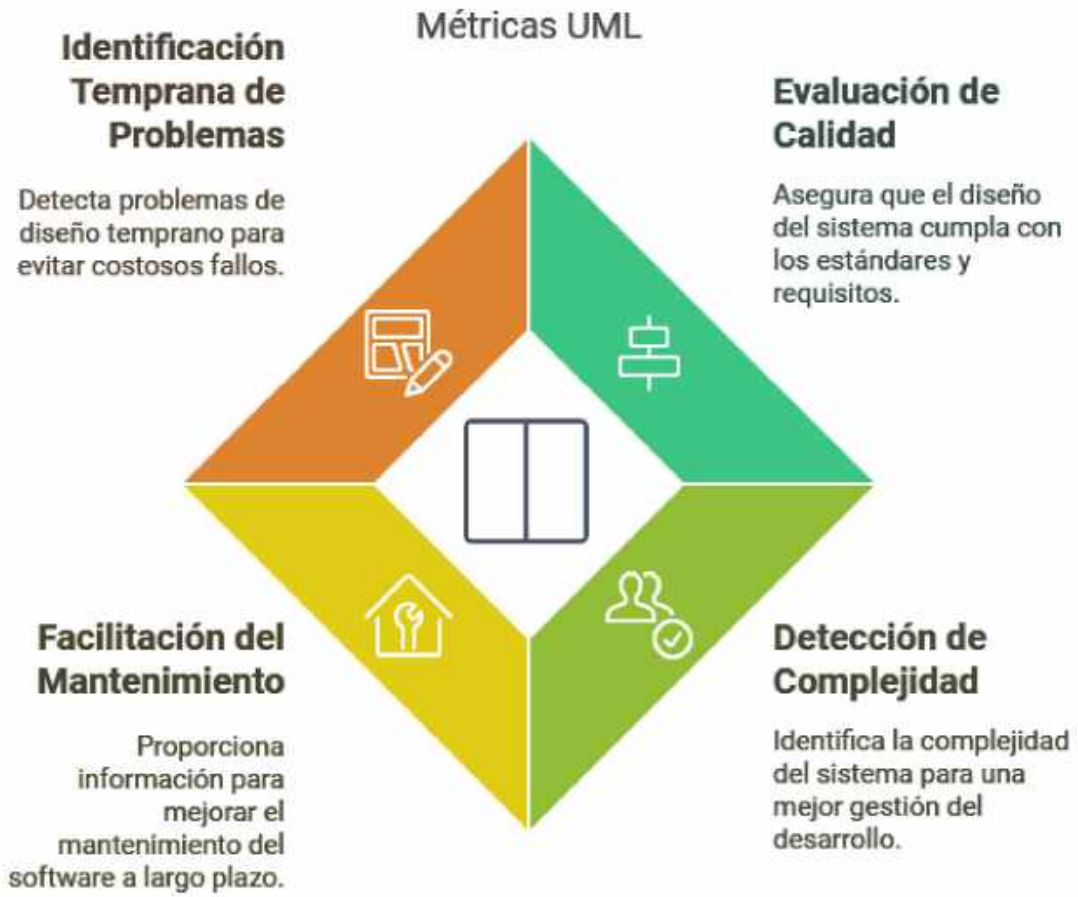
3.4.3 Confiabilidad, disponibilidad e integridad en el Desarrollo de Software.

3.4.4. Gestión de pruebas completas de seguridad de **software.**

Resultado de Aprendizaje

Al finalizar esta unidad, el estudiante será capaz de diseñar y analizar diagramas UML aplicando métricas y herramientas tecnológicas, implementar prácticas de seguridad en el ciclo de desarrollo de software, y realizar pruebas de calidad y seguridad utilizando técnicas y herramientas automatizadas, garantizando la confiabilidad, disponibilidad e integridad del software desarrollado.

DIAGRAMA DE APRENDIZAJE



SINTESIS

El desarrollo de software moderno requiere metodologías y herramientas que garanticen la calidad y seguridad de los sistemas. En este contexto, los **diagramas UML (Unified Modeling Language)** se destacan como una herramienta esencial para el modelado, diseño y documentación de aplicaciones. Estos diagramas permiten representar visualmente la estructura, el comportamiento y las interacciones dentro de un sistema, facilitando la comunicación entre los equipos de desarrollo y mejorando la eficiencia en la gestión de proyectos.

Por otro lado, la **seguridad de software** es un componente crítico en el ciclo de vida del desarrollo, ya que asegura que las aplicaciones sean resilientes frente a vulnerabilidades y amenazas. Incorporar prácticas de

seguridad desde las primeras fases del diseño y combinarlo con un análisis detallado mediante UML garantiza soluciones robustas, confiables y alineadas con estándares internacionales de calidad.

En conjunto, los diagramas UML y las estrategias de seguridad permiten a los desarrolladores no solo visualizar y planificar sistemas complejos, sino también construir aplicaciones seguras, manteniendo la integridad, disponibilidad y confidencialidad de los datos. Este enfoque integral es fundamental para enfrentar los retos actuales en el desarrollo de software.

3.1 **Métricas para diagramas UML**

Los diagramas UML (Unified Modeling Language) son herramientas visuales fundamentales para modelar, diseñar y documentar sistemas de software. Las métricas para diagramas UML permiten medir aspectos cuantitativos y cualitativos del diseño del sistema, facilitando la evaluación de la calidad, complejidad y mantenimiento del software. Estas métricas son esenciales para identificar problemas en etapas tempranas del desarrollo y garantizar que el diseño cumpla con los objetivos.

3.1.1. Introducción

Las métricas para diagramas UML son herramientas fundamentales en el desarrollo de software, ya que permiten evaluar la calidad y eficiencia del diseño modelado mediante el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés). Estas métricas proporcionan datos cuantitativos que facilitan la medición de aspectos como la complejidad, cohesión, acoplamiento y escalabilidad del sistema. Su aplicación ayuda a garantizar que el diseño cumpla con los estándares de calidad establecidos, permitiendo identificar problemas en etapas tempranas del ciclo de vida del software.

El uso de métricas en diagramas UML contribuye a optimizar el diseño, ya que detecta posibles inconsistencias o elementos que podrían dificultar el mantenimiento o la comprensión del sistema. Estas métricas permiten evaluar diversos aspectos de los diagramas, como el número de elementos presentes, la cantidad de relaciones entre ellos y el nivel de cohesión dentro de las clases. Además, proporcionan información valiosa sobre la complejidad del sistema, lo que facilita la planificación de recursos y tiempos durante las etapas posteriores del desarrollo.

Otro beneficio importante de las métricas UML es su capacidad para fomentar la colaboración entre los diferentes roles involucrados en un proyecto, como desarrolladores, analistas y gerentes. Al ofrecer una base común para analizar el diseño, estas métricas apoyan la toma de decisiones informadas, mejorando la comunicación y reduciendo el riesgo de errores en la implementación. En conjunto, estas herramientas no solo mejoran la calidad del software, sino que también optimizan los procesos y reducen costos a largo plazo.

Por lo tanto, la integración de métricas en los diagramas UML representa una práctica esencial para cualquier equipo de desarrollo que busque construir sistemas confiables, sostenibles y alineados con los objetivos del proyecto.

3.1.2 Tipos de UML

UML (Unified Modeling Language) y sus tipos de diagramas

El **UML** es un lenguaje gráfico que permite modelar sistemas de software complejos de forma visual, facilitando la comunicación entre desarrolladores, analistas y usuarios. Los diagramas UML se utilizan en todas las fases del desarrollo de software, desde el análisis de requerimientos hasta la implementación, pruebas y mantenimiento, ayudando en el proceso de evaluación del software al ofrecer una representación clara de su estructura y comportamiento.

1. Diagramas estructurales

Los diagramas estructurales se centran en describir los aspectos estáticos del sistema, como la relación entre sus clases, objetos y componentes. Son esenciales durante la fase de diseño y permiten evaluar la arquitectura del sistema.

- **1.1 Diagrama de clases**

Descripción:

Este diagrama representa la estructura estática del sistema mostrando:

Clases: Las entidades principales del sistema.

Atributos: Las propiedades o datos que describe cada clase.

Métodos: Las operaciones o comportamientos que las clases pueden realizar.

Relaciones:

Herencia: Representa una relación "es un" entre una clase base y una clase derivada.

Asociación: Relación entre dos clases que interactúan entre sí.

Dependencia: Una clase utiliza a otra temporalmente.

Agregación y composición: Representan relaciones "todo/parte".

Uso:

El diagrama de clases es fundamental en la fase de diseño orientado a objetos, ya que permite:

Modelar el sistema de forma detallada.

Identificar correctamente los componentes reutilizables.

Facilitar la generación de código.

Evaluación del software:

Ayuda a verificar que la estructura del sistema esté correctamente modelada, que las relaciones entre clases sean coherentes y que no existan dependencias innecesarias que puedan dificultar el mantenimiento.

Diagrama de objetos

Descripción:

Este diagrama es una instancia del diagrama de clases, es decir, muestra objetos concretos con valores específicos de atributos en un momento determinado.

Uso: Se utiliza principalmente para:

Mostrar el estado del sistema en un momento específico durante su ejecución.

Validar que las relaciones entre objetos definidos en el diagrama de clases sean correctas.

Evaluación del software:

Permite comprobar el estado de los objetos en situaciones específicas, validando su correcto funcionamiento según el diseño.

Diagrama de componentes

Descripción:

Este diagrama representa la estructura física del sistema, definiendo cómo se organizan los componentes de software, como bibliotecas, módulos y archivos ejecutables.

*Uso:*Es útil durante la fase de implementación para:

- Especificar la arquitectura física del sistema.
- Identificar los componentes reutilizables.
- Facilitar la distribución de los componentes en diferentes entornos.

Evaluación del software:

Ayuda a garantizar que los componentes del sistema estén correctamente integrados y que puedan ser reutilizados en otros proyectos, lo que mejora la eficiencia del desarrollo.

Diagramas de comportamiento

Estos diagramas modelan los aspectos dinámicos del sistema, es decir, cómo el sistema reacciona a eventos y cómo los objetos interactúan y cambian de estado con el tiempo.

Diagrama de casos de uso

Descripción:

Muestra las funcionalidades o servicios que el sistema ofrece a los actores externos (usuarios u otros sistemas). Un caso de uso describe una secuencia de interacciones entre los actores y el sistema para lograr un objetivo específico.

Uso: Se emplea en la fase de análisis de requerimientos.

Facilita la comunicación con los usuarios finales.

Define el alcance del sistema.

Evaluación del software:

Ayuda a verificar que todas las funcionalidades requeridas por los usuarios se hayan considerado y que se haya definido claramente cómo interactuarán con el sistema.

Diagrama de actividades

Descripción:

Modela flujos de trabajo o procesos dentro del sistema. Representa actividades, decisiones y flujos paralelos.

Uso:

Se emplea para modelar procesos complejos y secuencias de tareas.

Ayuda a identificar posibles cuellos de botella o actividades que puedan ser ejecutadas en paralelo.

Evaluación del software:

Permite evaluar si el flujo de trabajo del sistema es correcto y eficiente, ayudando a detectar errores lógicos y optimizar procesos.

Diagrama de estados

Descripción:

Modela los diferentes estados por los que pasa un objeto a lo largo de su vida, así como las transiciones entre esos estados.

Uso: Se utiliza para:

- Modelar el comportamiento de objetos con ciclos de vida complejos.
- Representar cómo los eventos desencadenan cambios de estado.

Evaluación del software:

Permite validar que los objetos cambien de estado correctamente en respuesta a eventos, evitando errores de comportamiento durante la ejecución.

Diagramas de interacción

Estos diagramas detallan cómo los objetos interactúan entre sí, centrándose en el intercambio de mensajes en tiempo de ejecución.

Diagrama de secuencia

Descripción:

Representa cómo los objetos interactúan en un flujo temporal, mostrando el orden en que se envían los mensajes.

Uso:

- Se emplea para modelar interacciones específicas dentro de un caso de uso.
- Permite identificar la secuencia de pasos necesaria para ejecutar una funcionalidad.

Evaluación del software:

- Ayuda a verificar que la secuencia de interacciones sea correcta y que los mensajes se envíen en el orden adecuado, garantizando que el sistema cumpla con los requisitos.

Diagrama de colaboración

Descripción:

Destaca cómo los objetos colaboran entre sí para llevar a cabo una funcionalidad. Se enfoca en las relaciones entre objetos más que en el tiempo de ejecución.

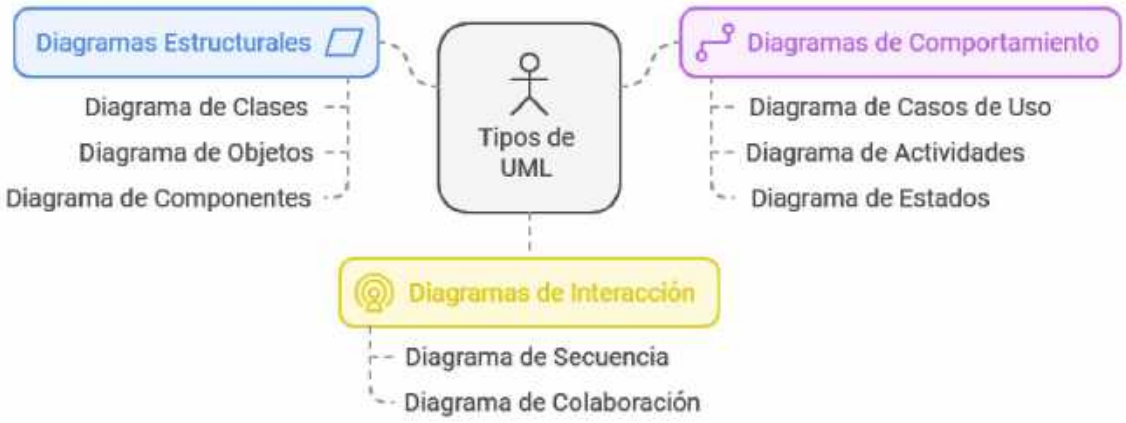
Uso:

- Se utiliza para modelar interacciones complejas que requieren la colaboración de múltiples objetos.
- Permite identificar cómo los objetos trabajan juntos para lograr un objetivo.

Evaluación del software:

- Ayuda a garantizar que la colaboración entre objetos esté correctamente diseñada, mejorando la cohesión y reduciendo el acoplamiento.

El uso de diagramas UML facilita el proceso de evaluación del software al ofrecer una representación clara de su diseño y comportamiento. Permite a los desarrolladores identificar errores, optimizar el diseño y asegurar que el sistema cumpla con los requisitos funcionales y no funcionales establecidos. Además, la correcta documentación a través de UML mejora el mantenimiento y la escalabilidad del software.



3.1.3 Plataformas para diagramas UML

El diseño de software utilizando diagramas UML (Unified Modeling Language) requiere herramientas específicas que permitan crear y gestionar diagramas de forma eficiente. Las plataformas para diagramas UML ofrecen funcionalidades adaptadas a las necesidades de los desarrolladores, desde interfaces amigables hasta capacidades avanzadas de modelado. A continuación, se presentan algunas de las plataformas más utilizadas, junto con sus características, ventajas, y casos prácticos de aplicación.

- **Lucidchart**

Descripción:

Lucidchart es una plataforma en línea que permite crear diagramas de UML y otros tipos de diagramas de manera intuitiva. Ofrece funciones de colaboración en tiempo real, lo que facilita el trabajo en equipo y la revisión conjunta de los diagramas.

Características clave:

- Interfaz amigable basada en arrastrar y soltar.
- Colaboración en tiempo real para equipos distribuidos.
- Integración con herramientas populares como Google Workspace, Microsoft Teams y Slack.
- Amplia biblioteca de plantillas y formas UML prediseñadas.

Ejemplo práctico:

Una empresa de desarrollo de software que trabaja de manera remota puede usar Lucidchart para diseñar un diagrama de clases, permitiendo que todos los miembros del equipo revisen y editen el modelo en tiempo real. Por ejemplo, un equipo podría modelar el sistema de reservas de un hotel, definiendo clases como "Habitación", "Cliente" y "Reserva", y sus relaciones.

- **Microsoft Visio**

Descripción:

Microsoft Visio es una herramienta profesional ampliamente utilizada en empresas para la creación de diagramas de alta calidad, incluidos diagramas UML. Se destaca por su capacidad de personalización y su compatibilidad con otras aplicaciones de Microsoft Office.

Características clave:

- Amplias opciones de diseño y formato para diagramas detallados.
- Integración total con el ecosistema de Microsoft, como Excel y PowerPoint.
- Soporte para diagramas complejos y detallados, con capacidad de análisis avanzado.
- Plantillas para diagramas UML, como casos de uso, clases y actividades.

Ejemplo práctico:

Una institución financiera que desarrolla un sistema de gestión de clientes puede utilizar Microsoft Visio para diseñar un diagrama de actividades, representando el flujo de procesos como la creación de cuentas, la aprobación de créditos y la generación de reportes. Este diagrama puede exportarse a PowerPoint para ser presentado a los gerentes del proyecto.

- **StarUML**

Descripción:

StarUML es un software dedicado al modelado UML que ofrece un entorno especializado para

desarrolladores y arquitectos de software. Es ideal para quienes necesitan funcionalidades avanzadas en el diseño de sistemas complejos.

Características clave:

- Soporte para UML 2.x, permitiendo trabajar con los estándares más recientes.
- Capacidad para generar código a partir de los diagramas.
- Extensible mediante complementos y scripts.
- Compatible con diagramas de casos de uso, clases, actividades, estados y más.

Ejemplo práctico:

Un equipo de desarrollo que crea un sistema de comercio electrónico puede usar StarUML para diseñar diagramas detallados de secuencia y clases, como el flujo de "Carrito de Compras" y la relación entre "Productos", "Usuarios" y "Órdenes". Además, pueden generar automáticamente fragmentos de código en lenguajes como Java o C#.

- **Draw.io (ahora Diagrams.net)**

Descripción:

Draw.io, conocido actualmente como Diagrams.net, es una herramienta gratuita basada en navegador que permite crear diagramas UML básicos. Es ideal para estudiantes, principiantes y proyectos pequeños que no requieren características avanzadas.

Características clave:

- Acceso gratuito sin necesidad de licencias.
- Integración con Google Drive y Dropbox para almacenamiento en la nube.
- Plantillas simples para diagramas UML.
- Función de exportación a formatos como PNG, PDF y SVG.

Ejemplo práctico:

Un grupo de estudiantes que trabaja en un proyecto universitario puede usar Draw.io para diseñar un diagrama de casos de uso que represente las interacciones principales en una aplicación de biblioteca, con actores como "Usuario", "Administrador" y "Sistema".

Comparativa de las plataformas

Plataforma	Costo	Uso Ideal	Nivel de Complejidad
Lucidchart	Gratuito y pago	Trabajo en equipo, proyectos colaborativos.	Medio
Microsoft Visio	Pago	Empresas y proyectos de alta calidad.	Avanzado
StarUML	Pago único	Diseño avanzado de sistemas complejos.	Alto
Draw.io	Gratuito	Proyectos pequeños y educativos.	Bajo

3.2 Práctica en la plataforma Lucidchart

Lucidchart es una herramienta en línea que permite la creación y edición colaborativa de diagramas UML.

Pasos para la práctica:

1. Registrarse en la plataforma (opcional, versión gratuita disponible).
2. Selecciona una plantilla UML o empieza desde cero.
3. Usar las herramientas de arrastrar y soltar para diagramar clases, casos de uso, secuencias u otros tipos UML.
4. Incorporar colores y notas explicativas para una mejor visualización.
5. Exportar el diagrama en formatos como PDF o PNG para su integración en documentos.

3.3 Pruebas de software

3.3.1 Técnicas de prueba

Las técnicas de pruebas aseguran que el software funcione según lo esperado. Tipos principales:

- **Pruebas de caja negra:** Evalúan la funcionalidad sin conocer la implementación interna.
- **Pruebas de caja blanca:** Analizan la lógica interna del código.
- **Pruebas de regresión:** Verifican que nuevas modificaciones no afectan las funcionalidades existentes.
- **Pruebas de estrés:** Miden el desempeño del software bajo condiciones extremas.

3.3.2 Procesos de prueba

Los procesos de pruebas suelen seguir estos pasos:

1. **Planificación:** Definir objetivos, alcance y recursos.
2. **Diseño:** Crear casos de prueba y seleccionar herramientas.
3. **Ejecución:** Realizar las pruebas según los casos diseñados.
4. **Registro:** Documentar los resultados y problemas encontrados.
5. **Cierre:** Resolver defectos y confirmar los arreglos.

3.3.3 Herramientas de automatización

La automatización de pruebas mejora la eficiencia y precisión. Herramientas comunes:

- **Selenium:** Automatiza pruebas para aplicaciones web.
- **JUnit:** Usado para pruebas unitarias en Java.
- **TestNG:** Similar a JUnit, con funcionalidades avanzadas para pruebas.

3.4 Seguridad del software

3.4.1 Tipos de seguridad de software

Tipos de Seguridad de Software

La seguridad de software se refiere a la implementación de prácticas, tecnologías y procesos que garantizan que un sistema o aplicación sea resistente frente a amenazas, protegiendo tanto los datos como la funcionalidad del software. Este enfoque abarca múltiples dimensiones que trabajan juntas para salvaguardar la confidencialidad, integridad y disponibilidad de la información y los sistemas. A continuación, se detallan los principales tipos de seguridad de software y su importancia.

Seguridad de Datos

La seguridad de datos se centra en proteger la información almacenada, procesada o transmitida por el software frente a accesos no autorizados, pérdida o modificación. Este tipo de seguridad es esencial para garantizar la privacidad de los usuarios y el cumplimiento de normativas legales como el **Reglamento General de Protección de Datos (GDPR)** o la **Ley de Protección de Datos Personales en Ecuador**.

Principales medidas para la seguridad de datos:

- **Cifrado de datos:** Transformar la información en un formato ilegible para quienes no tienen acceso autorizado. Ejemplo: el uso de **TLS (Transport Layer Security)** en comunicaciones web.
- **Control de acceso:** Implementar autenticación y autorización para garantizar que solo los usuarios correctos accedan a la información.
- **Copia de seguridad (backups):** Crear respaldos periódicos de datos críticos para prevenir pérdidas.

Ejemplo práctico:
En una aplicación bancaria, la seguridad de datos asegura que la información personal y financiera de los clientes esté protegida contra ataques como el robo de identidad. Esto se logra mediante cifrado de extremo a extremo y sistemas de autenticación multifactor.

Seguridad de Aplicaciones

La seguridad de aplicaciones busca prevenir y mitigar vulnerabilidades dentro del código del software. Este enfoque se centra en identificar errores en el desarrollo que podrían ser explotados por atacantes para comprometer la funcionalidad del sistema o acceder a datos sensibles.

Principales prácticas para la seguridad de aplicaciones:

- **Pruebas de penetración (pentesting):** Simular ataques para identificar vulnerabilidades antes de que los hackers reales puedan explotarlas.
- **Validación de entradas:** Asegurarse de que los datos ingresados por los usuarios no contengan elementos maliciosos como **inyecciones SQL** o scripts dañinos.
- **Actualización y parcheo:** Mantener el software actualizado para corregir fallas de seguridad conocidas.

Ejemplo práctico:
Una tienda en línea implementa medidas de seguridad en su aplicación para evitar ataques de inyección SQL que podrían exponer datos confidenciales, como nombres de usuarios y contraseñas. Esto se logra mediante el uso de consultas parametrizadas y validación de entradas.

Seguridad de Red

La seguridad de red protege las conexiones y comunicaciones de las aplicaciones, evitando que intrusos accedan o interfieran con los datos transmitidos. Este tipo de seguridad es esencial para garantizar que la información que viaja entre usuarios y servidores esté protegida contra intercepciones y alteraciones.

Principales herramientas y técnicas para la seguridad de red:

- **Firewalls:** Controlan y filtran el tráfico de red para bloquear accesos no autorizados.
- **Sistemas de detección y prevención de intrusiones (IDS/IPS):** Monitorizan el tráfico para identificar y neutralizar posibles amenazas.
- **VPN (Red Privada Virtual):** Encripta las conexiones para proteger la privacidad en redes públicas.

Ejemplo práctico:
Un sistema de correo electrónico corporativo utiliza una VPN para proteger la transmisión de mensajes entre empleados, evitando que un atacante en una red pública intercepte las comunicaciones. Además, emplea un firewall para filtrar el tráfico entrante y saliente, bloqueando intentos de acceso no autorizados.

Relación entre los Tipos de Seguridad

Aunque los tipos de seguridad se enfocan en áreas específicas, funcionan de manera conjunta para garantizar una protección integral del software. Por ejemplo, una aplicación de banca móvil puede usar medidas de seguridad de datos para proteger información confidencial, seguridad de aplicaciones para prevenir ataques al código y seguridad de red para asegurar la conexión entre los usuarios y los servidores.

Importancia de los Tipos de Seguridad en el Desarrollo de Software

La implementación de estos tipos de seguridad no solo protege a las organizaciones de pérdidas financieras y daños reputacionales, sino que también cumple con las regulaciones legales y genera confianza en los usuarios. En un entorno tecnológico donde las amenazas son cada vez más sofisticadas, asegurar el software es una prioridad que no puede pasarse por alto.

En conclusión, la seguridad de software abarca múltiples dimensiones, cada una esencial para prevenir riesgos y garantizar que los sistemas sean resilientes frente a las amenazas actuales. Al combinar seguridad de datos, aplicaciones y redes, los desarrolladores y las organizaciones pueden construir soluciones confiables y robustas.

3.4.2 Seguridad en el ciclo de desarrollo de software

Integración de la Seguridad en el Ciclo de Vida del Desarrollo de Software

La seguridad de software debe ser un componente integral en cada fase del ciclo de vida del desarrollo, desde la concepción inicial hasta el mantenimiento posterior. Asegurar que el software esté protegido contra amenazas y vulnerabilidades requiere un enfoque proactivo en cada etapa, adaptando las prácticas de seguridad para satisfacer las necesidades y riesgos específicos de cada fase. A continuación, se describe cómo se integra la seguridad en cada fase del ciclo de vida del desarrollo:

Análisis: Identificación de Riesgos y Definición de Requisitos de Seguridad

La primera fase del ciclo de vida del software es crucial para establecer las bases de la seguridad. En esta etapa, los equipos de desarrollo deben identificar los posibles riesgos y amenazas que podrían afectar al software, así como definir los requisitos de seguridad que guiarán todo el proceso de desarrollo.

Actividades clave:

- **Análisis de riesgos:** Identificar posibles amenazas (por ejemplo, ataques de inyección SQL, pérdida de datos) y evaluar su impacto y probabilidad.

- **Definición de requisitos de seguridad:** Establecer los requisitos fundamentales de seguridad, como la confidencialidad, integridad, disponibilidad y autenticidad de la información, y cómo se abordarán en el diseño del software.
- **Cumplimiento de normativas y estándares:** Asegurarse de que el software cumpla con las normativas de seguridad relevantes, como el **Reglamento General de Protección de Datos (GDPR)**, **ISO/IEC 27001**, o leyes nacionales de protección de datos.

Ejemplo práctico:

Si se está desarrollando un sistema de atención médica en línea, el análisis debe identificar los riesgos asociados con la privacidad de los datos de los pacientes y garantizar que el sistema cumpla con las normativas de protección de datos de salud, como la **HIPAA** en Estados Unidos o regulaciones similares en otras regiones.

Diseño: Implementación de Arquitecturas Seguras

El diseño del software debe incorporar prácticas que aseguren que el sistema sea resistente a posibles amenazas. En esta fase, se crean las arquitecturas del sistema que soportarán las funcionalidades del software, y deben diseñarse para minimizar las vulnerabilidades desde el inicio.

Actividades clave:

- **Diseño de una arquitectura segura:** Adoptar patrones de diseño y arquitecturas que refuercen la seguridad, como el diseño basado en capas, separación de privilegios y control de acceso.
- **Integración de controles de seguridad:** Incorporar controles como cifrado, autenticación, autorización y auditoría en la arquitectura del sistema.
- **Principio de menor privilegio:** Limitar los permisos de los usuarios y servicios a lo estrictamente necesario para reducir las posibilidades de abuso.

Ejemplo práctico:

En el diseño de un sistema de gestión empresarial, se debe implementar un modelo de arquitectura que segmente los módulos sensibles (como el módulo de recursos humanos) y limite el acceso solo a usuarios autorizados mediante autenticación robusta y roles definidos.

Codificación: Uso de Estándares Seguros de Programación

Durante la fase de codificación, el código debe desarrollarse siguiendo estándares de programación seguros que prevengan las vulnerabilidades comunes. Es fundamental que los desarrolladores implementen prácticas y técnicas que garanticen la seguridad del código fuente.

Actividades clave:

- **Adopción de buenas prácticas de codificación segura:** Evitar la introducción de vulnerabilidades como inyecciones SQL, desbordamientos de búfer y errores de validación de entradas.
- **Uso de bibliotecas y marcos de trabajo seguros:** Utilizar librerías y frameworks que ofrezcan mecanismos de seguridad integrados, como **OWASP ESAPI (Enterprise Security API)**, para facilitar la implementación de buenas prácticas de seguridad.
- **Revisión de código:** Realizar revisiones de código y auditorías para detectar posibles errores de seguridad.

Ejemplo práctico:

Al programar una aplicación de comercio electrónico, los desarrolladores deben seguir prácticas como la validación de entradas para evitar ataques de inyección SQL, y utilizar funciones de cifrado para almacenar de manera segura las contraseñas de los usuarios.

Pruebas: Validación de la Robustez Frente a Ataques

Las pruebas son una fase crítica para verificar que el software sea robusto frente a ataques. En esta etapa, se validan los controles de seguridad implementados durante las fases anteriores, buscando posibles vulnerabilidades que podrían ser explotadas por atacantes.

Actividades clave:

- **Pruebas de penetración (pentesting):** Simular ataques cibernéticos para identificar vulnerabilidades en el sistema.
- **Pruebas de seguridad estática (SAST):** Analizar el código fuente en busca de vulnerabilidades antes de la ejecución.
- **Pruebas de seguridad dinámica (DAST):** Evaluar el comportamiento del sistema en ejecución para identificar posibles fallos de seguridad en tiempo real.
- **Pruebas de caja negra y caja blanca:** Realizar pruebas tanto desde la perspectiva del usuario final (caja negra) como del desarrollador (caja blanca) para evaluar las diferentes dimensiones de seguridad.

Ejemplo práctico:

Antes de lanzar un sistema de pagos en línea, un equipo de seguridad realiza pruebas de penetración para evaluar cómo el sistema maneja ataques de fuerza bruta, manipulación de cookies y otros métodos que intentan vulnerar las autenticaciones y transacciones.

Despliegue y Mantenimiento: Actualizaciones Regulares y Monitoreo Continuo

Una vez que el software se ha desplegado en producción, la seguridad no termina, sino que debe mantenerse activa. La fase de despliegue y mantenimiento implica realizar actualizaciones periódicas para corregir vulnerabilidades emergentes y garantizar que el sistema siga funcionando de manera segura a medida que evolucionan las amenazas.

Actividades clave:

- **Actualizaciones y parches:** Aplicar parches de seguridad regularmente para corregir vulnerabilidades que se descubren después del lanzamiento del software.
- **Monitoreo continuo:** Implementar sistemas de monitoreo y alertas para detectar actividad sospechosa o intrusiones en tiempo real.
- **Gestión de incidentes:** Tener un plan de respuesta ante incidentes de seguridad para minimizar los daños en caso de que se produzca una violación de seguridad.

Ejemplo práctico:

En el caso de un sistema de gestión de inventarios, el mantenimiento implica la actualización regular de las bibliotecas de software para abordar vulnerabilidades conocidas y el monitoreo continuo de la red para detectar intentos de acceso no autorizado al sistema.

Integrar la seguridad a lo largo de todas las fases del ciclo de vida del desarrollo de software es esencial para crear sistemas confiables y robustos. Desde la identificación de riesgos en la fase de análisis hasta el monitoreo constante después del despliegue, cada fase debe incorporar medidas de seguridad que aseguren la protección de los datos, las aplicaciones y las redes. Este enfoque no solo previene vulnerabilidades, sino que también asegura el cumplimiento de normativas y genera confianza en los usuarios, lo que es crucial en un mundo cada vez más dependiente de la tecnología.

3.4.3 Confiabilidad, disponibilidad e integridad en el desarrollo de software

La confiabilidad, disponibilidad e integridad son tres atributos clave en el desarrollo de software que aseguran que las aplicaciones sean robustas, eficaces y seguras. Estos principios no solo afectan el

funcionamiento técnico de un software, sino también la confianza de los usuarios y la satisfacción de las expectativas comerciales. A continuación, se detalla cada uno de estos conceptos y su importancia en el ciclo de vida del software.

Confiabilidad del Software

La **confiabilidad** se refiere a la capacidad del software para funcionar de manera consistente y sin fallos dentro de un entorno definido, cumpliendo las especificaciones y comportándose como se espera, incluso en condiciones adversas. Un software confiable es aquel que se puede ejecutar durante largos períodos sin interrupciones o fallos, brindando una experiencia de usuario estable y predecible.

Aspectos clave para garantizar la confiabilidad:

- **Pruebas exhaustivas:** Realizar pruebas de rendimiento, estrés y carga para verificar que el software pueda manejar situaciones de uso intensivo y extremas sin fallar.
- **Manejo de errores adecuado:** Incorporar técnicas de manejo de errores eficaces para evitar que fallos menores provoquen fallos más grandes en el sistema.
- **Sistemas de recuperación:** Implementar mecanismos de recuperación ante fallos, como **backups automáticos** o sistemas de tolerancia a fallos.

Ejemplo práctico:

En una plataforma de comercio electrónico, la confiabilidad es esencial para garantizar que el sistema funcione correctamente durante las temporadas altas de ventas, como el Black Friday, sin que se presenten errores o caídas del servicio, lo que podría generar pérdidas económicas o dañar la reputación de la empresa.

Disponibilidad del Software

La **disponibilidad** se refiere a la capacidad de un software para estar accesible y funcional cuando se necesita, sin interrupciones innecesarias. La disponibilidad es crucial en aplicaciones críticas como las de atención médica, sistemas financieros o plataformas de e-commerce, donde el tiempo de inactividad puede tener consecuencias serias.

Aspectos clave para garantizar la disponibilidad:

- **Redundancia:** Implementar sistemas redundantes, como servidores o bases de datos duplicadas, para asegurar que si un componente falla, otro pueda tomar su lugar sin interrumpir el servicio.
- **Mantenimiento preventivo:** Realizar mantenimiento regular del sistema para identificar y corregir posibles problemas antes de que afecten la disponibilidad.

- **Escalabilidad:** Asegurar que el sistema pueda manejar un aumento en la carga de usuarios o solicitudes sin afectar su disponibilidad.

Ejemplo práctico:

En el caso de un banco en línea, la disponibilidad es esencial para garantizar que los usuarios puedan acceder a sus cuentas y realizar transacciones a cualquier hora, todos los días, sin interrupciones debido a fallos en los servidores o en el sistema.

Integridad del Software

La **integridad** se refiere a la capacidad del software para proteger los datos contra alteraciones o modificaciones no autorizadas, asegurando que la información se mantenga precisa y consistente durante todo su ciclo de vida. La integridad de los datos es crucial para mantener la confianza en el sistema, especialmente en aplicaciones que manejan información sensible como registros financieros, datos personales o registros médicos.

Aspectos clave para garantizar la integridad:

- **Cifrado de datos:** Utilizar cifrado para proteger los datos tanto en tránsito como en reposo, evitando que los atacantes o usuarios no autorizados puedan modificar los datos.
- **Controles de acceso:** Implementar un sistema de control de acceso para asegurarse de que solo usuarios autorizados puedan modificar los datos.
- **Auditoría y seguimiento:** Registrar todas las modificaciones realizadas sobre los datos para poder auditar y verificar su integridad, y detectar posibles fraudes o errores.

Ejemplo práctico:

En un sistema de gestión de inventarios, la integridad es esencial para asegurar que los registros de los productos (por ejemplo, cantidades y precios) no se alteren sin autorización. Si un usuario malintencionado pudiera modificar estos datos, esto podría generar pérdidas financieras o problemas logísticos graves.

Relación de Confiabilidad, Disponibilidad e Integridad en el Desarrollo de Software

Estos tres principios están estrechamente relacionados y son fundamentales para la creación de un software robusto y confiable. La **confiabilidad** asegura que el software funcionará correctamente bajo condiciones esperadas, la **disponibilidad** garantiza que el software esté accesible cuando se necesita y la **integridad** protege la información contra alteraciones no autorizadas. Un software que no sea confiable, no esté

disponible cuando se necesite o cuya integridad de datos sea comprometida, no solo será ineficaz, sino que también pondrá en riesgo la seguridad y la reputación de la empresa que lo desarrolla.

Ejemplo integrador:

En el caso de un sistema de salud en línea, la confiabilidad es vital para asegurar que el sistema esté operativo todo el tiempo, la disponibilidad es crucial para que los profesionales de la salud puedan acceder a los registros de los pacientes en cualquier momento, y la integridad es esencial para garantizar que los datos médicos no sean alterados o manipulados de manera incorrecta.

3.4.4 Gestión de pruebas completas de seguridad de software

Implementación de Pruebas Específicas para Garantizar la Seguridad del Software

La seguridad del software no puede dejarse al azar, especialmente en sistemas que manejan información crítica o servicios sensibles. Para garantizar la seguridad, es esencial implementar una serie de pruebas y técnicas especializadas durante todo el ciclo de vida del software. A continuación, se describen cuatro enfoques clave para evaluar y mejorar la seguridad de un software.

Análisis de Vulnerabilidades

El **análisis de vulnerabilidades** es el proceso de identificar debilidades dentro de un sistema o aplicación que podrían ser explotadas por atacantes. Esto se realiza mediante el uso de herramientas automáticas que escanean el software en busca de fallos de seguridad conocidos o configuraciones incorrectas que puedan abrir brechas.

Técnicas comunes en el análisis de vulnerabilidades:

- **Escaneo automatizado:** Utilización de herramientas como **Nessus** o **OpenVAS** para realizar escaneos de vulnerabilidades. Estas herramientas identifican fallos de seguridad comunes, como configuraciones erróneas, puertos abiertos innecesarios o software desactualizado con vulnerabilidades conocidas.
- **Revisión manual:** Los expertos en seguridad realizan una auditoría manual del código y la infraestructura para identificar posibles vulnerabilidades que las herramientas automáticas puedan haber pasado por alto.

Ejemplo práctico:

En una plataforma de banca en línea, un análisis de vulnerabilidades puede detectar que una API externa utilizada por la plataforma tiene configuraciones inseguras que permiten a los atacantes ejecutar comandos remotos, lo que podría permitirles comprometer los datos de los usuarios.

Pruebas de Penetración (Pen-Testing)

Las **pruebas de penetración** o **pen-testing** son simulaciones controladas de ataques cibernéticos que tienen como objetivo evaluar la defensa del software ante amenazas externas. Durante un pen-test, los especialistas en seguridad intentan explotar las vulnerabilidades del sistema de la misma manera en que lo haría un atacante real.

Objetivos de las pruebas de penetración:

- **Explotar debilidades:** Evaluar qué tan fácilmente un atacante puede obtener acceso no autorizado al sistema.
- **Evaluación de la respuesta a incidentes:** Analizar cómo reaccionan los sistemas y los equipos de seguridad ante un ataque simulado.
- **Evaluación de políticas de seguridad:** Verificar si las políticas de seguridad implementadas son efectivas para detener ataques.

Ejemplo práctico:

En una aplicación de red social, un equipo de pruebas de penetración podría intentar realizar un **ataque de inyección SQL** para verificar si los datos de los usuarios pueden ser comprometidos a través de entradas maliciosas en formularios de usuario, asegurándose de que se utilicen las mejores prácticas de validación y escape de entradas para prevenir estos ataques.

Escaneo de Código (Code Scanning)

El **escaneo de código** implica la revisión automática y manual del código fuente en busca de errores de programación que puedan comprometer la seguridad del software. Esta práctica es esencial para detectar fallos que podrían no ser visibles durante las pruebas tradicionales, como errores de implementación que permiten la ejecución de código malicioso o violaciones de principios de seguridad.

Técnicas comunes en escaneo de código:

- **Análisis estático de código (SAST):** Herramientas como **SonarQube** o **Checkmarx** escanean el código fuente para identificar posibles errores de seguridad, como variables mal manejadas, falta de validación de entradas o errores de lógica que podrían ser explotados.

- **Análisis dinámico de código (DAST):** Se ejecutan pruebas en tiempo real sobre el software en ejecución para detectar vulnerabilidades durante su funcionamiento. Este enfoque ayuda a identificar problemas de seguridad en la interfaz de usuario o en la interacción del software con otros sistemas.

Ejemplo práctico:

En un sistema de gestión de datos personales, un escaneo de código puede identificar que los datos de los usuarios se almacenan en texto plano, lo que puede comprometer la privacidad de los usuarios si un atacante obtiene acceso a la base de datos.

Monitoreo Continuo

El **monitoreo continuo** implica supervisar la actividad del software y el entorno operativo en tiempo real para detectar amenazas y anomalías de seguridad. Este enfoque es fundamental para detectar ataques que no pudieron ser prevenidos mediante pruebas anteriores, así como para identificar vulnerabilidades emergentes.

Técnicas de monitoreo continuo:

- **Análisis de tráfico en red:** Herramientas como **Wireshark** o **Splunk** analizan el tráfico de red para detectar patrones inusuales que puedan indicar un ataque, como ataques DDoS o intentos de acceso no autorizado.
- **Monitoreo de logs:** Se revisan los registros de actividad del software para identificar eventos sospechosos, como intentos de login fallidos o cambios no autorizados en archivos de configuración.
- **Alertas automatizadas:** Herramientas como **Prometheus** o **Nagios** pueden generar alertas en tiempo real si se detectan comportamientos anómalos que sugieren una brecha de seguridad.

Ejemplo práctico:

En un sistema de comercio electrónico, el monitoreo continuo puede detectar un aumento en el número de intentos fallidos de inicio de sesión, lo que podría indicar un intento de **ataque de fuerza bruta** para adivinar contraseñas, activando una alerta para que los administradores tomen medidas preventivas, como bloquear la cuenta del atacante.

ELABORACIÓN, REVISIÓN Y APROBACIÓN DE PARES	
Profesor(a)	
Mg. Martha Janina Duarte Mora	
Fecha de elaboración: 09/3/2024	
Comisión de revisión de pares de guías de estudio del Instituto Superior Tecnológico Tena	
Lcdo. Segundo Calisto Rochina Chileno	Mg. Alvaro Santiago Toalombo Díaz
Mg. Henry Fabian Chango Chango	Ing. Agustin Gonzalo Guanipatin Ramirez
Fecha de revisión: 29/03/2024	
Coordinador de Investigación, Desarrollo Tecnológico e Innovación	
Mg. Danilo Alexander Zamora Núñez	
Fecha de aprobación: 08/04/2024	