Washizaki-SWEBOK-KA Y Area de conocimiento-IDS1

Guía de Estudio: Ingeniería de Software

Esta guía se estructura en grandes bloques temáticos que abarcan desde los procesos fundamentales de desarrollo hasta los fundamentos teóricos y prácticos que sustentan la ingeniería de software. Se busca que, agrupando conceptos relacionados, la información resulte más fácil de consultar, estudiar y recordar.

Índice de Contenidos

- Gestión de la Ingeniería de Software (Cap. 9)
- Proceso de Ingeniería de Software (Cap. 10)
- Modelos y Métodos de Ingeniería de Software (Cap. 11)
- Calidad del Software (Cap. 12)
- Seguridad del Software (Cap. 13)
- Práctica Profesional (Cap. 14)
- Ingeniería Económica del Software (Cap. 15)

3. Fundamentos y Bases Teóricas
    - Fundamentos de la Computación (Cap. 16)
    - Fundamentos Matemáticos (Cap. 17)
    - Fundamentos de Ingeniería (Cap. 18)
4. Definiciones y Respuestas Clave

Procesos y Actividades en el Desarrollo de Software

Esta sección abarca los procesos prácticos que se llevan a cabo en un proyecto de software, desde la definición inicial hasta la gestión de cambios y la planificación del proyecto.

1. Requisitos

La **ingeniería de requisitos** se centra en la recolección, análisis, especificación y validación de lo que el software debe cumplir. Es la base para comprender las necesidades tanto del usuario como del negocio.

- **Actividades clave:**
    - Identificar necesidades del usuario y del negocio.
    - Documentar requisitos funcionales (qué debe hacer el sistema) y no funcionales (rendimiento, seguridad, usabilidad).
    - Gestionar cambios en los requisitos a lo largo del desarrollo.
    - Validar los requisitos con los stakeholders (usuarios, clientes).

Ejemplo:

> Un equipo de desarrollo se reúne con el cliente para aclarar cómo debe comportarse un nuevo módulo de la aplicación.

---

2. Diseño

El **diseño de software** define la arquitectura, componentes, interfaces y estructura del sistema antes de su implementación. Es la fase en la que se planifica cómo se va a resolver técnicamente el problema.

- **Actividades clave:**
  - Definir la arquitectura del sistema (por ejemplo, cliente-servidor, microservicios).
  - Diseñar la estructura de bases de datos.
  - Definir patrones de diseño y estándares de codificación.
  - Crear diagramas UML (diagrama de clases, de secuencia, de componentes).

Ejemplo:

> Un arquitecto de software determina que la aplicación web usará una API REST con una base de datos PostgreSQL.

---

3. Construcción

La **construcción** es el proceso de codificación y desarrollo del software, transformando el diseño y los requisitos en un producto funcional.

- **Actividades clave:**
  - Escribir código en el lenguaje de programación seleccionado.
  - Realizar pruebas unitarias básicas.
  - Integrar el código con otros módulos o sistemas.
  - Aplicar buenas prácticas de desarrollo (uso de patrones, modularidad).

Ejemplo:

> Un programador implementa una nueva funcionalidad en Java y la prueba antes de integrarla en el repositorio.

---

4. Pruebas

La fase de **pruebas** se encarga de verificar que el software funciona correctamente y cumple con los requisitos definidos, garantizando la calidad del producto.

- **Actividades clave:**
  - Diseñar y ejecutar pruebas unitarias, de integración, de sistema y de aceptación.
  - Usar herramientas de automatización de pruebas (Selenium, JUnit, Jest).
  - Reportar y corregir errores (bugs).
  - Validar que el software sea robusto y no tenga fallos críticos antes del despliegue.

Ejemplo:

> Un tester ejecuta pruebas automatizadas para verificar que una aplicación web funcione correctamente en diferentes navegadores.

---

5. Gestión de Configuración

La **gestión de configuración** administra y controla las versiones del software, asegurando que todos los cambios sean trazables y organizados.

- **Actividades clave:**
  - Uso de control de versiones (Git, SVN).
  - Administración de cambios en el código fuente.
  - Manejo de ramas y fusiones en el desarrollo colaborativo.
  - Automatización de despliegues y gestión de dependencias.

Ejemplo:

> Un desarrollador realiza un commit de su código en GitHub y crea una nueva versión del sistema.

---

6. Gestión de la Ingeniería de Software

Esta área se enfoca en la planificación, organización y supervisión de proyectos de software, asegurando el cumplimiento de costos, tiempos y calidad en el producto final.

- **Actividades clave:**
  - Planificación de proyectos de software.
  - Asignación de tareas y gestión de equipos.
  - Uso de metodologías ágiles (Scrum, Kanban) o tradicionales (Cascada).
  - Evaluación de riesgos y mejora continua.

Ejemplo:

> Un Scrum Master lidera una reunión diaria para coordinar tareas y resolver bloqueos en el equipo.

Resumen de las Áreas (Tabla Comparativa)

Área	Descripción	Ejemplo
Requisitos	Definir qué debe hacer el software.	Reunión con el cliente para entender necesidades.
Diseño	Planificar la estructura del sistema.	Crear diagramas UML para modelar la aplicación.
Construcción	Escribir y desarrollar el código.	Programar una funcionalidad nueva en Python.
Pruebas	Verificar que el software funcione correctamente.	Ejecutar pruebas automatizadas con Selenium.
Gestión de Configuración	Administrar versiones y cambios en el software.	Usar Git para controlar cambios en el código.
Gestión de Ingeniería de Software	Planificar y supervisar el proyecto.	Organizar reuniones diarias con el equipo de desarrollo.

Perspectivas Técnicas y Organizacionales

Esta sección profundiza en los conceptos, métodos y enfoques organizativos que dan forma a la ingeniería de software, abarcando desde los requisitos y la arquitectura hasta la calidad, seguridad y gestión económica.

Capítulo 1. Software Requirements (Requisitos de Software)

- **Definición:**

  Los requisitos de software especifican las necesidades y restricciones de un producto o proyecto, así como las actividades para desarrollarlos y mantenerlos a lo largo del ciclo de vida.

- **Perspectivas clave:**
  - **Funcional:** Qué debe hacer el software.
  - **No funcional:** Restricciones tecnológicas y aspectos de calidad del servicio.
- **Problemas comunes:**
  - **Incompletitud:** Necesidades no identificadas.
  - **Ambigüedad:** Requisitos interpretables de múltiples formas.
- **Impacto:**

  Errores en los requisitos pueden generar costos exponenciales en mantenimiento, por lo que una documentación clara es crucial para comunicar la intención del software a futuros mantenedores.

Capítulo 2. Software Architecture (Arquitectura de Software)

- **Definición:**

  La arquitectura de software representa las estructuras fundamentales de un sistema, incluyendo elementos, relaciones y propiedades, para facilitar su análisis y construcción.

- **Evolución:**

  Se ha diferenciado del diseño tradicional por enfocarse en decisiones estructurales críticas y en el entorno completo del sistema (hardware, usuarios y otros sistemas).

- **Componentes clave:**

    - **Vistas y puntos de vista:** Representaciones para diferentes stakeholders (por ejemplo, lógica, procesos, despliegue).
    - **Estilos y patrones:** Soluciones reutilizables como cliente-servidor o microservicios.
- **Objetivo:**

  Proporcionar una visión compartida que guíe el diseño, construcción y evolución del sistema.

Capítulo 3. Software Design (Diseño de Software)

- **Definición:**

  El diseño transforma los requisitos en especificaciones implementables, organizando componentes, interfaces y comportamientos.

- **Etapas:**

  1. **Diseño arquitectónico:** Definición de la estructura global del sistema.
  2. **Diseño de alto nivel:** Interacción entre los componentes principales.
  3. **Diseño detallado:** Especificación interna de módulos y algoritmos.

- **Principios clave:**

  Abstracción, encapsulación, bajo acoplamiento y alta cohesión, además de la importancia de la documentación para facilitar construcción, pruebas y mantenimiento.

- **Enfoques comunes:**

  Orientado a objetos, basado en componentes o dirigido por eventos.

Capítulo 4. Software Construction (Construcción de Software)

Introducción:

Esta fase implica la creación y mantenimiento detallado del software mediante codificación, verificación, pruebas unitarias, integración y depuración.

- **Relación con otros KA:**
    - Se relaciona estrechamente con el diseño (parte del diseño se realiza durante la construcción).
    - Integra pruebas unitarias y de integración.

    - Produce elementos que forman parte de la gestión de configuración (código, documentación).
    - La calidad del código es el entregable final, por lo que su excelencia es crítica.
- **Fundamentos clave:**
    1. **Minimizar la complejidad:** Escribir código simple y legible.
    2. **Anticipar cambios:** Enfoque ágil y uso de prácticas como DevOps.
    3. **Construir para la verificación:** Facilitar la detección de fallos.
    4. **Reutilizar activos:** Uso de bibliotecas, frameworks y COTS.
    5. **Estándares en construcción:** Definir lenguajes, convenciones y manejo de excepciones.

Ejemplo de pregunta:

¿Qué técnica se usa para reducir la complejidad durante la construcción?

→ **Respuesta:** Modularidad, estándares de codificación y herramientas de análisis de complejidad ciclomática.

Capítulo 5. Software Testing (Pruebas de Software)

Introducción:

El testing es la validación dinámica que comprueba que el sistema se comporta según lo esperado.

- **Componentes clave:**
    - **Sistema bajo prueba (SUT):** Puede ser un módulo, aplicación o servicio.
    - **Caso de prueba:** Define entradas, condiciones y resultados esperados.
    - **Selección finita:** Debido a la imposibilidad de probar todas las combinaciones, se busca un balance entre recursos y cobertura.
- **Objetivos:**
    - **Verificación:** Asegurar el cumplimiento de las especificaciones.
    - **Validación:** Confirmar que el producto satisface las necesidades del usuario.

    - **Detección de fallos:** Identificar errores antes de la entrega.
- **Tipos de pruebas:**
    - **Unitarias:** Prueban componentes aislados.
    - **Integración:** Evalúan la interacción entre componentes.
    - **Sistema:** Verifican requisitos no funcionales como seguridad y rendimiento.
    - **Aceptación:** Realizadas por el usuario final.

Ejemplo de pregunta:

¿Qué nivel de prueba evalúa requisitos no funcionales como la seguridad?

→ **Respuesta:** Pruebas de sistema.

Capítulo 6. Software Engineering Operations (Operaciones de Ingeniería de Software)

Introducción:

Esta área se encarga de las actividades necesarias para desplegar, operar y dar soporte al software en entornos productivos, asegurando la integridad y estabilidad del sistema.

- **Evolución:**
    - **DevOps:** Integra desarrollo y operaciones para automatizar procesos de CI/CD (Integración Continua/Entrega Continua).
    - **Infraestructura como Código (IaC):** Automatiza la configuración de entornos mediante scripts.
- **Roles clave:**
    - **Ingeniero de Operaciones:** Encargado de servicios operativos como monitoreo y despliegue.
    - **Site Reliability Engineering (SRE):** Se enfoca en la disponibilidad, rendimiento y automatización del sistema.
- **Procesos principales:**
    1. **Planificación:** Considerar capacidad, continuidad y seguridad.
    2. **Entrega:** Gestionar despliegues, cambios y, en caso de ser necesario, realizar rollbacks.

3. **Control:** Monitorear y gestionar incidentes.

- **Herramientas:**

  Uso de contenedores (Docker), orquestadores (Kubernetes) y herramientas de automatización de pruebas.

Ejemplo de pregunta:

¿Qué práctica de DevOps permite entregas frecuentes de software?

→ **Respuesta:** Integración Continua/Entrega Continua (CI/CD).

Capítulo 7. Software Maintenance (Mantenimiento de Software)

- **Definición:**

  Conjunto de actividades para proporcionar soporte costo-efectivo al software en operación, abarcando la corrección de errores, adaptación a nuevos entornos y mejora de funcionalidades.

- **Importancia:**

  Es esencial para la evolución del software después de su implementación; se estima que más del 80% del esfuerzo se destina a mejoras y no solo a correcciones.

  Se integran prácticas de DevOps para lograr entregas continuas y operaciones eficientes.

- **Categorías de mantenimiento:**
  - **Correctivo:** Reparar fallos detectados.
  - **Adaptativo:** Ajustar el software a nuevos entornos, como actualizaciones de sistemas operativos.
  - **Perfectivo:** Mejorar el rendimiento o la mantenibilidad.
  - **Preventivo:** Corregir defectos latentes antes de que se conviertan en problemas críticos.

- **Desafíos:**

  Comprender código heredado, gestionar deuda técnica y alinear los cambios con los objetivos organizacionales.

Capítulo 8. Software Configuration Management (Gestión de Configuración de Software)

- **Definición:**

  Proceso para identificar, controlar y auditar los elementos de configuración (CI) durante todo el ciclo de vida del software, asegurando su integridad y trazabilidad.

- **Actividades clave:**
  - **Identificación de CI:** Documentar componentes críticos como código, documentación y herramientas.
  - **Control de cambios:** Evaluar y aprobar modificaciones mediante un *Configuration Control Board (CCB)*.
  - **Auditorías:** Verificar que los CI cumplen con las especificaciones (auditorías funcionales y físicas).
  - **Gestión de versiones:** Mantener bibliotecas y controlar las releases.
- **Herramientas:**

  Sistemas de control de versiones (Git), herramientas de construcción automatizada (Jenkins) y bases de datos de configuración (CMDB).

- **Relación con la calidad:**

  Garantiza la consistencia entre requisitos, diseño y producto final.

Capítulo 9. Software Engineering Management (Gestión de la Ingeniería de Software)

- **Definición:**

  Involucra la planificación, estimación, medición y control de proyectos de software para entregar productos que satisfagan a los stakeholders.

- **Enfoques:**
  - **Predictivo:** (por ejemplo, modelo en cascada) con planificación detallada al inicio.
  - **Adaptativo:** (por ejemplo, Agile, DevOps) basado en iteraciones cortas y retroalimentación continua.
- **Áreas clave:**

- **Gestión de riesgos:** Identificar y mitigar amenazas, como cambios inesperados en los requisitos o fallos técnicos.
- **Gestión de calidad:** Asegurar atributos críticos como seguridad, rendimiento y mantenibilidad.
- **Medición:** Uso de métricas (como velocidad en Scrum o defectos por iteración) para decisiones informadas.
- **Adquisición de software:** Evaluación de componentes externos (COTS, open source) y gestión de proveedores.

- **Desafíos:**

  Equilibrar creatividad y disciplina, gestionar equipos multidisciplinarios y adaptarse a tecnologías emergentes (por ejemplo, inteligencia artificial y computación en la nube).

Capítulo 10. Proceso de Ingeniería de Software

Introducción:

La ingeniería de software es la disciplina que aplica principios sistemáticos, cuantificables y gestionables al desarrollo, operación y mantenimiento del software.

- **Proceso de Software:**

  Conjunto de actividades interrelacionadas que transforman entradas (requisitos) en salidas (software funcional).

- **Estándares clave:**

  ISO/IEC/IEEE 12207 (ciclo de vida), CMMI (mejora de procesos) y metodologías ágiles (iterativas).

- **Conceptos Clave:**

  1. **Ciclos de Vida:**
     - Modelos predictivos (Cascada, V-Model), iterativos (Espiral, Incremental) y ágiles (Scrum, XP).
     - Integración de DevOps para entregas continuas.
  2. **Gestión de Procesos:**
     Planificación, adaptación, monitorización y mejora continua (ciclo PDCA).

3. **Herramientas:**
   Uso de BPMN, UML y herramientas CASE para modelar y gestionar procesos.

Preguntas de Estudio:

- ¿Qué diferencia un ciclo de vida predictivo de uno ágil?
- ¿Cómo contribuye DevOps a la calidad del software?

Capítulo 11. Modelos y Métodos de Ingeniería de Software

Introducción:

El modelado es la abstracción del software que permite entender, diseñar y comunicar sistemas complejos. Se emplean diversos métodos para estructurar y formalizar el desarrollo.

- **Métodos:**
  - Estructurados (análisis funcional), orientados a objetos (UML) o ágiles (prototipos).
- **Conceptos Clave:**
  1. **Tipos de Modelos:**
     - **Estructurales:** Diagramas de clases, componentes.
     - **De Comportamiento:** Diagramas de secuencia, máquinas de estados.
  2. **Métodos Formales:**
     Verificación matemática de requisitos (por ejemplo, Alloy).
  3. **Métodos Ágiles:**
     Enfoques como Scrum y XP que priorizan historias de usuario y pruebas automatizadas.

Preguntas de Estudio:

- ¿Qué ventajas ofrece el modelado UML en el diseño de software?
- ¿Por qué los métodos ágiles priorizan la entrega incremental?

Capítulo 12. Calidad del Software

Introducción:

La calidad del software mide el grado en que éste cumple con requisitos explícitos e implícitos, siguiendo estándares como ISO 25010.

- **Enfoque:**

  Se centra en la prevención de defectos, la verificación (¿se construyó correctamente?) y la validación (¿se construyó lo correcto?).

- **Conceptos Clave:**

  1. **Gestión de Calidad:**
     - **SQA (Aseguramiento de Calidad):** Auditorías y revisiones técnicas.
     - **SQC (Control de Calidad):** Pruebas y análisis (estático y dinámico).
  2. **Métricas:**
     Densidad de defectos, tiempo medio entre fallos (MTBF).
  3. **Normativas:**
     ISO 9001 (sistemas de gestión) y CMMI (madurez de procesos).

Preguntas de Estudio:

- ¿Cómo contribuyen las revisiones de código a la calidad del software?
- ¿Qué es un "caso de aseguramiento" (assurance case) en sistemas críticos?

Capítulo 13. Seguridad del Software

Introducción:

La seguridad en el software se centra en proteger la confidencialidad, integridad y disponibilidad de la información (la triada CIA), integrando medidas desde el diseño mismo del sistema (Shift Left).

- **Conceptos Clave:**

  1. **Ciclo de Vida Seguro (SDLC):**
     Incluye análisis de riesgos, modelado de amenazas y pruebas de penetración.
  2. **Estándares:**
     Common Criteria para evaluación de seguridad y OWASP Top 10 para vulnerabilidades web.

3. **Herramientas:**
   Analizadores estáticos (como SonarQube) y herramientas de fuzzing (AFL).

Preguntas de Estudio:

- ¿Qué es un ataque de inyección SQL y cómo prevenirlo?
- ¿Por qué es importante el modelado de amenazas en el diseño seguro?

Capítulo 14. Práctica Profesional en Ingeniería de Software

Introducción:

Se aborda la ética profesional y las habilidades blandas necesarias en el desarrollo de software, haciendo hincapié en la responsabilidad social, legal y moral.

- **Aspectos Clave:**
  - **Ética Profesional:** Basada en códigos de conducta como los de ACM/IEEE.
  - **Habilidades Blandas:** Comunicación, trabajo en equipo y gestión de stakeholders.
  - **Acreditación y Certificación:** Ejemplo: Certificación ISO/IEC 24773 para profesionales.
  - **Aspectos Legales:** Propiedad intelectual, patentes, derechos de autor y normativas como el RGPD.
  - **Documentación:** Manuales técnicos y trazabilidad de requisitos.

Preguntas de Estudio:

- ¿Qué implica el principio de "privacidad por diseño" en el RGPD?
- ¿Cómo gestionar conflictos éticos en proyectos de software?

Capítulo 15. Ingeniería Económica del Software

Introducción:

Esta área integra principios económicos en la toma de decisiones técnicas para maximizar el valor del software, evaluando costos, beneficios y alineación estratégica.

- **Enfoque clave:**
  - Alinear decisiones técnicas con los objetivos estratégicos (rentabilidad, sostenibilidad, innovación).
  - Evaluar propuestas mediante análisis financieros (flujos de efectivo, TCO, ROI, VAN, TIR).
  - Considerar activos intangibles como la cultura organizacional, experiencia y lealtad del cliente.
- **Aplicación:**

  Cubre todo el ciclo de vida del software (desde pre-proyecto hasta retiro) e incluye técnicas de análisis costo-beneficio y optimización.
- **Temas clave:**
  - Valor temporal del dinero, equivalencia de flujos y alternativas mutuamente excluyentes.
  - Toma de decisiones con múltiples criterios (compensatorios vs. no compensatorios).

Ejemplo de pregunta:

"¿A qué KA pertenece decidir si desarrollar o comprar un componente de software?"

Respuesta: Ingeniería Económica del Software (evalúa costos, beneficios y alineación estratégica).

Fundamentos y Bases Teóricas

En esta sección se abordan los conocimientos esenciales en ciencias de la computación, matemáticas y principios de ingeniería que sustentan las prácticas de desarrollo.

Capítulo 16. Fundamentos de la Computación

- **Objetivo:**

  Proporcionar bases técnicas para diseñar y construir software robusto, diferenciando roles y competencias.
- **Enfoque clave:**

    - Diferenciar entre *programador* (implementa código) y *ingeniero de software* (se ocupa de arquitectura, algoritmos y mantenimiento).
    - Dominar conceptos como arquitectura de sistemas, estructuras de datos, algoritmos, redes y seguridad.
- **Temas clave:**
    - **Arquitectura de computadoras:** Modelos Von Neumann, Harvard, RISC vs. CISC.
    - **Estructuras de datos y algoritmos:** Complejidad computacional (notación O grande), técnicas de búsqueda y ordenamiento.
    - **Programación:** Paradigmas (orientado a objetos, distribuido), depuración y estándares de código.
    - **Sistemas operativos:** Gestión de procesos, memoria y dispositivos.
    - **Bases de datos:** Modelos relacionales, normalización y SQL.
    - **Redes:** Modelo OSI, protocolos (TCP/IP) y seguridad.
    - **IA/ML:** Fundamentos de razonamiento y aprendizaje (supervisado y no supervisado).

Ejemplo de pregunta:

"Pruebas y revisiones de software son actividades relacionadas con los..."

Respuesta: Procesos de Verificación y Validación (KA de Calidad del Software).

Capítulo 17. Mathematical Foundations (Fundamentos Matemáticos)

Introducción:

Este capítulo establece las bases matemáticas que permiten a los ingenieros de software traducir lógica precisa en código.

- **Enfoque:**
    - Lógica formal (proposicional y de predicados) y técnicas de demostración (inducción, contradicción, etc.).
    - Teoría de conjuntos, grafos y árboles, fundamentales para estructurar datos y algoritmos.

    - Máquinas de estado finito (FSM), gramáticas formales (CFG, CSG) y teoría de números (divisibilidad, números primos, GCD).
    - Probabilidad discreta, estructuras algebraicas (grupos, anillos), cálculo y precisión numérica.
- **Relevancia:**

  Permite razonar sobre certezas en sistemas, verificar la consistencia lógica del código y abstraer problemas complejos.

Capítulo 18. Engineering Foundations (Fundamentos de Ingeniería)

Introducción:

Aplicación de enfoques sistemáticos y cuantificables de la ingeniería al desarrollo de software.

- **Objetivo:**

  Resolver problemas reales mediante conocimientos científicos y prácticas estandarizadas bajo restricciones de recursos, tiempo y costos.
- **Temas clave:**
    - **Proceso de ingeniería:** Definición de problemas, generación de soluciones, evaluación y mejora continua.
    - **Diseño:** Abordar problemas “mal definidos” (wicked problems) mediante modelos y prototipos.
    - **Métodos empíricos:** Uso de experimentos controlados, estudios observacionales y retrospectivos.
    - **Análisis estadístico:** Muestreo, correlación, regresión y pruebas de hipótesis.
    - **Medición:** Escalas de medición (nominal, ordinal, etc.), fiabilidad, validez y teoría de la medición.
    - **Estándares:** Cumplimiento de normas que garanticen calidad y seguridad.
    - **Industry 4.0:** Integración de IA, IoT y software en procesos industriales.
- **Herramientas:**

RCA (análisis de causa raíz), simulación, modelado y técnicas como el *Goal-Question-Metric*.

Definiciones y Respuestas Clave

Para consolidar el estudio, es útil recordar las siguientes definiciones y respuestas a posibles preguntas:

1. **La Ingeniería de Software es...**

   La aplicación de un enfoque sistemático y cuantificable para desarrollar, operar y mantener software, integrando disciplinas como requisitos, arquitectura y diseño.

2. **Un proceso es...**

   Un conjunto de actividades estructuradas (por ejemplo, requisitos, diseño, pruebas) que transforman necesidades en software funcional, asegurando calidad y eficiencia.

3. **Ingeniería es...**

   Aplicar conocimiento científico y prácticas estandarizadas para resolver problemas reales bajo restricciones de recursos, tiempo y costos.