



INGENIERÍA DE SOFTWARE / ISC – 2020

Esta guía presenta un esquema de los temas que deben ser estudiados en profundidad para realizar el ETC

UNIDAD TEMÁTICA I: Fundamentos

UNIDAD DE COMPETENCIA

Describe el plan de proyecto con base en los fundamentos de ingeniería de software.

I.1.- Los Fundamentos de Ingeniería de Software

Los **Fundamentos de Ingeniería de Software** se refieren a los principios, métodos y prácticas que guían el desarrollo de software de calidad. A continuación, se detallan algunos de los aspectos clave:

1. Definición

- La ingeniería de software es una disciplina que se ocupa de la creación y mantenimiento de software de manera sistemática, controlada y eficiente.

2. Principios Clave

- **Abstracción:** Simplificar problemas complejos al enfocarse en los aspectos más relevantes.
- **Modularidad:** Dividir el software en componentes o módulos independientes que faciliten el desarrollo y mantenimiento.
- **Reusabilidad:** Diseñar componentes que puedan ser utilizados en diferentes aplicaciones.
- **Mantenibilidad:** Facilitar la modificación y actualización del software.
- **Escalabilidad:** Asegurar que el sistema pueda crecer y adaptarse a nuevas necesidades.

3. Fases del Desarrollo de Software

- **Análisis de Requisitos:** Entender lo que los usuarios necesitan.
- **Diseño:** Planificar la arquitectura y los componentes del software.
- **Implementación:** Codificar el software de acuerdo con el diseño.

- **Pruebas:** Verificar que el software funciona correctamente y cumple con los requisitos.
- **Mantenimiento:** Actualizar y corregir el software después de su implementación.

4. Modelos de Desarrollo

- **Cascada:** Un enfoque secuencial donde cada fase debe completarse antes de pasar a la siguiente.
- **Ágil:** Un enfoque iterativo que permite adaptarse a cambios rápidos y fomentar la colaboración continua.
- **DevOps:** Combina desarrollo y operaciones para mejorar la colaboración y la eficiencia en el ciclo de vida del software.

5. Herramientas y Técnicas

- **Control de Versiones:** Herramientas como Git para gestionar cambios en el código.
- **Metodologías de Pruebas:** Técnicas para asegurar la calidad del software.
- **Gestión de Proyectos:** Herramientas y técnicas para planificar y supervisar el progreso del desarrollo.

Estos fundamentos son esenciales para garantizar que el software sea confiable, eficiente y cumpla con las expectativas de los usuarios.

1.2 Plan de Proyecto Software

El **Plan de Proyecto de Software** es un documento que describe cómo se llevará a cabo el desarrollo de un software, incluyendo sus objetivos, recursos, cronograma, y cómo se gestionarán los riesgos y la calidad del producto.

1.2.1 Ámbito de Software

- **Definición:** Se refiere a la delimitación de lo que se incluirá y lo que no se incluirá en el proyecto de software.
- **Componentes:**
 - **Requisitos Funcionales:** Descripción de las funciones que el software debe realizar.
 - **Requisitos No Funcionales:** Criterios de calidad como rendimiento, usabilidad, y seguridad.

- **Exclusiones:** Aspectos que quedan fuera del alcance del proyecto.

1.2.2 Factibilidad y Análisis de Riesgo

- **Factibilidad:** Evaluación de si el proyecto es viable desde diferentes perspectivas:
 - **Técnica:** Si la tecnología necesaria está disponible y es adecuada.
 - **Económica:** Análisis de costos y beneficios para determinar la rentabilidad.
 - **Operativa:** Capacidad de la organización para implementar y mantener el software.
- **Análisis de Riesgo:** Identificación y evaluación de posibles riesgos que puedan afectar el proyecto, incluyendo:
 - **Riesgos Técnicos:** Problemas relacionados con la tecnología.
 - **Riesgos de Gestión:** Desafíos en la planificación y ejecución del proyecto.
 - **Riesgos de Requisitos:** Cambios o malentendidos en los requisitos del cliente.

1.2.3 Métricas y Estimación

- **Métricas:** Herramientas cuantitativas para medir el progreso y la calidad del software, tales como:
 - **Líneas de Código (LOC):** Medida del tamaño del software.
 - **Tasa de Defectos:** Número de defectos encontrados por unidad de tamaño.
- **Estimación:** Proceso de prever el tiempo y los recursos necesarios para completar el proyecto, utilizando técnicas como:
 - **Estimaciones Expertas:** Opiniones de expertos en el área.
 - **Métodos de Puntos de Función:** Evaluación basada en la funcionalidad entregada.

1.2.4 Planificación del Proyecto. Herramientas para la Administración de Proyectos

- **Planificación:** Creación de un cronograma que detalle las actividades, recursos y tiempos necesarios para completar el proyecto.
- **Herramientas:** Software y metodologías que facilitan la administración del proyecto, tales como:
 - **Diagramas de Gantt:** Visualización de tareas y plazos.
 - Centrado en la programación de tareas y su duración.

- Ideal para el seguimiento del progreso de las tareas a lo largo del tiempo.
- **Diagrama PERT:** Analiza tareas y sus interrelaciones, enfocándose en el flujo del proyecto.
 - Centrado en el análisis de las tareas y sus interrelaciones.
 - Utiliza estimaciones de tiempo (optimista, pesimista y más probable) para calcular el tiempo total del proyecto.
 - Más útil en proyectos complejos con múltiples dependencias y donde es crucial entender el camino crítico.

- **Herramientas de Gestión de Proyectos:** Como Jira, Trello o Microsoft Project.

Estas herramientas son complementarias y pueden usarse juntas para una gestión de proyectos más efectiva.

1.2.5 Supervisión y Control del Plan de Proyecto

- **Supervisión:** Monitoreo continuo del progreso del proyecto en comparación con el plan establecido.
- **Control:** Acciones correctivas para abordar desviaciones del plan, que incluyen:
 - **Revisiones de Progreso:** Evaluaciones periódicas del estado del proyecto.
 - **Informes de Estado:** Documentos que comunican el avance y los problemas a las partes interesadas.

Estos elementos son fundamentales para asegurar que el proyecto de software se ejecute de manera eficiente y cumpla con los objetivos establecidos.

1.3 Modelos de Procesos

Los **Modelos de Procesos** son enfoques estructurados que describen las etapas y actividades involucradas en el desarrollo de software. Estos modelos ayudan a planificar, gestionar y controlar el proceso de desarrollo. Algunos de los modelos más comunes incluyen:

- **Modelo en Cascada:** Un enfoque secuencial donde cada fase debe completarse antes de pasar a la siguiente.
- **Modelo Iterativo:** Se desarrollan versiones del software en ciclos, permitiendo revisiones y mejoras continuas.
- **Modelo en Espiral:** Combina elementos del modelo en cascada y del iterativo, enfatizando la gestión de riesgos.

- **Modelo Ágil:** Se centra en la flexibilidad y la colaboración, permitiendo adaptaciones rápidas a los cambios.

1.4 Principios y Metodologías Ágiles

Las **Metodologías Ágiles** son enfoques de desarrollo de software que promueven la flexibilidad, la colaboración y la entrega continua de valor. Se basan en principios como la comunicación constante, la adaptación al cambio y la entrega incremental.

1.4.1 XP (Extreme Programming)

- **Definición:** Una metodología ágil que se centra en mejorar la calidad del software y la capacidad de respuesta a los cambios.

- **Características:**

- **Desarrollo Iterativo:** Se realizan entregas frecuentes de pequeñas funcionalidades.

- **Pruebas Automatizadas:** Se enfatiza la creación de pruebas automatizadas para asegurar la calidad.

- **Programación en Pareja:** Dos desarrolladores trabajan juntos en una sola estación de trabajo para mejorar la calidad del código.

1.4.2 SCRUM

- **Definición:** Un marco ágil que se utiliza para gestionar proyectos complejos y fomentar la colaboración en equipos.

- **Características:**

- **Sprints:** Ciclos de trabajo cortos (generalmente de 2 a 4 semanas) donde se completan funcionalidades específicas.

- **Roles Definidos:** Incluye roles como el Product Owner, Scrum Master y el Equipo de Desarrollo.

- **Reuniones Diarias:** Reuniones breves para coordinar el trabajo y resolver impedimentos.

1.4.3 KANBAN

- **Definición:** Un método ágil que se centra en la visualización del trabajo y la gestión del flujo de tareas.

- **Características:**

- **Tablero Kanban:** Visualiza las tareas en diferentes etapas del proceso (por ejemplo, "Por hacer", "En progreso", "Hecho").

- **Límites de Trabajo en Progreso (WIP):** Se establecen límites sobre cuántas tareas pueden estar en progreso al mismo tiempo para evitar sobrecargas.

- **Mejora Continua:** Se busca optimizar el flujo de trabajo y reducir el tiempo de ciclo.

1.4.4 LEAN

- **Definición:** Una filosofía de gestión que se originó en la manufactura y se aplica al desarrollo de software para maximizar el valor y minimizar el desperdicio.

- **Características:**

- **Enfoque en el Valor:** Se centra en entregar valor al cliente y eliminar actividades que no aportan valor.

- **Flujo Continuo:** Promueve un flujo continuo de trabajo para mejorar la eficiencia.

- **Mejora Continua (Kaizen):** Se fomenta la cultura de mejora continua en todos los niveles del equipo.

Estos modelos y metodologías ayudan a los equipos de desarrollo a ser más eficientes, adaptables y centrados en el cliente, lo que resulta en un software de mayor calidad y una mejor satisfacción del usuario.

UNIDAD TEMÁTICA II: Diseño y construcción

UNIDAD DE COMPETENCIA

Determina el diseño del sistema software a partir del conjunto de requerimientos.

2.1 Ingeniería de Requerimientos

La **Ingeniería de Requerimientos** es el proceso de definir, documentar y gestionar los requisitos del software. Este proceso es fundamental para asegurar que el producto final cumpla con las expectativas de los usuarios y las partes interesadas.

- **Actividades Principales:**

- **Recopilación de Requerimientos:** Entrevistas, encuestas y talleres para entender las necesidades de los usuarios.

- **Análisis de Requerimientos:** Evaluar y priorizar los requerimientos recopilados.

- **Documentación:** Crear documentos de requisitos claros y comprensibles, como Especificaciones de Requerimientos de Software (SRS).

- **Validación:** Asegurarse de que los requisitos sean correctos, completos y viables.

2.2 Diseño del Sistema Software

El **Diseño del Sistema Software** es la fase en la que se define la arquitectura y los componentes del software para cumplir con los requisitos establecidos. Se centra en cómo se estructurará el software y cómo interactuarán sus diferentes partes.

2.2.1 Arquitectura Lógica del Sistema

- **Definición:** Es la estructura fundamental del sistema, que define la organización de los componentes y sus interacciones.
- **Componentes:**
 - **Módulos:** Divisiones del sistema que encapsulan funcionalidades específicas.
 - **Interfaces:** Puntos de interacción entre módulos y con el usuario.
 - **Patrones de Diseño:** Soluciones reutilizables a problemas comunes en el diseño de software.

2.2.2 Interfaz de Usuario

- **Definición:** Es la parte del software con la que los usuarios interactúan directamente.
- **Características:**
 - **Diseño Visual:** Aspecto estético de la interfaz, incluyendo colores, tipografías y disposición de elementos.
 - **Usabilidad:** Facilidad con la que los usuarios pueden navegar y utilizar el software.
 - **Accesibilidad:** Asegurar que la interfaz sea usable por personas con diferentes capacidades.

2.2.3 Experiencia de Usuario (UX)

- **Definición:** Se refiere a la percepción general del usuario al interactuar con el software, abarcando aspectos emocionales y funcionales.
- **Elementos Clave:**
 - **Investigación de Usuarios:** Entender las necesidades y comportamientos de los usuarios.
 - **Prototipos y Pruebas de Usabilidad:** Crear modelos del software y evaluar cómo los usuarios interactúan con ellos.
 - **Iteración:** Mejorar continuamente la experiencia basándose en la retroalimentación de los usuarios.

2.3 Herramientas de Modelado

Las **Herramientas de Modelado** son software que ayudan a los ingenieros de software a crear representaciones visuales de los sistemas, facilitando la comprensión, diseño y documentación.

- **Tipos de Herramientas:**
 - **Modelado UML (Unified Modeling Language):** Herramientas como Lucidchart o Visual Paradigm para crear diagramas de clases, secuencia, y casos de uso.
 - **Herramientas de Prototipado:** Software como Figma o Adobe XD para diseñar y probar interfaces de usuario.
 - **Herramientas de Gestión de Requerimientos:** Como JIRA o Trello, que ayudan a rastrear y gestionar los requisitos a lo largo del ciclo de vida del desarrollo.

Estas fases y herramientas son esenciales para garantizar que el software desarrollado sea de alta calidad, cumpla con los requisitos del cliente y proporcione una buena experiencia al usuario.

UNIDAD TEMÁTICA III: Pruebas de software

UNIDAD DE COMPETENCIA

Diseña el plan de pruebas con base en los distintos tipos de pruebas aplicables al sistema de software.

3.1 Tipos de Pruebas

Las **Pruebas de Software** son actividades que se realizan para verificar que el software cumple con los requisitos especificados y es de alta calidad. Se pueden clasificar en diferentes tipos según su enfoque y objetivos.

3.1.1 Pruebas Funcionales

- **Definición:** Se centran en verificar que el software funcione de acuerdo con los requisitos funcionales especificados.

- **Objetivos:**

- Validar que las funcionalidades del software operen como se espera.
- Asegurar que los resultados sean correctos para diferentes entradas.

- **Ejemplos:**

- **Pruebas de Unidades:** Verifican componentes individuales del software.
- **Pruebas de Integración:** Evalúan la interacción entre diferentes módulos o componentes.
- **Pruebas de Sistema:** Comprueban el sistema completo en su entorno operativo.
- **Pruebas de Aceptación:** Realizadas por el cliente para validar que el software cumple con sus expectativas.

3.1.2 Pruebas No Funcionales

- **Definición:** Se enfocan en evaluar aspectos del software que no están relacionados directamente con las funcionalidades, como el rendimiento, la usabilidad y la seguridad.

- **Objetivos:**

- Medir cómo el software se comporta bajo diferentes condiciones.
- Asegurar que el sistema cumpla con los requisitos de calidad.

- **Ejemplos:**

- **Pruebas de Rendimiento:** Evalúan la velocidad, capacidad de respuesta y estabilidad bajo carga.
- **Pruebas de Usabilidad:** Miden la facilidad de uso y la experiencia del usuario.
- **Pruebas de Seguridad:** Identifican vulnerabilidades y aseguran que el software proteja los datos de los usuarios.

3.2 Herramientas para Pruebas

Las **Herramientas para Pruebas** son aplicaciones que ayudan a automatizar y gestionar el proceso de pruebas, mejorando la eficiencia y efectividad de las mismas.

- **Tipos de Herramientas:**

- **Herramientas de Automatización de Pruebas:** Como Selenium o TestComplete, que permiten automatizar pruebas funcionales y no funcionales.

- **Herramientas de Pruebas de Rendimiento:** Como JMeter o LoadRunner, que simulan cargas de usuarios para evaluar el rendimiento del sistema.

- **Herramientas de Gestión de Pruebas:** Como TestRail o Zephyr, que ayudan a planificar, rastrear y gestionar el proceso de pruebas.

- **Herramientas de Pruebas de Seguridad:** Como OWASP ZAP o Burp Suite, que ayudan a identificar vulnerabilidades en aplicaciones web.

3.3 Evaluación de las Pruebas

La **Evaluación de las Pruebas** es el proceso de revisar y analizar los resultados de las pruebas para determinar la calidad del software y su preparación para el lanzamiento.

-**Aspectos a Evaluar:**

- **Cobertura de Pruebas:** Medir qué porcentaje de los requisitos o del código ha sido probado.
- **Resultados de Pruebas:** Analizar los defectos encontrados, su gravedad y su impacto en el sistema.
- **Efectividad de las Pruebas:** Evaluar si las pruebas han identificado defectos significativos y si han contribuido a mejorar la calidad del software.
- **Retroalimentación:** Recoger comentarios de los testers y usuarios para identificar áreas de mejora en el proceso de pruebas.

La evaluación de las pruebas es crucial para tomar decisiones informadas sobre el lanzamiento del software y para planificar futuras mejoras y pruebas.

UNIDAD TEMÁTICA IV

Calidad y modelos de madurez

UNIDAD DE COMPETENCIA

Evalúa la calidad del sistema software con base en las normas de calidad y madurez.

4.1 Calidad del Proceso Software

La **Calidad del Proceso de Software** se refiere a la capacidad de un proceso de desarrollo de software para producir productos de alta calidad de manera consistente. Esto incluye la planificación, gestión y ejecución de actividades que aseguran que el desarrollo se realice de acuerdo con las mejores prácticas y estándares.

- **Aspectos Clave:**

- **Definición de Procesos:** Establecer procesos claros y documentados que guíen el desarrollo.
- **Mejora Continua:** Implementar prácticas de mejora continua para optimizar los procesos.
- **Medición y Análisis:** Utilizar métricas para evaluar la efectividad de los procesos y hacer ajustes según sea necesario.

4.2 Calidad del Producto Software

La **Calidad del Producto de Software** se refiere a las características y atributos del software final que determinan su capacidad para satisfacer las necesidades de los usuarios y cumplir con los requisitos especificados.

- **Características de Calidad:**

- **Funcionalidad:** El software debe cumplir con los requisitos funcionales.
- **Usabilidad:** Debe ser fácil de usar y entender por los usuarios finales.

- **Rendimiento:** Debe responder de manera eficiente bajo condiciones de carga.
- **Seguridad:** Debe proteger contra accesos no autorizados y vulnerabilidades.
- **Mantenibilidad:** Debe ser fácil de modificar y actualizar.

4.3 Modelos y Normas de Calidad

Los **Modelos y Normas de Calidad** son marcos y estándares que establecen criterios para evaluar y mejorar la calidad en el desarrollo de software.

4.3.1 ISO

- **Definición:** La Organización Internacional de Normalización (ISO) desarrolla estándares internacionales, incluyendo aquellos relacionados con la calidad del software.
- **Ejemplo:** ISO 9001, que establece requisitos para un sistema de gestión de calidad.
- ISO 9001.
- ISO 25000.

4.3.2 IEEE

- **Definición:** El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) proporciona estándares que abordan diversos aspectos del desarrollo de software.
- **Ejemplo:** IEEE 829, que define un estándar para la documentación de pruebas de software.
- IEEE Std 1061-1998

4.3.3 IEC

- **Definición:** La Comisión Electrotécnica Internacional (IEC) se centra en estándares relacionados con la tecnología eléctrica y electrónica, incluyendo software.
- **Ejemplo:** IEC 61508, que aborda la seguridad funcional de sistemas eléctricos y electrónicos.
- ISO/IEC 15939

4.3.4 Otras

- **Ejemplo:** Modelos como CMMI (Capability Maturity Model Integration) y normas específicas de la industria que pueden ser relevantes para la calidad del software.

4.4 Modelos de Madurez

Los **Modelos de Madurez** son marcos que ayudan a las organizaciones a evaluar y mejorar sus procesos de desarrollo de software.

4.4.1 Proceso de Software Personal (PSP)

- **Definición:** Un modelo que ayuda a los desarrolladores individuales a mejorar su proceso de trabajo y la calidad del software que producen.
- **Enfoque:** Se centra en la autoevaluación y la mejora continua de las habilidades y procesos personales.

4.4.2 Proceso de Software de Equipo (TSP)

- **Definición:** Un modelo que extiende el PSP a equipos de desarrollo, promoviendo la colaboración y la mejora de procesos a nivel de equipo.
- **Enfoque:** Se centra en la gestión del equipo y la planificación de proyectos para mejorar la calidad y la productividad.

4.4.3 Modelo de Capacidad de Madurez Integrado (CMMI)

- **Definición:** Un modelo que proporciona un marco para mejorar los procesos de software en organizaciones.
- **Niveles de Madurez:** Incluye cinco niveles que van desde "Inicial" hasta "Optimizado", cada uno con prácticas específicas que las organizaciones pueden implementar para mejorar.

4.4.4 MoProSoft

- **Definición:** Un modelo de proceso para la mejora del software en México, que busca establecer buenas prácticas en el desarrollo de software.
- **Enfoque:** Se centra en la adaptación de procesos a las necesidades específicas de las organizaciones mexicanas, promoviendo la calidad y la eficiencia.

Estos conceptos son fundamentales para garantizar que el software desarrollado sea de alta calidad y cumpla con los estándares y expectativas del mercado.