



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSÍ
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

Reporte Intermedio de trabajo de período sabático del:
ISC GUILLERMO GÁMEZ GALEANA

B.1 Elaboración de Apuntes
para la materia
Fundamentos de Sistemas de Información
IFE-1015

De acuerdo con el proyecto calendarizado en el documento PS-02
“**B.1 Elaboración de Apuntes** elaborado durante el ejercicio del
periodo **Sabático 1/2024**”.

Soledad de Graciano Sánchez, S.L.P., a 1 de agosto de 2024

CARTA DE RECONOCIMIENTO DEL AUTOR DE LOS DERECHOS A FAVOR DEL TECN

Ciudad de México, 01/08/2024

TECNOLÓGICO NACIONAL DE MÉXICO.

PRESENTE.

Bajo protesta de decir verdad, GUILLERMO GÁMEZ GALEANA, personal docente adscrito al INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSÍ del Tecnológico Nacional de México, manifiesto que en cumplimiento de mis actividades relacionadas con el Año Sabático, elaboré la obra titulada "B.1 ELABORACIÓN DE APUNTES PARA LA MATERIA FUNDAMENTOS DE SISTEMAS DE INFORMACIÓN IFE-1015".

Con base en lo anterior, y con fundamento en los artículos 83 de la Ley Federal del Derecho de Autor y 46 de su Reglamento, reconozco que el Tecnológico Nacional de México es titular de los derechos patrimoniales sobre la misma y le corresponden las facultades relativas a la divulgación, integridad de la obra y de colección, conservando el derecho a figurar como autor. Asimismo, respondo por la autoría y originalidad de la citada obra; y relevo de toda responsabilidad al Tecnológico Nacional de México de cualquier demanda o reclamación que llegara a formular alguna persona física o moral que considere que con esta obra es afectado en alguno de los derechos protegidos por la Ley en cita, asumiendo todas las consecuencias legales y económicas.

ATENTAMENTE



ISC GUILLERMO GÁMEZ GALEANA

Contribución académica

La contribución de este documento a los objetivos educativos del programa educativo de la materia de Fundamentos de Sistemas de Información la cual forma parte de la retícula de la Carrera de Ingeniería Informática del Tecnológico Nacional de México, es lograr que el estudiante desarrolle un pensamiento lógico, que entienda y aplique los conceptos de los principios lógicos y funcionales de los sistemas de información para el desarrollo de sistemas, siendo capaz de resolver problemas basados en procesos de la ingeniería. Se busca presentar situaciones problemáticas, asociando éstas con la realidad del estudiante, mejorando su aprendizaje. Con el uso de este material, permitirá clarificar los conceptos básicos de los sistemas de información con el fin de establecer las bases para el desarrollo de sistemas que resuelvan problemas de ingeniería y se establecerá el marco conceptual donde el estudiante y el docente podrán tener una comunicación efectiva. El docente debe hacer ver al estudiante las implicaciones o consecuencias del concepto revisado ya que en la actualidad las implicaciones CienciaTecnología-Sociedad son muy importantes; traduciendo todo esto en un esfuerzo integrador de conocimientos.

Contenido

Introducción	6
Unidad I Introducción a los sistemas de información	7
1.1 Concepto de Sistemas de Información	7
1.2 Dimensiones de los sistemas de información	8
1.3 Clasificación de los sistemas de información	9
Unidad II Ingeniería de software	12
2.1 Definición de ingeniería de software	12
2.2 Capas de la ingeniería de software	14
2.3 Etapas de la ingeniería de software	16
2.3.1 Análisis	18
2.3.2 Diseño	20
2.3.3 Desarrollo	22
2.3.4 Pruebas	24
2.3.5 Implementación	27
2.3.6 Mejora continua	30
Unidad III Modelos prescriptivos del desarrollo de sistemas de información ..	33
3.1 Modelo en cascada	33
3.2 Modelos evolutivos	35
3.3 Modelos especiales	38
3.4 El proceso unificado de desarrollo de software	42
3.5 Modelo de proceso de software IEEE	45
3.6 Herramientas CASE	49
Unidad IV Paradigmas de la ingeniería de software	53
4.1 El enfoque estructurado	54
4.2 El enfoque orientado a objetos	57
4.3 Enfoques de vanguardia	60
Unidad V Gestión de proyectos de sistemas de información	63
5.1 Actividades de gestión	65
5.2 Estudio de viabilidad del proyecto	68
5.3 Análisis Costo/Beneficio	71
5.4 Gestión de riesgos	74
5.5 Planificación del proyecto	78
5.6 Calendarización del proyecto	81
Bibliografía	84

Anexo 1. ACTIVIDADES DE APRENDIZAJE	85
Unidad I	
A1.1.1 Estudio de caso	85
A1.2.1 Ejercicio propuesto	86
A1.3.1 Aprendizaje basado en problemas	87
Unidad II	
A2.3.1.1 Estudio de caso	89
A2.3.2.1 Ejercicio propuesto	94
A2.3.2.2 Aprendizaje basado en problemas	95
Unidad III	
A3.1.1 Estudio de caso	98
A3.1.2 Ejercicio propuesto	99
A3.1.3 Aprendizaje basado en problemas	101
Unidad IV	
A4.1.1 Estudio de caso	106
A4.2.2 Ejercicio propuesto	111
A4.2.3 Aprendizaje basado en problemas	113
Unidad V	
A5.1.1 Estudio de caso	117
A5.5.1 Ejercicio propuesto	124
A5.6.1 Aprendizaje basado en problemas	126
Anexo 2. Plantilla del reporte de actividades	129
Anexo 3. Rúbrica	130
Anexo 4. Instrumentación didáctica	131

Introducción

Una de las preocupaciones más notables y que debe estar presente en cualquier organismo es la mejora continua. Para el caso de las instituciones Educativas, es necesario buscar mecanismos que coadyuven a elevar la calidad y eficiencia educativa. Uno de estos mecanismos lo proporciona precisamente la elaboración de apuntes, ya que contribuye de manera tangible a la adquisición del conocimiento por parte de los estudiantes, siendo, además, un parámetro de control de avance en los programas de estudio al estar unificado en todos los grupos de la misma materia. Con la elaboración de apuntes de la materia de Fundamentos de Sistemas de Información se pretende consolidar los conocimientos adquiridos por el estudiante con la teoría, concretando en un problema real las explicaciones recibidas. Estos conceptos son de gran importancia y deben ser seguidos por todos los estudiantes en forma individual o en equipos reducidos. Con estos apuntes se permite al alumno tener la capacidad de pensar para abordar y resolver nuevos problemas reales, además, representan una herramienta útil de apoyo para el docente, ya que además de contener el material requerido, le permitirá dosificarlo de acuerdo a tiempos y temas.

UNIDAD I INTRODUCCIÓN A LOS SISTEMAS DE INFORMACIÓN

1.1 Concepto de Sistemas de Información

Un **sistema** es un conjunto organizado de elementos que interactúan entre sí para recopilar, procesar, almacenar y distribuir datos con el fin de apoyar la toma de decisiones y el control en una organización. Estos sistemas pueden incluir hardware, software, datos, procedimientos y personal.

Por ejemplo, un sistema de información de una empresa puede incluir computadoras, servidores, bases de datos, software de gestión y empleados que utilizan estos recursos para acceder y actualizar la información necesaria para las operaciones diarias.

La información se define como un conjunto de datos procesados y organizados de manera significativa, que proporciona conocimiento sobre un determinado tema o situación. La información puede ser utilizada para tomar decisiones, resolver problemas o comunicar mensajes.

Sistemas de Información:

Un sistema de información es un conjunto de elementos interrelacionados que recopilan, procesan, almacenan y distribuyen información para apoyar la toma de decisiones y el control en una organización. Estos sistemas pueden ser manuales o automatizados, y están diseñados para mejorar la eficiencia y efectividad de las operaciones de una empresa.

Características de los sistemas de información:

- **Procesamiento de datos:** Los sistemas de información procesan datos para convertirlos en información útil.
- **Almacenamiento de datos:** Almacenan datos de forma segura para su posterior acceso.
- **Distribución de información:** Distribuyen información a los usuarios relevantes en el momento adecuado.
- **Apoyo a la toma de decisiones:** Ayudan a los usuarios a tomar decisiones informadas basadas en la información disponible.
- **Automatización de procesos:** Automatizan tareas y procesos para aumentar la eficiencia operativa.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A1.1.1

1.2 Dimensiones de los sistemas de información.

Las dimensiones de los sistemas de información se refieren a los diferentes aspectos que deben considerarse para comprender completamente cómo funcionan y cómo pueden ser utilizados eficazmente. Estas dimensiones incluyen:

1. Organizacional:

- **Estructura Organizativa:** Cómo está organizada la empresa, incluyendo jerarquías, divisiones y funciones.
- **Procesos de Negocio:** Los procedimientos y flujos de trabajo que se utilizan para realizar tareas específicas.
- **Cultura Organizacional:** Valores, normas y comportamientos que prevalecen en la empresa.
- **Políticas y Normas:** Reglas y regulaciones internas que deben seguirse.

2. Tecnológica:

- **Hardware:** Los dispositivos físicos como servidores, computadoras, dispositivos de almacenamiento, etc.
- **Software:** Aplicaciones y programas que se utilizan para procesar y gestionar información.
- **Redes:** Infraestructura de comunicación que conecta diferentes componentes del sistema de información.
- **Bases de Datos:** Sistemas de almacenamiento y gestión de datos.

3. Humana:

- **Usuarios:** Las personas que utilizan el sistema de información, incluyendo empleados, clientes, proveedores, etc.
- **Desarrolladores y Soporte Técnico:** Profesionales responsables de crear, mantener y solucionar problemas en el sistema.
- **Gestión de TI:** Personal encargado de la administración y supervisión de los recursos de tecnología de la información.

4. Información:

- **Datos:** Materia prima que se procesa para obtener información útil.
- **Información:** Datos procesados y organizados que tienen significado y valor para la toma de decisiones.
- **Conocimiento:** Información interpretada y utilizada para generar valor añadido y ventajas competitivas.

5. Entorno Externo:

- **Competencia:** Empresas competidoras y su influencia en las decisiones de sistemas de información.
- **Regulaciones y Leyes:** Normativas legales que afectan la gestión de la información y la tecnología.
- **Avances Tecnológicos:** Nuevas tecnologías y tendencias que pueden influir en la estrategia de sistemas de información.
- **Factores Económicos y Sociales:** Condiciones económicas y sociales que impactan en la implementación y uso de los sistemas de información.

Estas dimensiones interactúan y se interrelacionan, influyendo en la eficacia y eficiencia de los sistemas de información dentro de una organización.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A1.2.1

1.3 Clasificación de los sistemas de información.

La clasificación de los sistemas de información se puede realizar de diversas maneras, según diferentes criterios. Aquí se presentan algunas de las clasificaciones más comunes:

1. Según el Nivel Organizacional

1. **Sistemas de Procesamiento de Transacciones (TPS):**
 - **Función:** Manejan y registran las transacciones diarias rutinarias y necesarias para el funcionamiento del negocio.
 - **Ejemplo:** Sistemas de puntos de venta, sistemas de inventario.
2. **Sistemas de Información Gerencial (MIS):**
 - **Función:** Proveen información y reportes que ayudan en la toma de decisiones gerenciales.
 - **Ejemplo:** Reportes de ventas, análisis de rendimiento.
3. **Sistemas de Apoyo a las Decisiones (DSS):**
 - **Función:** Ayudan a la toma de decisiones no rutinarias mediante el análisis de datos.
 - **Ejemplo:** Análisis de escenarios, sistemas de previsión.
4. **Sistemas de Información Ejecutiva (EIS):**
 - **Función:** Proveen información crítica para los altos directivos y ejecutivos de una organización.
 - **Ejemplo:** Cuadros de mando integral, análisis de tendencias.
5. **Sistemas de Automatización de Oficinas (OAS):**
 - **Función:** Apoyan las actividades diarias de oficina mediante la automatización de tareas administrativas.
 - **Ejemplo:** Procesadores de texto, sistemas de correo electrónico.

2. Según su Funcionalidad

1. **Sistemas de Información de Contabilidad (AIS):**
 - **Función:** Gestionan la información contable y financiera.
 - **Ejemplo:** Sistemas de contabilidad general, cuentas por pagar y por cobrar.
2. **Sistemas de Información de Recursos Humanos (HRIS):**
 - **Función:** Gestionan la información relacionada con el personal y los recursos humanos.
 - **Ejemplo:** Sistemas de nómina, gestión de talento.
3. **Sistemas de Información de Marketing:**
 - **Función:** Ayudan en la gestión de actividades de marketing y ventas.
 - **Ejemplo:** Sistemas de gestión de relaciones con clientes (CRM).
4. **Sistemas de Información de Producción:**
 - **Función:** Apoyan la planificación y control de la producción.
 - **Ejemplo:** Sistemas de planificación de recursos empresariales (ERP).

3. Según el Tipo de Soporte que Brindan

1. Sistemas Transaccionales:

- **Función:** Soportan las operaciones y transacciones diarias.
- **Ejemplo:** Sistemas de ventas en línea.

2. Sistemas de Control y Monitoreo:

- **Función:** Controlan y monitorean procesos específicos.
- **Ejemplo:** Sistemas de control de calidad.

3. Sistemas de Colaboración:

- **Función:** Facilitan la colaboración y comunicación entre usuarios.
- **Ejemplo:** Plataformas de gestión de proyectos, herramientas de colaboración en línea.

4. Según el Tipo de Arquitectura

1. Sistemas Centralizados:

- **Función:** Toda la información y el procesamiento se manejan en un solo lugar.
- **Ejemplo:** Sistemas de mainframe.

2. Sistemas Distribuidos:

- **Función:** La información y el procesamiento están distribuidos en múltiples ubicaciones.
- **Ejemplo:** Sistemas en la nube.

3. Sistemas en Red:

- **Función:** Los sistemas están interconectados a través de una red, permitiendo la comunicación y el intercambio de datos.
- **Ejemplo:** Intranets corporativas.

5. Según el Usuario Final

1. Sistemas Personales:

- **Función:** Diseñados para ser utilizados por individuos.
- **Ejemplo:** Aplicaciones de productividad personal.

2. Sistemas Departamentales:

- **Función:** Soportan las necesidades de un departamento específico dentro de la organización.
- **Ejemplo:** Sistemas de gestión de ventas.

3. Sistemas Corporativos:

- **Función:** Soportan las necesidades de toda la organización.
- **Ejemplo:** Sistemas de planificación de recursos empresariales (ERP).

Esta clasificación no es exhaustiva y las categorías pueden solaparse, ya que un sistema de información puede pertenecer a más de una categoría dependiendo de su uso y propósito. Estos recursos proporcionan una base sólida para entender los principios y prácticas de la ingeniería de información y su aplicación en el mundo real.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A1.3.1

Referencias

1. Laudon, K.C., & Laudon, J.P. (2020). Management Information Systems: Managing the Digital Firm. Este libro proporciona una visión comprensiva de cómo los sistemas de información se utilizan para apoyar las operaciones y la toma de decisiones en las organizaciones.
2. Stair, R., & Reynolds, G. (2019). Principles of Information Systems. Una introducción a los conceptos fundamentales de los sistemas de información, incluyendo su diseño, desarrollo y gestión.
3. Turban, E., Sharda, R., & Delen, D. (2020). Decision Support and Business Intelligence Systems. Este texto profundiza en cómo los sistemas de soporte de decisiones y las tecnologías de inteligencia empresarial se utilizan para mejorar la toma de decisiones empresariales.
4. O'Brien, J.A., & Marakas, G.M. (2011). Introduction to Information Systems. Proporciona una visión general de los sistemas de información y cómo se utilizan en el entorno empresarial actual.

UNIDAD II INGENIERIA DE SOFTWARE

2.1 Definición de Ingeniería Software

La Ingeniería de Software es una disciplina de la ingeniería que se enfoca en el diseño, desarrollo, mantenimiento, prueba y evaluación del software. Su objetivo principal es aplicar principios de ingeniería y metodologías sistemáticas para la creación de software de alta calidad, eficiente, fiable y que cumpla con las necesidades y expectativas de los usuarios.

Definición Formal

La Ingeniería de Software puede definirse como:

"La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software."

Objetivos de la Ingeniería de Software

1. **Calidad:** Garantizar que el software sea de alta calidad y cumpla con los estándares y requisitos especificados.
2. **Eficiencia:** Asegurar que el software sea eficiente en términos de rendimiento y uso de recursos.
3. **Mantenibilidad:** Facilitar la actualización y modificación del software para adaptarse a nuevas necesidades o corregir errores.
4. **Fiabilidad:** Desarrollar software que funcione de manera consistente y predecible bajo condiciones especificadas.
5. **Usabilidad:** Crear software que sea fácil de usar y entender para los usuarios finales.

Principios de la Ingeniería de Software

1. **Modularidad:** Dividir el software en componentes o módulos manejables que se pueden desarrollar y probar de manera independiente.
2. **Abstracción:** Utilizar niveles de abstracción adecuados para gestionar la complejidad del software.
3. **Encapsulamiento:** Ocultar los detalles internos de los componentes y exponer solo las interfaces necesarias.
4. **Evolución:** Planificar y diseñar el software para facilitar su adaptación y evolución a lo largo del tiempo.
5. **Verificación y Validación:** Asegurar que el software cumple con sus requisitos y funciona correctamente mediante pruebas y revisiones sistemáticas.

Procesos en la Ingeniería de Software

1. **Requisitos:** Captura y análisis de las necesidades y expectativas de los usuarios y stakeholders.
2. **Diseño:** Planificación de la arquitectura y los componentes del software, definiendo cómo interactuarán entre sí.
3. **Implementación:** Desarrollo del código fuente del software.
4. **Pruebas:** Verificación de que el software funciona correctamente y cumple con los requisitos especificados.
5. **Mantenimiento:** Actualización y mejora del software después de su implementación para corregir errores y adaptarse a nuevas necesidades.
6. **Gestión de Proyectos:** Coordinación y administración de recursos, tiempos y actividades para asegurar el éxito del proyecto de software.

Métodos y Herramientas

La Ingeniería de Software utiliza una variedad de métodos y herramientas para facilitar el desarrollo y mantenimiento del software, incluyendo:

- **Metodologías Ágiles:** Scrum, Kanban.
- **Modelos de Desarrollo:** Cascada, Desarrollo Iterativo, Prototipado, Modelo V.
- **Lenguajes de Programación:** Java, Python, C++, etc.
- **Herramientas de Gestión de Proyectos:** JIRA, Trello.
- **Sistemas de Control de Versiones:** Git, Subversion.

En resumen, la Ingeniería de Software es una disciplina integral que combina conocimientos técnicos, principios de ingeniería y metodologías de gestión para producir software de alta calidad que satisface las necesidades de los usuarios y se adapta a los cambios en el tiempo.

2.2 Capas de la ingeniería de software

La Ingeniería de Software se organiza en varias capas, cada una de las cuales aborda diferentes aspectos del desarrollo y mantenimiento del software. Estas capas proporcionan una estructura sistemática y organizada para la creación de software de calidad. Las principales capas de la Ingeniería de Software son:

1. Capa de Proceso

- **Definición:** Esta capa abarca las actividades, métodos y prácticas utilizadas para gestionar y controlar el desarrollo de software.
- **Elementos:**
 - **Modelos de proceso:** Modelos como el Proceso Unificado de Rational (RUP), Modelo en Cascada, Desarrollo Iterativo, Agile, Scrum, Kanban.
 - **Gestión de proyectos:** Planificación, seguimiento, y control de los proyectos de software.
 - **Metodologías:** Conjunto de prácticas y procedimientos estándar para el desarrollo de software.

2. Capa de Métodos

- **Definición:** Esta capa incluye los métodos técnicos y las prácticas que se emplean para el diseño, desarrollo, pruebas y mantenimiento del software.
- **Elementos:**
 - **Análisis de requisitos:** Métodos para la captura y definición de requisitos del software.
 - **Diseño de software:** Técnicas de diseño arquitectónico, de componentes y de interfaces.
 - **Codificación:** Buenas prácticas de programación y estilos de codificación.
 - **Pruebas:** Métodos y técnicas para la verificación y validación del software.
 - **Mantenimiento:** Estrategias y prácticas para la corrección de errores y la actualización del software.

3. Capa de Herramientas

- **Definición:** Esta capa comprende las herramientas y tecnologías que apoyan y automatizan las actividades de las otras capas.
- **Elementos:**
 - **Entornos de Desarrollo Integrado (IDE):** Herramientas como Visual Studio, Eclipse, IntelliJ IDEA.
 - **Sistemas de control de versiones:** Git, Subversion.
 - **Herramientas de gestión de proyectos:** JIRA, Trello, Asana.
 - **Herramientas de construcción y automatización:** Maven, Gradle, Jenkins.

- **Herramientas de pruebas:** JUnit, Selenium, TestNG.
- **Herramientas de documentación:** Javadoc, Doxygen.

4. Capa de Calidad

- **Definición:** Esta capa se centra en garantizar que el software cumpla con los estándares de calidad necesarios.
- **Elementos:**
 - **Aseguramiento de la calidad:** Procesos y actividades para garantizar la calidad en cada fase del desarrollo de software.
 - **Control de calidad:** Inspección y prueba del software para identificar defectos y garantizar que cumpla con los requisitos.
 - **Estándares de calidad:** Normas y prácticas que guían la producción de software de alta calidad (por ejemplo, ISO/IEC 9126, CMMI).
 -

Interrelación de las Capas

- **Proceso:** Define el marco y la secuencia de actividades.
- **Métodos:** Proporcionan los enfoques técnicos y las prácticas específicas dentro del marco del proceso.
- **Herramientas:** Soportan y automatizan las actividades definidas por el proceso y aplicadas mediante los métodos.
- **Calidad:** Asegura que todas las actividades realizadas y los productos resultantes cumplan con los estándares y requisitos de calidad.

Esta estructura en capas permite a los ingenieros de software abordar de manera sistemática y efectiva los desafíos asociados con el desarrollo de software, asegurando que todos los aspectos necesarios se consideren y gestionen adecuadamente.

2.3 Etapas de la ingeniería de software

Las etapas de la ingeniería de software, también conocidas como el ciclo de vida del desarrollo de software, describen el proceso completo desde la concepción del software hasta su retirada. A continuación, se presentan las etapas clave:

1. Análisis de Requisitos

- **Definición:** Recopilación y documentación de las necesidades y expectativas de los usuarios y stakeholders.
- **Actividades:**
 - Entrevistas y reuniones con stakeholders.
 - Análisis de procesos y flujos de trabajo existentes.
 - Documentación de requisitos funcionales y no funcionales.
 - Creación de especificaciones de requisitos del software (SRS).

2. Diseño del Sistema

- **Definición:** Planificación de la arquitectura y los componentes del software, definiendo cómo interactuarán entre sí.
- **Actividades:**
 - Diseño arquitectónico: Definir la estructura global del sistema.
 - Diseño detallado: Especificar los componentes individuales y sus interacciones.
 - Creación de diagramas de diseño (UML, diagramas de flujo, etc.).
 - Selección de tecnologías y plataformas.

3. Desarrollo (Codificación)

- **Definición:** Traducción del diseño en código fuente ejecutable.
- **Actividades:**
 - Programación de componentes y módulos.
 - Integración de componentes.
 - Adopción de buenas prácticas de codificación y estándares.
 - Documentación del código.

4. Pruebas (Testing)

- **Definición:** Verificación y validación del software para asegurar que cumple con los requisitos especificados y funciona correctamente.
- **Actividades:**
 - Pruebas unitarias: Verificación de componentes individuales.
 - Pruebas de integración: Verificación de la interacción entre componentes.
 - Pruebas de sistema: Validación del sistema completo.

- Pruebas de aceptación: Verificación con usuarios finales para asegurar que el software cumple con sus necesidades.
- Pruebas de regresión: Asegurar que los cambios no introducen nuevos errores.

5. Implementación

- **Definición:** Instalación y configuración del software en el entorno de producción.
- **Actividades:**
 - Preparación del entorno de producción.
 - Migración de datos (si es necesario).
 - Instalación del software.
 - Capacitación de usuarios finales.
 - Verificación post-despliegue.

6. Mantenimiento

- **Definición:** Actualización y mejora del software después de su implementación para corregir errores, mejorar el rendimiento o adaptarse a nuevas necesidades.
- **Actividades:**
 - Corrección de errores (mantenimiento correctivo).
 - Mejoras y nuevas funcionalidades (mantenimiento evolutivo).
 - Optimización del rendimiento (mantenimiento perfectivo).
 - Adaptación a nuevos entornos o tecnologías (mantenimiento adaptativo).

7. Mejora Continua (Finalización)

- **Definición:** Retirada del software del uso activo cuando ya no es necesario o es reemplazado por una versión nueva.
- **Actividades:**
 - Notificación a los usuarios sobre el fin de soporte.
 - Migración de datos a nuevos sistemas.
 - Desinstalación del software.
 - Archivar documentación relevante.

Modelos de Ciclo de Vida del Software

Dependiendo del proyecto y las necesidades específicas, se pueden utilizar diferentes modelos de ciclo de vida, tales como:

- **Modelo en Cascada:** Fases secuenciales, cada una debe completarse antes de comenzar la siguiente.

- **Modelo Iterativo e Incremental:** Desarrollo en ciclos repetidos, incrementando funcionalidad en cada iteración.
- **Modelo Agile:** Enfoque flexible y adaptable, con desarrollo iterativo y entregas frecuentes.
- **Modelo en Espiral:** Combina elementos de los modelos en cascada e iterativo, con un enfoque en la evaluación de riesgos.

Cada una de estas etapas es crucial para garantizar que el software desarrollado sea de alta calidad, satisfaga las necesidades de los usuarios y pueda mantenerse y evolucionar con el tiempo.

2.3.1 Análisis

El análisis en la ingeniería de software es una etapa crucial del ciclo de vida del desarrollo de software que se centra en entender y documentar las necesidades y requisitos del usuario y otras partes interesadas. Esta fase es fundamental porque proporciona la base para el diseño, desarrollo y pruebas del software.

Objetivos del Análisis de Requisitos

1. **Entender las Necesidades del Cliente:**
 - Identificar y comprender lo que los usuarios y stakeholders necesitan y esperan del sistema.
 - Definir claramente los problemas que el software debe resolver.
2. **Establecer Requisitos Funcionales y No Funcionales:**
 - **Requisitos Funcionales:** Especifican lo que el sistema debe hacer, como las funciones y características específicas.
 - **Requisitos No Funcionales:** Incluyen atributos de calidad como rendimiento, usabilidad, seguridad, y escalabilidad.
3. **Crear una Base para el Diseño:**
 - Proporcionar una descripción detallada que guíe el diseño y desarrollo del software.

Actividades Clave en la Fase de Análisis

1. **Recopilación de Requisitos:**
 - **Entrevistas:** Reunirse con usuarios, clientes y otras partes interesadas para recopilar información.
 - **Cuestionarios y Encuestas:** Distribuir preguntas estructuradas para obtener datos específicos.
 - **Observación:** Analizar el entorno de trabajo y las prácticas actuales de los usuarios.
 - **Revisión de Documentos:** Examinar documentación existente como manuales, informes y políticas.
2. **Análisis de Requisitos:**

- **Modelado de Requisitos:** Usar técnicas como diagramas de casos de uso, diagramas de flujo de datos (DFD), y diagramas de actividad para representar visualmente los requisitos.
 - **Priorización de Requisitos:** Determinar cuáles requisitos son esenciales y cuáles son opcionales o pueden ser diferidos.
 - **Verificación y Validación de Requisitos:** Asegurar que los requisitos recopilados son completos, consistentes y precisos.
3. **Documentación de Requisitos:**
- **Especificación de Requisitos del Software (SRS):** Crear un documento detallado que describa todos los requisitos funcionales y no funcionales del sistema.
 - **Prototipos:** Desarrollar prototipos o modelos iniciales del sistema para ayudar a visualizar y refinar los requisitos.
 - **Historias de Usuario:** En metodologías ágiles, usar historias de usuario para describir funcionalidades desde la perspectiva del usuario final.
4. **Revisión y Aprobación de Requisitos:**
- **Revisiones y Reuniones de Validación:** Realizar revisiones con stakeholders para asegurarse de que los requisitos capturados son correctos y completos.
 - **Aprobación Formal:** Obtener la aprobación formal de los requisitos antes de proceder al diseño.

Herramientas y Técnicas Comunes en el Análisis de Requisitos

- **Diagramas UML (Unified Modeling Language):** Diagramas de casos de uso, diagramas de clase, diagramas de secuencia, etc.
- **Diagramas de Flujo de Datos (DFD):** Para representar el flujo de información dentro del sistema.
- **Historias de Usuario y Backlogs:** Usados en metodologías ágiles.
- **Matrices de Trazabilidad de Requisitos:** Para rastrear la cobertura de requisitos a través del ciclo de vida del desarrollo.

Importancia del Análisis de Requisitos

- **Reducción de Costos y Retrasos:** Detectar y corregir errores en las primeras etapas del desarrollo es mucho más económico que hacerlo en etapas posteriores.
- **Mejora de la Calidad del Software:** Un análisis exhaustivo ayuda a garantizar que el software final cumpla con las expectativas y necesidades de los usuarios.
- **Comunicación Efectiva:** Proporciona una base clara y común de entendimiento entre los desarrolladores, usuarios y otros stakeholders.

En resumen, la fase de análisis en la ingeniería de software es esencial para garantizar que el software desarrollado sea adecuado, eficiente y eficaz, satisfaciendo las necesidades y expectativas de los usuarios y stakeholders.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A2.3.1.1

2.3.2 Diseño

La fase de diseño en la ingeniería de software es crucial para convertir los requisitos documentados en la fase de análisis en una arquitectura y un plan detallado que guiará el desarrollo del software. Esta etapa se centra en cómo se construirá el sistema, considerando tanto la estructura general como los detalles específicos de los componentes.

Objetivos del Diseño de Software

1. **Definir la Arquitectura del Sistema:**
 - Crear una estructura organizativa que defina cómo interactuarán los componentes del sistema.
2. **Describir los Componentes del Sistema:**
 - Identificar y definir los módulos o componentes del software y sus responsabilidades.
3. **Especificar las Interfaces:**
 - Describir cómo los componentes se comunicarán entre sí y con el entorno externo.
4. **Asegurar la Calidad:**
 - Implementar principios de diseño que mejoren la mantenibilidad, escalabilidad, rendimiento y seguridad del sistema.

Niveles de Diseño

Diseño de Arquitectura

- **Definición:** Planificación de la estructura general del sistema.
- **Actividades:**
 - **Selección de Arquitectura:** Elegir un estilo arquitectónico adecuado (por ejemplo, cliente-servidor, microservicios, MVC).
 - **Diagramas de Arquitectura:** Crear diagramas que representen la estructura general del sistema y la interacción entre sus componentes principales.
 - **Distribución de Responsabilidades:** Asignar responsabilidades específicas a los diferentes componentes del sistema.
 - **Definición de Subsistemas y Módulos:** Dividir el sistema en subsistemas y módulos manejables.

Diseño de Componentes

- **Definición:** Planificación detallada de los módulos y componentes individuales.
- **Actividades:**
 - **Diagramas de Clase:** Usar diagramas UML para definir las clases y sus relaciones.

- **Diagramas de Secuencia:** Modelar las interacciones entre objetos en el tiempo.
- **Diagramas de Componentes:** Mostrar la organización y las dependencias entre los componentes.
- **Especificación de Interfaces:** Definir claramente las interfaces entre los componentes y cómo se comunicarán.

Diseño de Datos

- **Definición:** Estructuración de los datos que el sistema utilizará y gestionará.
- **Actividades:**
 - **Modelo de Datos:** Crear un modelo de datos lógico y físico.
 - **Diagramas ER (Entidad-Relación):** Representar las entidades y sus relaciones.
 - **Definición de Esquemas de Bases de Datos:** Especificar las tablas, campos, claves y relaciones.

Diseño de Interfaces de Usuario

- **Definición:** Planificación de la interacción del usuario con el sistema.
- **Actividades:**
 - **Prototipos de UI:** Crear prototipos y maquetas de la interfaz de usuario.
 - **Diagramas de Navegación:** Representar el flujo de navegación entre diferentes pantallas o vistas.
 - **Guías de Estilo:** Definir estándares de diseño y usabilidad.

Principios de Diseño de Software

1. **Modularidad:** Dividir el sistema en módulos o componentes independientes y manejables.
2. **Abstracción:** Simplificar la complejidad mediante la ocultación de detalles internos y la exposición de interfaces claras.
3. **Encapsulamiento:** Asegurar que los datos y métodos de un módulo no sean accesibles desde otros módulos excepto a través de interfaces públicas.
4. **Acoplamiento y Cohesión:** Minimizar el acoplamiento (dependencia) entre módulos y maximizar la cohesión (unidad funcional) dentro de cada módulo.
5. **Reusabilidad:** Diseñar componentes que puedan ser reutilizados en diferentes partes del sistema o en otros proyectos.
6. **Mantenibilidad:** Facilitar la modificación y mejora del software en el futuro.

Documentación del Diseño

- **Documentos de Diseño de Software (SDD):** Incluir descripciones detalladas de la arquitectura, componentes, interfaces y modelos de datos.

- **Especificaciones Técnicas:** Documentar los detalles técnicos y las decisiones de diseño.
- **Prototipos y Mockups:** Crear representaciones visuales del sistema para clarificar el diseño.

Herramientas de Diseño

- **Herramientas de Modelado UML:** Como Enterprise Architect, Visual Paradigm, y Lucidchart.
- **Herramientas de Prototipado:** Como Figma, Sketch, y Adobe XD.
- **Herramientas de Gestión de Bases de Datos:** Como MySQL Workbench y Microsoft SQL Server Management Studio.

Importancia del Diseño de Software

- **Prevención de Errores Costosos:** Un buen diseño ayuda a identificar y resolver problemas antes de que se conviertan en errores costosos durante la implementación.
- **Facilitación del Desarrollo:** Proporciona una guía clara y detallada para los desarrolladores.
- **Mejora de la Calidad:** Asegura que el software sea escalable, mantenible y eficiente.

En resumen, la fase de diseño es esencial para establecer una base sólida para el desarrollo del software, asegurando que todos los aspectos críticos del sistema se consideren y planifiquen adecuadamente.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A2.3.2.1 y A2.3.2.2

2.3.3 Desarrollo

La fase de desarrollo en la ingeniería de software, también conocida como implementación o codificación, es donde se traduce el diseño detallado del sistema en código fuente ejecutable. Esta fase es crucial porque convierte los planes y diagramas en un producto funcional.

Objetivos del Desarrollo

1. **Traducción del Diseño a Código:**
 - Implementar el diseño del software utilizando lenguajes de programación adecuados.
2. **Producción de Código de Alta Calidad:**
 - Asegurar que el código sea eficiente, mantenible y conforme a los estándares de calidad.
3. **Integración de Componentes:**

- Integrar los diferentes módulos y componentes desarrollados para formar un sistema cohesivo.

Actividades Clave en la Fase de Desarrollo

1. Configuración del Entorno de Desarrollo:

- **Entornos de Desarrollo Integrado (IDE):** Configuración de IDEs como Visual Studio, Eclipse, IntelliJ IDEA.
- **Control de Versiones:** Uso de sistemas de control de versiones como Git para gestionar el código.
- **Automatización y Scripts:** Configuración de herramientas de construcción y scripts de automatización (por ejemplo, Maven, Gradle).

2. Programación:

- **Codificación de Componentes:** Implementar los módulos y componentes según el diseño detallado.
- **Pruebas Unitarias:** Escribir y ejecutar pruebas unitarias para verificar la funcionalidad de los componentes individuales.
- **Revisión de Código:** Realizar revisiones de código para asegurar la calidad y consistencia del código.
- **Documentación del Código:** Escribir comentarios y documentación para facilitar la comprensión y mantenimiento del código.

3. Integración:

- **Integración Continua (CI):** Utilizar herramientas de integración continua como Jenkins, Travis CI para automatizar la integración y pruebas del código.
- **Pruebas de Integración:** Verificar que los componentes funcionen correctamente cuando se integran.
- **Gestión de Dependencias:** Asegurar que todas las dependencias del proyecto estén correctamente gestionadas y documentadas.

4. Pruebas de Software:

- **Pruebas Funcionales:** Asegurarse de que el software cumple con los requisitos especificados.
- **Pruebas de Rendimiento:** Evaluar cómo se comporta el software bajo diversas condiciones de carga.
- **Pruebas de Seguridad:** Identificar y corregir posibles vulnerabilidades de seguridad.

Buenas Prácticas en la Fase de Desarrollo

1. Adopción de Estándares de Codificación:

- Utilizar convenciones y estándares de codificación para asegurar la coherencia y legibilidad del código.

2. Uso de Principios SOLID:

- Aplicar los principios de diseño orientado a objetos SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface

Segregation, Dependency Inversion) para mejorar la mantenibilidad y escalabilidad del código.

3. Refactorización:

- Mejorar continuamente el código mediante la refactorización para eliminar redundancias y optimizar el diseño sin cambiar la funcionalidad externa.

4. Automatización de Pruebas:

- Automatizar tanto las pruebas unitarias como las de integración para asegurar una cobertura de pruebas consistente y eficiente.

5. Gestión de Dependencias:

- Usar herramientas como Maven, Gradle, npm para gestionar las dependencias del proyecto de manera eficiente.

Herramientas Comunes en el Desarrollo de Software

- **IDEs y Editores de Código:** Visual Studio, IntelliJ IDEA, Eclipse, Visual Studio Code, Atom.
- **Sistemas de Control de Versiones:** Git, Subversion.
- **Herramientas de Integración Continua (CI):** Jenkins, Travis CI, CircleCI.
- **Herramientas de Pruebas:** JUnit, NUnit, Selenium, TestNG.
- **Herramientas de Gestión de Proyectos:** JIRA, Trello, Asana.

Importancia del Desarrollo

- **Concretización del Diseño:** La fase de desarrollo transforma el diseño abstracto en un producto tangible y funcional.
- **Calidad del Producto Final:** La calidad del código escrito durante esta fase impacta directamente en la calidad, rendimiento y mantenibilidad del producto final.
- **Eficiencia y Productividad:** Un buen proceso de desarrollo con herramientas y prácticas adecuadas puede mejorar significativamente la eficiencia y productividad del equipo de desarrollo.

En resumen, la fase de desarrollo es fundamental en el ciclo de vida del software, ya que es donde se construye el producto real a partir del diseño planificado. Es crucial seguir buenas prácticas y utilizar herramientas adecuadas para asegurar que el software desarrollado sea de alta calidad, eficiente y fácil de mantener.

2.3.4 Pruebas

La fase de pruebas en la ingeniería de software, también conocida como testing, es fundamental para garantizar que el software desarrollado funcione correctamente y cumpla con los requisitos especificados. Esta fase implica verificar y validar el software para detectar y corregir errores antes de su despliegue.

Objetivos de las Pruebas de Software

1. **Verificar el Funcionamiento Correcto:**
 - Asegurar que el software cumple con los requisitos funcionales y no funcionales.
2. **Detectar y Corregir Errores:**
 - Identificar defectos y fallos en el software para corregirlos antes de su implementación.
3. **Garantizar la Calidad:**
 - Asegurar que el software sea fiable, seguro, eficiente y fácil de usar.

Tipos de Pruebas de Software

1. **Pruebas Unitarias:**
 - **Definición:** Verificación de componentes individuales o unidades del código.
 - **Objetivo:** Asegurar que cada unidad funciona correctamente de manera aislada.
 - **Herramientas:** JUnit, NUnit, TestNG.
2. **Pruebas de Integración:**
 - **Definición:** Verificación de la interacción entre componentes integrados.
 - **Objetivo:** Asegurar que los componentes interactúan correctamente y funcionan juntos.
 - **Herramientas:** Selenium, Postman.
3. **Pruebas de Sistema:**
 - **Definición:** Verificación del sistema completo para asegurar que cumple con los requisitos especificados.
 - **Objetivo:** Evaluar el comportamiento del sistema en su totalidad.
 - **Herramientas:** TestComplete, QTP/UFT.
4. **Pruebas de Aceptación:**
 - **Definición:** Verificación del software por parte de los usuarios finales o stakeholders.
 - **Objetivo:** Asegurar que el software cumple con las expectativas y requisitos del usuario.
 - **Métodos:** Pruebas alfa y beta.
5. **Pruebas de Regresión:**
 - **Definición:** Verificación del software para asegurar que las modificaciones no introducen nuevos errores.
 - **Objetivo:** Asegurar que el software sigue funcionando correctamente después de cambios.
 - **Herramientas:** Selenium, JUnit, TestNG.
6. **Pruebas de Rendimiento:**
 - **Definición:** Evaluación del comportamiento del software bajo condiciones de carga y estrés.
 - **Objetivo:** Asegurar que el software funcione de manera eficiente bajo diferentes condiciones de carga.
 - **Herramientas:** JMeter, LoadRunner.
7. **Pruebas de Seguridad:**

- **Definición:** Evaluación de la seguridad del software para identificar vulnerabilidades.
 - **Objetivo:** Asegurar que el software está protegido contra amenazas y ataques.
 - **Herramientas:** OWASP ZAP, Burp Suite.
8. **Pruebas de Usabilidad:**
- **Definición:** Evaluación de la interfaz y experiencia de usuario.
 - **Objetivo:** Asegurar que el software es fácil de usar y entender.
 - **Métodos:** Pruebas de usabilidad, pruebas heurísticas.

Proceso de Pruebas de Software

1. **Planificación de Pruebas:**
 - Definir el alcance, objetivos, enfoque y recursos para las pruebas.
 - Crear un plan de pruebas detallado que incluya casos de prueba, cronograma y responsabilidades.
2. **Diseño de Casos de Prueba:**
 - Identificar y documentar los casos de prueba basados en los requisitos del software.
 - Especificar los datos de prueba, condiciones iniciales y resultados esperados.
3. **Configuración del Entorno de Pruebas:**
 - Preparar el entorno necesario para realizar las pruebas, incluyendo hardware, software, y herramientas.
4. **Ejecución de Pruebas:**
 - Ejecutar los casos de prueba y registrar los resultados.
 - Identificar y reportar defectos y errores encontrados durante las pruebas.
5. **Evaluación de Resultados:**
 - Analizar los resultados de las pruebas para determinar si el software cumple con los requisitos y criterios de aceptación.
 - Priorizar y corregir los defectos encontrados.
6. **Cierre de Pruebas:**
 - Documentar y comunicar los resultados de las pruebas.
 - Realizar una evaluación post-mortem para identificar lecciones aprendidas y mejorar el proceso de pruebas.

Herramientas de Pruebas

- **Automatización de Pruebas:** Selenium, QTP/UFT, TestComplete.
- **Gestión de Pruebas:** JIRA, TestRail, Quality Center.
- **Pruebas de Rendimiento:** JMeter, LoadRunner.
- **Pruebas de Seguridad:** OWASP ZAP, Burp Suite.
- **Pruebas Unitarias:** JUnit, NUnit, TestNG.

Importancia de las Pruebas de Software

- **Mejora de la Calidad:** Asegura que el software cumple con los estándares de calidad y satisface las necesidades del usuario.
- **Reducción de Riesgos:** Identifica y corrige defectos antes de que el software sea desplegado, reduciendo el riesgo de fallos en producción.
- **Aumento de la Confianza:** Proporciona confianza en que el software funcionará como se espera bajo diversas condiciones.
- **Cumplimiento de Requisitos:** Verifica que el software cumple con todos los requisitos funcionales y no funcionales especificados.

En resumen, la fase de pruebas es esencial para asegurar que el software desarrollado sea fiable, seguro, eficiente y cumpla con las expectativas y necesidades de los usuarios. Un proceso de pruebas bien planificado y ejecutado puede prevenir problemas costosos y asegurar la calidad del producto final.

2.3.5 Implementación

La fase de implementación en la ingeniería de software, también conocida como despliegue, es el proceso de instalar y configurar el software desarrollado en el entorno de producción. Esta fase asegura que el software esté listo y disponible para ser utilizado por los usuarios finales. La implementación es una etapa crítica ya que cualquier error en esta fase puede afectar la funcionalidad y la aceptación del software.

Objetivos de la Implementación

1. **Instalar el Software en el Entorno de Producción:**
 - Asegurar que el software esté correctamente instalado y funcionando en el entorno de producción.
2. **Configuración del Software:**
 - Ajustar las configuraciones necesarias para que el software funcione correctamente según las necesidades del cliente.
3. **Migración de Datos:**
 - Transferir los datos existentes al nuevo sistema de manera segura y precisa.
4. **Verificación Post-Despliegue:**
 - Realizar pruebas para asegurarse de que la implementación fue exitosa y que el software funciona como se espera.
5. **Capacitación y Documentación:**
 - Proveer formación y documentación necesaria para que los usuarios finales puedan utilizar el software eficientemente.

Actividades Clave en la Fase de Implementación

1. **Preparación del Entorno de Producción:**
 - **Configuración de Servidores:** Asegurar que los servidores (hardware y software) estén preparados y configurados adecuadamente.
 - **Configuración de Redes:** Asegurar que la red y los servicios asociados estén configurados y sean accesibles.
2. **Instalación del Software:**
 - **Instalación de Componentes:** Instalar todos los componentes del software en los servidores de producción.
 - **Configuración de Parámetros:** Ajustar las configuraciones necesarias (por ejemplo, archivos de configuración, bases de datos).
3. **Migración de Datos:**
 - **Transferencia de Datos:** Migrar los datos existentes desde el sistema anterior al nuevo sistema.
 - **Validación de Datos:** Verificar que los datos migrados sean correctos y completos.
4. **Pruebas de Implementación:**
 - **Pruebas de Aceptación por el Usuario (UAT):** Permitir que los usuarios finales prueben el software en el entorno de producción para asegurar que cumple con sus necesidades.
 - **Pruebas de Sistema:** Ejecutar pruebas completas para verificar que el sistema en producción funcione correctamente.
5. **Documentación:**
 - **Guías de Usuario:** Proveer documentación que explique cómo usar el software.
 - **Documentación Técnica:** Proveer documentación técnica para los administradores del sistema.
6. **Capacitación:**
 - **Formación de Usuarios:** Capacitar a los usuarios finales sobre cómo usar el software.
 - **Formación del Equipo de Soporte:** Capacitar al equipo de soporte sobre cómo manejar y resolver posibles problemas.
7. **Monitoreo Post-Implementación:**
 - **Monitoreo del Sistema:** Supervisar el sistema para asegurar que funcione correctamente y detectar cualquier problema.
 - **Soporte Post-Implementación:** Proveer soporte para resolver problemas que surjan después de la implementación.

Modelos de Implementación

1. **Implementación Directa (Big Bang):**
 - **Definición:** El sistema nuevo reemplaza al viejo en un solo evento.
 - **Ventajas:** Rápido y directo.
 - **Desventajas:** Alto riesgo si hay fallos.
2. **Implementación Paralela:**

- **Definición:** Ambos sistemas (viejo y nuevo) funcionan en paralelo durante un período.
- **Ventajas:** Menor riesgo al comparar resultados.
- **Desventajas:** Costoso y consume más recursos.
- 3. **Implementación en Fases:**
 - **Definición:** El sistema se implementa en fases o módulos.
 - **Ventajas:** Menor riesgo y más control.
 - **Desventajas:** Puede ser más lento y complejo.
- 4. **Implementación Piloto:**
 - **Definición:** El sistema nuevo se implementa en un segmento limitado de la organización.
 - **Ventajas:** Permite detectar y corregir problemas antes de una implementación completa.
 - **Desventajas:** Puede no representar todos los posibles problemas a escala completa.

Herramientas Comunes en la Implementación

- **Sistemas de Gestión de Configuración:** Ansible, Puppet, Chef.
- **Herramientas de Despliegue Continuo:** Jenkins, GitLab CI/CD, Travis CI.
- **Sistemas de Monitoreo:** Nagios, Zabbix, Prometheus.
- **Herramientas de Migración de Datos:** Talend, Apache NiFi, Microsoft Data Migration Assistant.

Importancia de la Implementación

- **Disponibilidad del Sistema:** Asegura que el software esté disponible y funcione correctamente en el entorno de producción.
- **Satisfacción del Cliente:** Garantiza que el software cumpla con las expectativas y necesidades de los usuarios finales.
- **Reducción de Riesgos:** Minimiza los riesgos asociados con la puesta en producción de nuevos sistemas.
- **Transición Suave:** Facilita una transición suave desde el sistema antiguo al nuevo, minimizando interrupciones en las operaciones.

En resumen, la fase de implementación es fundamental para llevar el software desde el entorno de desarrollo hasta el entorno de producción, asegurando que esté listo y disponible para los usuarios finales de manera eficiente y efectiva.

Mantenimiento

El mantenimiento de sistemas es el proceso continuo de actualización y mejora del software después de su implementación. Incluye varias actividades:

1. **Corrección de Errores:** Solución de fallos y errores que aparecen después del despliegue.

2. **Mejoras y Actualizaciones:** Incorporación de nuevas funcionalidades y mejora de las existentes.
3. **Optimización:** Mejoras en el rendimiento y eficiencia del sistema.
4. **Adaptación:** Modificaciones para adaptarse a cambios en el entorno o en los requisitos del usuario.
5. **Prevención:** Actividades para evitar problemas futuros y mejorar la robustez del sistema.

Estrategias de Mantenimiento

Existen varias estrategias para el mantenimiento efectivo de sistemas:

- **Mantenimiento Correctivo:** Corrección de errores y fallos.
- **Mantenimiento Adaptativo:** Adaptación del sistema a nuevos entornos o tecnologías.
- **Mantenimiento Perfectivo:** Mejoras en el rendimiento y la funcionalidad del sistema.
- **Mantenimiento Preventivo:** Prevención de problemas futuros mediante la mejora de la estructura y el código.

Herramientas y Tecnologías

- **Sistemas de Gestión de Configuración:** Git, SVN
- **Herramientas de Automatización de Pruebas:** Selenium, JUnit
- **Monitoreo y Logging:** Nagios, ELK Stack
- **Plataformas CI/CD:** Jenkins, GitLab CI

El éxito en el desarrollo y mantenimiento de sistemas requiere una planificación cuidadosa, la adopción de buenas prácticas y el uso de herramientas adecuadas para garantizar que el software sea confiable, eficiente y fácil de mantener.

2.3.6 Mejora continua

La mejora continua en el desarrollo de sistemas se centra en la optimización constante de procesos, herramientas y prácticas para aumentar la calidad del software, la eficiencia del equipo y la satisfacción del cliente. Aquí tienes una guía sobre cómo aplicar la mejora continua en el contexto del desarrollo de sistemas:

Principios de Mejora Continua en el Desarrollo de Sistemas

1. **Retroalimentación Rápida:** Obtener y actuar sobre la retroalimentación de los usuarios y de las pruebas lo más rápido posible.
2. **Iteración y Adaptación:** Trabajar en ciclos cortos de desarrollo, como sprints, para iterar y mejorar continuamente.
3. **Calidad Incorporada:** Asegurar que la calidad esté integrada en cada fase del desarrollo, desde el diseño hasta la implementación y las pruebas.

4. **Automatización:** Automatizar las pruebas, la integración y el despliegue para reducir errores humanos y acelerar el proceso.
5. **Colaboración y Comunicación:** Fomentar una comunicación abierta y efectiva entre todos los miembros del equipo.

Métodos y Herramientas para la Mejora Continua en el Desarrollo de Sistemas

1. **Ciclo PDCA (Plan-Do-Check-Act)**
 - **Plan:** Identificar áreas de mejora en el proceso de desarrollo o en el software.
 - **Do:** Implementar cambios en un entorno controlado.
 - **Check:** Evaluar los resultados de los cambios.
 - **Act:** Adoptar los cambios exitosos y ajustar los que no lo son.
2. **Scrum y Agile**
 - **Scrum:** Utilizar sprints y reuniones diarias para evaluar el progreso y ajustar las metas.
 - **Agile:** Adoptar principios ágiles como la entrega continua de valor y la respuesta rápida al cambio.
3. **DevOps**
 - Integrar las prácticas de desarrollo y operaciones para mejorar la colaboración y la eficiencia.
 - Utilizar pipelines de CI/CD (Integración Continua/Despliegue Continuo) para automatizar el flujo de trabajo.
4. **Lean Software Development**
 - Aplicar principios Lean como la eliminación del desperdicio, la entrega rápida y la mejora continua.
5. **Kaizen**
 - Implementar pequeños cambios incrementales regularmente para mejorar procesos y prácticas.
6. **Six Sigma**
 - Utilizar métodos y herramientas estadísticas para identificar y eliminar defectos en el proceso de desarrollo.

Herramientas de Soporte

1. **Herramientas de Gestión de Proyectos:** Jira, Trello, Asana
2. **Sistemas de Control de Versiones:** Git, SVN
3. **Plataformas CI/CD:** Jenkins, GitLab CI, CircleCI
4. **Herramientas de Pruebas Automatizadas:** Selenium, JUnit, TestNG
5. **Monitoreo y Logging:** ELK Stack (Elasticsearch, Logstash, Kibana), Splunk, Grafana
6. **Herramientas de Code Review:** GitHub, GitLab, Bitbucket

Beneficios de la Mejora Continua en el Desarrollo de Sistemas

- **Calidad Mejorada:** Menos defectos y mayor confiabilidad del software.
- **Mayor Eficiencia:** Procesos de desarrollo más rápidos y menos retrabajo.

- **Mayor Satisfacción del Cliente:** Entregas más frecuentes y alineadas con las necesidades del cliente.
- **Mayor Moral del Equipo:** Un entorno de trabajo más colaborativo y gratificante.
- **Mejor Adaptabilidad:** Capacidad de responder rápidamente a cambios en los requisitos y el entorno tecnológico.

Implementación de la Mejora Continua en el Desarrollo de Sistemas

1. **Evaluación Inicial:** Analizar el estado actual del proceso de desarrollo y identificar áreas de mejora.
2. **Formación y Capacitación:** Capacitar al equipo en principios y herramientas de mejora continua.
3. **Pequeños Cambios Incrementales:** Implementar mejoras en pequeños pasos para minimizar el riesgo.
4. **Medición y Seguimiento:** Utilizar métricas y KPIs para evaluar el impacto de las mejoras.
5. **Retroalimentación y Ajuste:** Recoger retroalimentación del equipo y de los usuarios, y ajustar las prácticas en consecuencia.

La mejora continua es esencial para mantener la competitividad y la calidad en el desarrollo de sistemas, asegurando que los productos se mantengan relevantes y de alta calidad en un entorno de cambios rápidos.

Referencias

1. **Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach.** Este libro es uno de los textos más ampliamente utilizados en el campo y cubre una amplia gama de temas relacionados con la ingeniería de software, desde la gestión de proyectos hasta las técnicas de diseño y pruebas.
2. **Sommerville, I. (2015). Software Engineering.** Un texto fundamental que proporciona una visión integral de los principios y prácticas de la ingeniería de software, incluyendo metodologías ágiles y tradicionales.
3. **Van Vliet, H. (2008). Software Engineering: Principles and Practice.** Este libro proporciona una combinación equilibrada de teoría y práctica, con énfasis en la aplicación de principios de ingeniería a problemas del mundo real.
4. **Fowler, M. (2018). Refactoring: Improving the Design of Existing Code.** Un recurso clave para entender cómo mejorar el diseño del software existente a través de técnicas de refactorización.
5. **Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering.** Un clásico que ofrece una visión profunda sobre los desafíos de la gestión de proyectos de software y las lecciones aprendidas a lo largo de los años.

UNIDAD III MODELOS PRESCRIPTIVOS DEL DESARROLLO DE SISTEMAS DE INFORMACIÓN

Los modelos prescriptivos en el desarrollo de sistemas de información son enfoques estructurados y secuenciales que proporcionan una guía detallada sobre cómo llevar a cabo el proceso de desarrollo de software. Estos modelos ayudan a organizar y gestionar el desarrollo de sistemas de manera eficiente y predecible. Aquí se describen algunos de los modelos prescriptivos más conocidos:

3.1. Modelo en Cascada.

El modelo en cascada, también conocido como "Waterfall", es uno de los primeros enfoques utilizados en el desarrollo de software. Se caracteriza por ser un proceso lineal y secuencial donde cada fase del desarrollo debe completarse antes de que la siguiente comience. Aquí se detallan las fases del modelo en cascada, sus ventajas, desventajas y su aplicabilidad:

Fases del Modelo en Cascada

1. Requisitos (Requirement Analysis)

- **Descripción:** En esta fase, se recopilan y documentan todos los requisitos del sistema. Los analistas de sistemas interactúan con los clientes y usuarios finales para comprender sus necesidades y expectativas.
- **Entregables:** Documento de especificación de requisitos del sistema (SRS).

2. Diseño del Sistema (System Design)

- **Descripción:** Basado en los requisitos recopilados, se diseñan la arquitectura del sistema y los componentes del software. Se crean diagramas de flujo, esquemas de base de datos y otros documentos de diseño.
- **Entregables:** Documento de diseño del sistema (SDD).

3. Implementación (Implementation)

- **Descripción:** Los desarrolladores comienzan a codificar el sistema de acuerdo con los documentos de diseño. Esta fase implica la conversión del diseño en código ejecutable.
- **Entregables:** Código fuente del sistema.

4. Pruebas (Testing)

- **Descripción:** El sistema desarrollado se prueba exhaustivamente para identificar y corregir errores y defectos. Se llevan a cabo pruebas unitarias, de integración, de sistema y de aceptación.
- **Entregables:** Informe de pruebas y software validado.

5. Despliegue (Deployment)

- **Descripción:** El software probado se despliega en el entorno de producción donde los usuarios finales pueden comenzar a utilizarlo.
- **Entregables:** Software operativo y documentación del usuario.

6. Mantenimiento (Maintenance)

- **Descripción:** Después del despliegue, el software entra en la fase de mantenimiento. Esta fase incluye la corrección de errores que no se detectaron durante las pruebas, la mejora del software según las necesidades cambiantes del usuario y la adaptación a nuevos entornos.
- **Entregables:** Actualizaciones del software y documentación actualizada.

Ventajas del Modelo en Cascada

1. **Claridad y Estructura:** Cada fase tiene objetivos y entregables claros, lo que facilita la planificación y la gestión del proyecto.
2. **Facilidad de Gestión:** La naturaleza secuencial del modelo permite un seguimiento y control del progreso relativamente sencillo.
3. **Documentación Completa:** La extensa documentación en cada fase proporciona una base sólida para el desarrollo y el mantenimiento del software.
4. **Fases Definidas:** La conclusión de una fase antes de comenzar la siguiente puede ayudar a minimizar los riesgos.

Desventajas del Modelo en Cascada

1. **Rigidez:** Una vez que una fase se ha completado, es difícil realizar cambios sin afectar las fases subsiguientes.
2. **Requisitos Inflexibles:** Asume que todos los requisitos pueden definirse al principio del proyecto, lo cual puede no ser realista.
3. **Retrasos en la Retroalimentación:** Los usuarios finales no ven el producto hasta las etapas finales, lo que puede resultar en sorpresas y falta de alineación con sus expectativas.
4. **Adaptabilidad Limitada:** No es adecuado para proyectos donde los requisitos pueden cambiar o evolucionar significativamente durante el desarrollo.

Aplicabilidad del Modelo en Cascada

El modelo en cascada es más adecuado para proyectos donde:

- Los requisitos están bien definidos y son estables.
- Los cambios durante el desarrollo son mínimos o inexistentes.
- Los proyectos son de corta duración y no complejos.
- La calidad del producto y la precisión de la documentación son críticas.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A3.1.1, A3.1.2 y A3.1.3

3.2. Modelos Evolutivos.

Los modelos evolutivos en el desarrollo de sistemas de información se centran en la entrega incremental y iterativa del software. En lugar de desarrollar todo el sistema en una sola fase, los modelos evolutivos permiten construir el sistema en partes, refinando y mejorando el software con cada iteración. Esto facilita la incorporación de cambios y la mejora continua basada en la retroalimentación del usuario. Aquí se presentan algunos de los modelos evolutivos más conocidos:

1. Modelo Incremental

El modelo incremental combina los beneficios del modelo en cascada con un enfoque iterativo, permitiendo la entrega gradual de partes del sistema.

Fases del Modelo Incremental:

1. **Análisis de Requisitos Iniciales:** Identificación de los requisitos básicos.
2. **Planificación de Incrementos:** Dividir el proyecto en incrementos manejables.
3. **Diseño y Desarrollo de Incrementos:** Diseño, implementación y prueba de cada incremento.
4. **Integración y Despliegue de Incrementos:** Integración y despliegue de cada incremento en el entorno de producción.
5. **Retroalimentación y Refinamiento:** Recoger retroalimentación y realizar mejoras en los incrementos subsiguientes.

Ventajas:

- Entregas parciales del producto permiten una retroalimentación temprana.
- Reducción de riesgos al abordar el proyecto en partes más pequeñas.
- Mejor adaptación a los cambios en los requisitos.

Desventajas:

- Necesidad de una buena planificación y gestión de cada incremento.
- Complejidad en la integración de incrementos.

2. Modelo de Prototipado

El modelo de prototipado se centra en la creación rápida de un prototipo funcional para comprender mejor los requisitos del usuario.

Fases del Modelo de Prototipado:

1. **Recolección de Requisitos:** Recopilación de los requisitos iniciales.
2. **Desarrollo del Prototipo:** Creación rápida de un prototipo del sistema.
3. **Evaluación del Prototipo:** Evaluación y retroalimentación del usuario.
4. **Refinamiento del Prototipo:** Ajuste del prototipo basado en la retroalimentación.
5. **Desarrollo Completo:** Desarrollo del sistema completo basado en el prototipo refinado.

Ventajas:

- Mejora la comprensión de los requisitos.
- Facilita la retroalimentación temprana y continua del usuario.
- Reduce el riesgo de malentendidos en los requisitos.

Desventajas:

- El prototipo puede ser considerado como el sistema final, llevando a expectativas no realistas.
- Puede resultar en una mala arquitectura si no se maneja adecuadamente.

3. Modelo Espiral

El modelo espiral combina elementos de diseño y prototipado en un enfoque iterativo, con un fuerte énfasis en la evaluación de riesgos.

Fases del Modelo Espiral:

1. **Planificación:** Identificación de objetivos, alternativas y restricciones.
2. **Análisis de Riesgos:** Evaluación y mitigación de riesgos.
3. **Desarrollo y Prueba:** Desarrollo y prueba del sistema.
4. **Evaluación del Cliente:** Evaluación del cliente y planificación del siguiente ciclo.

Ventajas:

- Gestión de riesgos efectiva en cada iteración.
- Flexibilidad para adaptarse a cambios en los requisitos.
- Promueve la participación continua del cliente.

Desventajas:

- Complejidad en la gestión del proyecto.
- Puede ser costoso y consumir tiempo, especialmente en la evaluación de riesgos.

4. Modelo Agile

El enfoque Agile se basa en iteraciones cortas y la entrega continua de valor al cliente. Metodologías populares dentro del enfoque Agile incluyen Scrum, Kanban, y Extreme Programming (XP).

Fases del Modelo Agile:

1. **Planificación del Sprint:** Planificación de las tareas para la iteración (sprint).
2. **Desarrollo del Sprint:** Implementación de las funcionalidades planificadas.
3. **Revisión del Sprint:** Revisión y demostración de las funcionalidades desarrolladas.
4. **Retrospectiva del Sprint:** Evaluación del proceso y planificación de mejoras para el próximo sprint.

Ventajas:

- Flexibilidad y adaptabilidad a los cambios.
- Entrega continua de valor y retroalimentación frecuente del cliente.
- Mejora de la colaboración y comunicación dentro del equipo.

Desventajas:

- Requiere una alta disciplina y compromiso del equipo.
- Puede ser difícil de implementar en grandes organizaciones con estructuras rígidas.

Comparación de Modelos Evolutivos

- **Modelo Incremental:** Adecuado para proyectos donde se pueden entregar incrementos funcionales gradualmente.
- **Modelo de Prototipado:** Beneficioso cuando los requisitos no están claros y se necesita retroalimentación continua del usuario.
- **Modelo Espiral:** Recomendado para proyectos grandes y complejos con riesgos significativos.
- **Modelo Agile:** Ideal para entornos dinámicos donde los requisitos cambian frecuentemente y se valora la entrega continua de valor.

Aplicabilidad de los Modelos Evolutivos

Los modelos evolutivos son especialmente adecuados para proyectos donde:

- Los requisitos no están completamente definidos o pueden cambiar.
- Es importante obtener retroalimentación temprana y continua del usuario.

- Se requiere una entrega rápida y frecuente de partes del sistema.
- El proyecto es complejo y presenta riesgos significativos que deben gestionarse de manera iterativa.

3.3. Modelos Especiales.

Los modelos especiales en el desarrollo de sistemas de información son enfoques que se utilizan en circunstancias específicas y con objetivos particulares que no se abordan adecuadamente con los modelos más tradicionales. Estos modelos están diseñados para satisfacer necesidades únicas o para aprovechar metodologías innovadoras. Aquí se presentan algunos de los modelos especiales más reconocidos:

1. Modelo de Desarrollo Basado en Componentes (CBD - Component-Based Development)

El desarrollo basado en componentes se centra en construir sistemas a partir de componentes de software reutilizables. Cada componente es una unidad funcional independiente que puede ser ensamblada para formar un sistema completo.

Fases del Modelo CBD:

1. **Análisis de Requisitos:** Determinación de las funcionalidades necesarias y selección de componentes existentes.
2. **Diseño de Componentes:** Diseño de componentes nuevos y adaptación de componentes existentes.
3. **Desarrollo y Prueba de Componentes:** Codificación y prueba de cada componente de forma independiente.
4. **Integración de Componentes:** Ensamblaje de componentes en el sistema completo.
5. **Pruebas del Sistema:** Pruebas integrales del sistema ensamblado.

Ventajas:

- Reutilización de componentes reduce el tiempo y el costo de desarrollo.
- Facilita la evolución y el mantenimiento del sistema.
- Aumenta la consistencia y calidad del software.

Desventajas:

- Dependencia de la disponibilidad de componentes adecuados.
- Complejidad en la integración de componentes de diferentes fuentes.

2. Modelo de Desarrollo de Software Orientado a Aspectos (AOSD - Aspect-Oriented Software Development)

El desarrollo de software orientado a aspectos aborda la modularización de preocupaciones transversales (como la seguridad, el logging, o la gestión de transacciones) que no se manejan bien con la programación orientada a objetos tradicional.

Fases del Modelo AOSD:

1. **Identificación de Aspectos:** Determinación de las preocupaciones transversales.
2. **Diseño de Aspectos:** Diseño de módulos que encapsulan cada aspecto.
3. **Implementación de Aspectos:** Codificación de los aspectos.
4. **Integración de Aspectos:** Combinación de aspectos con los módulos principales del sistema.
5. **Pruebas del Sistema:** Verificación de la funcionalidad del sistema completo, incluyendo los aspectos.

Ventajas:

- Mejora la modularidad y el mantenimiento del software.
- Permite una mayor reutilización de código.
- Facilita la implementación de funcionalidades transversales.

Desventajas:

- Complejidad en la identificación y diseño de aspectos.
- Dificultades en la integración y pruebas de aspectos.

3. Modelo de Desarrollo Dirigido por Modelos (MDD - Model-Driven Development)

El desarrollo dirigido por modelos se basa en la creación y transformación de modelos que representan diferentes aspectos del sistema. El código fuente se genera a partir de estos modelos.

Fases del Modelo MDD:

1. **Creación de Modelos:** Desarrollo de modelos de alto nivel que representan la estructura y comportamiento del sistema.
2. **Transformación de Modelos:** Conversión de modelos de alto nivel en modelos más detallados o directamente en código.
3. **Generación de Código:** Generación automática de código fuente a partir de modelos detallados.

4. **Pruebas y Refinamiento:** Pruebas del sistema generado y refinamiento de modelos según sea necesario.

Ventajas:

- Mejora la abstracción y comprensión del sistema.
- Reduce errores y inconsistencias mediante la generación automática de código.
- Facilita la comunicación entre diferentes partes interesadas.

Desventajas:

- Requiere herramientas y conocimientos especializados.
- Dependencia en la calidad de las herramientas de generación de código.

4. Modelo Ágil Escalado (Scaled Agile Framework - SAFe)

SAFe es un marco de trabajo que permite aplicar principios ágiles en organizaciones grandes, coordinando múltiples equipos que trabajan en diferentes partes del sistema.

Fases del Modelo SAFe:

1. **Planificación de Programas:** Definición de objetivos y planificación a nivel de programa.
2. **Desarrollo de Iteraciones:** Desarrollo de funcionalidades en sprints cortos, coordinados entre equipos.
3. **Revisión de Iteraciones:** Evaluación de los resultados de cada iteración y ajustes necesarios.
4. **Release Train:** Coordinación de lanzamientos de software que incluyen entregas de múltiples equipos.
5. **Retrospectiva y Mejora Continua:** Evaluación del proceso y aplicación de mejoras continuas.

Ventajas:

- Facilita la aplicación de principios ágiles en organizaciones grandes y complejas.
- Mejora la coordinación y comunicación entre equipos.
- Promueve la entrega continua de valor.

Desventajas:

- Complejidad en la implementación y gestión de múltiples equipos.
- Requiere un cambio cultural significativo en la organización.

5. Modelo de Desarrollo de Software Dirigido por Pruebas (TDD - Test-Driven Development)

TDD es una metodología que enfatiza la creación de pruebas antes de desarrollar el código que debe pasar esas pruebas.

Fases del Modelo TDD:

1. **Escribir una Prueba:** Crear una prueba automatizada que inicialmente fallará.
2. **Desarrollar el Código:** Escribir el mínimo código necesario para pasar la prueba.
3. **Ejecutar la Prueba:** Verificar que la prueba pase con éxito.
4. **Refactorización:** Mejorar el código sin cambiar su comportamiento.
5. **Repetir:** Repetir el ciclo para cada nueva funcionalidad.

Ventajas:

- Mejora la calidad y la cobertura del código mediante pruebas exhaustivas.
- Facilita la detección temprana de errores.
- Promueve un diseño de código más limpio y mantenible.

Desventajas:

- Requiere una disciplina rigurosa y puede aumentar el tiempo de desarrollo inicial.
- Puede ser difícil de aplicar en proyectos con requisitos poco claros.

Aplicabilidad de los Modelos Especiales

Estos modelos especiales son aplicables en diferentes contextos dependiendo de los requisitos del proyecto, las características del equipo y la naturaleza del sistema que se va a desarrollar. Por ejemplo:

- **CBD** es útil cuando se pueden reutilizar componentes de software.
- **AOSD** es ideal para sistemas con preocupaciones transversales complejas.
- **MDD** es adecuado para proyectos donde los modelos pueden proporcionar una mejor abstracción y automatización.
- **SAFe** es beneficioso para organizaciones grandes que necesitan coordinar múltiples equipos ágiles.
- **TDD** es excelente para proyectos donde la calidad del código y la detección temprana de errores son prioritarias.

Estos modelos permiten una mayor flexibilidad y adaptabilidad, facilitando la entrega de software de alta calidad que se ajusta mejor a las necesidades cambiantes del usuario y del entorno.

3.4. El Proceso Unificado de Desarrollo de software.

El Proceso Unificado de Desarrollo de Software (RUP - Rational Unified Process) es un marco de trabajo adaptable y basado en la iteración para la ingeniería de software. Fue desarrollado por Rational Software, ahora parte de IBM, y se basa en mejores prácticas del desarrollo de software orientado a objetos. RUP es altamente estructurado y flexible, permitiendo a los equipos de desarrollo adaptarlo a las necesidades específicas de sus proyectos. A continuación, se detallan los principios, fases, disciplinas, y ventajas del Proceso Unificado.

Principios del Proceso Unificado

1. **Iterativo e Incremental:** El desarrollo se realiza en ciclos iterativos, cada uno de los cuales resulta en un incremento funcional del sistema.
2. **Gestionado por Casos de Uso:** Los casos de uso guían el desarrollo del sistema, asegurando que los requisitos del usuario final se entiendan y se implementen correctamente.
3. **Centrado en la Arquitectura:** La arquitectura del sistema se define temprano y se refina a lo largo del proyecto, proporcionando una base sólida para el desarrollo.
4. **Control de Calidad:** La calidad se integra en el proceso a través de revisiones, pruebas y validaciones continuas.

Fases del Proceso Unificado

El Proceso Unificado se divide en cuatro fases principales, cada una con objetivos específicos:

1. **Inicio (Inception)**
 - **Objetivo:** Definir el alcance del proyecto, identificar los riesgos principales y desarrollar una visión preliminar del sistema.
 - **Actividades Clave:**
 - Definición de los casos de uso y requisitos clave.
 - Identificación de riesgos.
 - Estimación de costos y tiempo.
 - Preparación de un plan inicial del proyecto.
 - **Entregables:**
 - Documento de visión del proyecto.
 - Lista de casos de uso.
 - Plan de proyecto inicial.
2. **Elaboración (Elaboration)**
 - **Objetivo:** Refinar los requisitos, desarrollar una arquitectura sólida y mitigar los riesgos más significativos.
 - **Actividades Clave:**
 - Refinamiento de los casos de uso y los requisitos.
 - Desarrollo y validación de la arquitectura del sistema.
 - Realización de prototipos y pruebas.

- Actualización del plan del proyecto con mayor precisión.
- **Entregables:**
 - Modelo de casos de uso detallado.
 - Modelo de diseño arquitectónico.
 - Plan de proyecto actualizado.
- 3. **Construcción (Construction)**
 - **Objetivo:** Completar el desarrollo del sistema, integrar los componentes y realizar pruebas completas.
 - **Actividades Clave:**
 - Implementación de componentes y funcionalidades.
 - Integración del sistema completo.
 - Pruebas unitarias, de integración y del sistema.
 - Preparación de documentación del usuario y del sistema.
 - **Entregables:**
 - Código fuente del sistema.
 - Resultados de las pruebas.
 - Documentación técnica y del usuario.
- 4. **Transición (Transition)**
 - **Objetivo:** Entregar el sistema a los usuarios finales y asegurar una transición suave al entorno de producción.
 - **Actividades Clave:**
 - Implementación del sistema en el entorno de producción.
 - Entrenamiento de usuarios y personal de soporte.
 - Corrección de problemas identificados post-implementación.
 - Recolección de retroalimentación de los usuarios.
 - **Entregables:**
 - Sistema operativo en producción.
 - Material de entrenamiento y soporte.
 - Informe de evaluación post-implementación.

Disciplinas del Proceso Unificado

Las disciplinas en RUP representan áreas de actividad concurrentes que se llevan a cabo a lo largo del ciclo de vida del proyecto. Estas disciplinas incluyen:

1. **Modelado de Negocios (Business Modeling)**
2. **Requisitos (Requirements)**
3. **Análisis y Diseño (Analysis and Design)**
4. **Implementación (Implementation)**
5. **Pruebas (Testing)**
6. **Despliegue (Deployment)**
7. **Gestión de Configuración y Cambio (Configuration and Change Management)**
8. **Gestión de Proyecto (Project Management)**
9. **Entorno (Environment)**

Ventajas del Proceso Unificado

1. **Flexibilidad y Adaptabilidad:** RUP puede adaptarse a una amplia variedad de proyectos y contextos, permitiendo ajustes en función de las necesidades específicas.
2. **Enfoque en Calidad y Riesgos:** La integración de pruebas continuas y la gestión de riesgos aseguran la calidad y la mitigación de problemas potenciales.
3. **Documentación y Trazabilidad:** La creación de documentos detallados y la trazabilidad desde los requisitos hasta la implementación facilitan el mantenimiento y la evolución del sistema.
4. **Mejores Prácticas:** RUP incorpora mejores prácticas de la industria, promoviendo un enfoque estructurado y profesional en el desarrollo de software.
5. **Iterativo e Incremental:** Permite abordar proyectos complejos de manera incremental, mejorando la capacidad de respuesta a los cambios y la retroalimentación continua.

Desventajas del Proceso Unificado

1. **Complejidad y Costo:** Implementar y seguir RUP puede ser complejo y costoso, especialmente para equipos pequeños o proyectos de menor envergadura.
2. **Sobrecarga de Documentación:** La cantidad de documentación requerida puede ser significativa, lo que puede resultar en una sobrecarga para el equipo de desarrollo.
3. **Curva de Aprendizaje:** La adopción de RUP puede requerir una capacitación considerable, especialmente para equipos sin experiencia previa en el modelo.

Aplicabilidad del Proceso Unificado

RUP es especialmente útil para proyectos de software grandes y complejos donde:

- Se requiere una arquitectura robusta y bien definida.
- La calidad y la gestión de riesgos son críticas.
- Es necesario tener una trazabilidad clara y documentación detallada.
- Los proyectos tienen requisitos que pueden evolucionar durante el ciclo de vida del desarrollo.

En resumen, el Proceso Unificado de Desarrollo de Software (RUP) ofrece un enfoque estructurado y adaptable que incorpora las mejores prácticas del desarrollo de software, proporcionando un marco robusto para la gestión de proyectos complejos y la entrega de sistemas de alta calidad.

3.5. Modelo de Proceso de Software IEEE.

El Modelo de Proceso de Software IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) establece un estándar para la planificación, ejecución y gestión del desarrollo de software. Este modelo se documenta en la norma IEEE 12207, que es un estándar reconocido internacionalmente para los procesos de ciclo de vida de software. Su objetivo principal es proporcionar un marco integral para todas las actividades asociadas con la adquisición, suministro, desarrollo, operación, mantenimiento y retiro de software.

Estructura del Modelo de Proceso de Software IEEE

La norma IEEE 12207 se organiza en tres categorías principales de procesos:

1. **Procesos de Ciclo de Vida Primarios**
2. **Procesos de Apoyo**
3. **Procesos Organizacionales**

1. Procesos de Ciclo de Vida Primarios

Estos procesos cubren las actividades fundamentales desde la concepción hasta el retiro del software. Incluyen:

Adquisición (Acquisition)

- **Descripción:** Proceso en el que una organización obtiene un producto de software o servicio.
- **Actividades Clave:**
 - Definir y documentar los requisitos de adquisición.
 - Selección del proveedor.
 - Contratación y gestión del contrato.
 - Aceptación del producto.

Suministro (Supply)

- **Descripción:** Proceso en el que una organización proporciona un producto de software o servicio a otra organización.
- **Actividades Clave:**
 - Preparación de la oferta.
 - Gestión del proyecto de suministro.
 - Entrega y soporte del producto.

Desarrollo (Development)

- **Descripción:** Proceso que abarca la creación del software, desde la concepción hasta el despliegue.

- **Actividades Clave:**
 - Análisis de requisitos.
 - Diseño del sistema y software.
 - Implementación y codificación.
 - Integración y pruebas.
 - Entrega y despliegue.

*Descripción: **Proceso de utilización del software en su entorno operativo.***

- **Actividades Clave:**
 - Operación diaria del software.
 - Gestión de datos y rendimiento.
 - Monitoreo y control del sistema.
 - Mantenimiento (Maintenance)
- **Descripción:** Proceso de modificar el software después de su entrega para corregir defectos, mejorar el rendimiento o adaptarlo a un entorno cambiado.
- **Actividades Clave:**
 - Recepción y evaluación de solicitudes de cambio.
 - Implementación de cambios.
 - Verificación y validación de cambios.
 - Actualización de la documentación.

2. Procesos de Apoyo

Estos procesos proporcionan apoyo a los procesos de ciclo de vida primarios y aseguran la calidad y eficiencia del software.

Documentación (Documentation)

- **Descripción:** Proceso de creación y gestión de documentos necesarios durante el ciclo de vida del software.
- **Actividades Clave:**
 - Creación de manuales de usuario y técnicos.
 - Mantenimiento y actualización de la documentación

Gestión de Configuración (Configuration Management)

- **Descripción:** Proceso de identificar, controlar, auditar y dar cuenta de los elementos de configuración del software.
- **Actividades Clave:**
 - Control de versiones y cambios.
 - Auditorías de configuración.

Gestión de la Calidad (Quality Management)

- **Descripción:** Proceso de asegurar que el software cumple con los requisitos de calidad especificados.
- **Actividades Clave:**
 - Planificación y aseguramiento de la calidad.
 - Revisión y auditorías de calidad.

Verificación (Verification)

- **Descripción:** Proceso de evaluar si los productos de software cumplen con los requisitos especificados.
- **Actividades Clave:**
 - Revisiones, inspecciones y pruebas de los productos.

Validación (Validation)

- **Descripción:** Proceso de asegurar que el software cumple con las necesidades y expectativas del usuario.
- **Actividades Clave:**
 - Pruebas de aceptación por parte del usuario.

Revisión Conjunta (Joint Review)

- **Descripción:** Proceso de realizar evaluaciones y revisiones conjuntas con todas las partes interesadas.
- **Actividades Clave:**
 - Reuniones de revisión y evaluación del progreso.

Auditoría (Audit)

- **Descripción:** Proceso de revisión independiente del software y de los procesos relacionados.
- **Actividades Clave:**
 - Planificación y realización de auditorías.
 - Informe de auditoría y seguimiento.

Resolución de Problemas (Problem Resolution)

- **Descripción:** Proceso de identificar, analizar y resolver problemas encontrados durante el ciclo de vida del software.
- **Actividades Clave:**
 - Registro y clasificación de problemas.
 - Resolución y seguimiento de problemas.

3. Procesos Organizacionales

Estos procesos se centran en establecer y mantener la infraestructura y el entorno necesario para soportar los procesos de ciclo de vida y de apoyo.

Gestión (Management)

- **Descripción:** Proceso de planificar, dirigir, controlar y coordinar todas las actividades del ciclo de vida del software.
- **Actividades Clave:**
 - Planificación y organización del proyecto.
 - Monitoreo y control del proyecto.
 - Gestión de recursos y riesgos.

Infraestructura (Infrastructure)

- **Descripción:** Proceso de proporcionar los recursos, entornos y herramientas necesarias para apoyar el desarrollo y mantenimiento del software.
- **Actividades Clave:**
 - Provisión de entornos de desarrollo y pruebas.
 - Mantenimiento de herramientas y tecnologías.

Mejora (Improvement)

- **Descripción:** Proceso de identificar y llevar a cabo mejoras en los procesos de desarrollo y mantenimiento del software.
- **Actividades Clave:**
 - Evaluación de procesos actuales.
 - **Implementación de mejoras continuas.**

Entrenamiento (Training)

- **Descripción:** Proceso de desarrollar y mantener la competencia del personal involucrado en el ciclo de vida del software.
- **Actividades Clave:**
 - Planificación y ejecución de programas de entrenamiento.
 - Evaluación de la efectividad del entrenamiento.

Ventajas del Modelo de Proceso de Software IEEE

1. **Estandarización:** Proporciona un marco estándar y reconocido internacionalmente que asegura la consistencia y la calidad del software.
2. **Flexibilidad:** Puede ser adaptado a las necesidades específicas de diferentes proyectos y organizaciones.
3. **Mejora de la Calidad:** La integración de procesos de verificación y validación asegura que el software cumple con los requisitos y expectativas del cliente.

4. **Gestión Eficiente:** Facilita una gestión eficiente de proyectos mediante la planificación, monitoreo y control estructurado.
5. **Reducción de Riesgos:** La identificación y gestión temprana de riesgos ayuda a minimizar problemas futuros.

Desventajas del Modelo de Proceso de Software IEEE

1. **Complejidad:** La implementación del modelo puede ser compleja y requerir un esfuerzo considerable en la adaptación y capacitación del personal.
2. **Costo:** Puede ser costoso, especialmente para organizaciones pequeñas o proyectos con recursos limitados.
3. **Sobrecarga de Documentación:** La cantidad de documentación requerida puede resultar en una carga adicional para los equipos de desarrollo.

Aplicabilidad del Modelo de Proceso de Software IEEE

El Modelo de Proceso de Software IEEE es especialmente aplicable a:

- **Proyectos Grandes y Complejos:** Donde la estandarización y la gestión eficiente son cruciales.
- **Organizaciones que Buscan Certificación:** Aquellas que desean cumplir con estándares internacionales y obtener certificaciones de calidad.
- **Proyectos Críticos:** Donde la calidad y la fiabilidad del software son de suma importancia, como en sectores aeroespacial, médico, y financiero.

En resumen, el Modelo de Proceso de Software IEEE proporciona un marco integral y estructurado para la gestión del ciclo de vida del software, promoviendo la calidad, consistencia y eficiencia en el desarrollo y mantenimiento de sistemas de software.

3.6. Herramientas CASE

Las Herramientas CASE (Computer-Aided Software Engineering) son aplicaciones de software que ayudan a los desarrolladores y gerentes de proyectos a automatizar y gestionar diversas tareas en el proceso de desarrollo de software. Estas herramientas pueden ser utilizadas en diferentes fases del ciclo de vida del software, desde el análisis de requisitos hasta el mantenimiento del sistema, proporcionando soporte para diseño, codificación, pruebas, documentación y más.

Tipos de Herramientas CASE

Las herramientas CASE se pueden clasificar en varias categorías, según las fases del ciclo de vida del software que soportan:

1. **Herramientas CASE Front-End:** Se enfocan en las fases iniciales del desarrollo, como la planificación, el análisis y el diseño.

2. **Herramientas CASE Back-End:** Soportan las fases posteriores, como la implementación, las pruebas y el mantenimiento.
3. **Herramientas CASE Integradas:** Proporcionan soporte a través de todo el ciclo de vida del software, integrando funcionalidades de las herramientas front-end y back-end.

Funcionalidades de las Herramientas CASE

Las herramientas CASE ofrecen una amplia gama de funcionalidades, incluyendo:

1. **Modelado y Diseño**
 - Diagramas de flujo de datos (DFD)
 - Diagramas de entidad-relación (ERD)
 - Modelado UML (Unified Modeling Language)
 - Diseño de bases de datos
2. **Generación de Código**
 - Generación automática de código fuente a partir de modelos de diseño
 - Plantillas de código y asistentes
3. **Gestión de Requisitos**
 - Captura y documentación de requisitos
 - Seguimiento y trazabilidad de requisitos
4. **Gestión de Proyectos**
 - Planificación y programación de proyectos
 - Gestión de recursos
 - Control de versiones y configuración
5. **Pruebas y Verificación**
 - Generación de casos de prueba
 - Automatización de pruebas
 - Análisis estático y dinámico del código
6. **Documentación**
 - Generación automática de documentación técnica y de usuario
 - Mantenimiento de manuales y guías
7. **Colaboración y Comunicación**
 - Herramientas de colaboración en equipo
 - Integración con sistemas de gestión de proyectos y control de versiones

Ejemplos de Herramientas CASE

Aquí hay algunos ejemplos populares de herramientas CASE, clasificadas según sus funcionalidades principales:

Herramientas CASE Front-End

1. **IBM Rational Rose:** Herramienta de modelado UML que soporta el análisis y diseño orientado a objetos.

2. **Enterprise Architect:** Plataforma completa de modelado, diseño y gestión de proyectos basada en UML.
3. **Lucidchart:** Herramienta en línea para la creación de diagramas de flujo, ERD y otros diagramas de diseño.

Herramientas CASE Back-End

1. **Eclipse:** Entorno de desarrollo integrado (IDE) que soporta la codificación, pruebas y depuración de software.
2. **JIRA:** Herramienta de gestión de proyectos y seguimiento de incidencias que soporta el control de versiones y la gestión de requisitos.
3. **Selenium:** Herramienta para la automatización de pruebas de aplicaciones web.

Herramientas CASE Integradas

1. **IBM Rational Software Architect:** Plataforma de diseño y desarrollo de software que soporta modelado, generación de código, pruebas y gestión de cambios.
2. **Visual Paradigm:** Herramienta de modelado y desarrollo de software que soporta UML, gestión de requisitos, generación de código y documentación.
3. **Oracle SQL Developer:** Herramienta para el desarrollo y gestión de bases de datos que incluye funcionalidades de modelado y generación de código SQL.

Ventajas de las Herramientas CASE

1. **Aumento de la Productividad:** Automatizan tareas repetitivas y complejas, permitiendo a los desarrolladores concentrarse en aspectos más creativos y críticos del desarrollo.
2. **Mejora de la Calidad del Software:** Facilitan la creación de diseños precisos y consistentes, y soportan la detección temprana de errores.
3. **Facilitación de la Colaboración:** Proporcionan un entorno integrado donde los equipos pueden trabajar juntos de manera más eficiente.
4. **Trazabilidad y Documentación:** Ayudan a mantener una documentación completa y actualizada, lo que facilita la trazabilidad de los requisitos y cambios a lo largo del ciclo de vida del software.

Desventajas de las Herramientas CASE

1. **Costo:** Algunas herramientas CASE pueden ser costosas, tanto en términos de licencias como de capacitación del personal.
2. **Curva de Aprendizaje:** Puede requerir tiempo y esfuerzo significativo para que los equipos se familiaricen y se vuelvan competentes en el uso de estas herramientas.

3. **Dependencia Tecnológica:** La adopción de una herramienta CASE puede crear dependencia en la tecnología específica, lo que puede ser un riesgo si la herramienta deja de ser soportada o se discontinúa.

Aplicabilidad de las Herramientas CASE

Las herramientas CASE son aplicables a una amplia variedad de proyectos y entornos de desarrollo, incluyendo:

- **Proyectos Grandes y Complejos:** Donde la gestión de requisitos, diseño y trazabilidad son cruciales.
- **Desarrollo Orientado a Objetos:** Herramientas que soportan UML y otros métodos de diseño orientado a objetos.
- **Entornos de Desarrollo Ágil:** Herramientas que facilitan la colaboración y la gestión ágil de proyectos.
- **Desarrollo Basado en Componentes:** Herramientas que soportan la reutilización de componentes y la integración de sistemas.

En resumen, las herramientas CASE son esenciales para mejorar la eficiencia y la calidad en el desarrollo de software, proporcionando un entorno automatizado y estructurado que apoya todas las fases del ciclo de vida del software. Su correcta implementación y uso pueden significar una ventaja competitiva significativa en la entrega de proyectos de software.

Cada uno de estos modelos tiene sus propias ventajas y desventajas, y la elección del modelo correcto puede tener un impacto significativo en el éxito del proyecto de desarrollo de sistemas de información

Referencias

1. **Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach.** Este libro ofrece una visión detallada de los diferentes modelos de desarrollo de software y su aplicación en la ingeniería de software.
2. **Sommerville, I. (2015). Software Engineering.** Proporciona una descripción comprensiva de los modelos prescriptivos y su relevancia en el desarrollo de sistemas de información.
3. **Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement.** Este artículo presenta el modelo en espiral y discute su aplicación práctica en el desarrollo de software.
4. **Royce, W. W. (1970). Managing the Development of Large Software Systems.** Un documento seminal que describe el modelo en cascada y sus implicaciones para la gestión de proyectos de software.
5. **Boehm, B. W. (1981). Software Engineering Economics.** Este libro ofrece una perspectiva económica sobre el desarrollo de software y analiza diferentes modelos de desarrollo en términos de costos y beneficios.

UNIDAD IV PARADIGMAS DE LA INGENIERIA DE SOFTWARE

En la ingeniería de software, los paradigmas son enfoques o metodologías que guían el proceso de desarrollo de software. Cada paradigma proporciona un conjunto de principios y prácticas que ayudan a los desarrolladores a enfrentar diferentes desafíos y problemas en el desarrollo de software. Aquí te presento algunos de los paradigmas más comunes en la ingeniería de software:

1. Paradigma Estructurado

Se basa en la división del programa en módulos o subrutinas más pequeñas y manejables. Se utiliza un enfoque secuencial y se enfatiza en la claridad y la simplicidad del código.

- **Ejemplo:** Programación estructurada con lenguajes como C y Pascal.

2. Paradigma Orientado a Objetos (POO)

Este paradigma se centra en la creación de objetos que representan entidades del mundo real. Los objetos tienen atributos (datos) y métodos (funciones) que manipulan esos datos. Promueve el encapsulamiento, la herencia y el polimorfismo.

- **Ejemplo:** Programación orientada a objetos con lenguajes como Java, C++ y Python.

3. Paradigma Funcional

Se basa en funciones matemáticas puras y evita el estado mutable y los efectos secundarios. Promueve el uso de funciones como ciudadanos de primera clase y el concepto de inmutabilidad.

- **Ejemplo:** Programación funcional con lenguajes como Haskell, Lisp y Scala.

4. Paradigma Lógico

Este enfoque se basa en la lógica matemática y la inferencia. Los programas se definen mediante reglas y hechos, y la ejecución del programa consiste en resolver consultas mediante un proceso de deducción.

- **Ejemplo:** Programación lógica con lenguajes como Prolog.

5. Paradigma de Componentes

Se centra en la construcción de software mediante la integración de componentes reutilizables y bien definidos. Cada componente tiene una interfaz clara y puede ser desarrollado y probado independientemente.

- **Ejemplo:** Desarrollo basado en componentes con tecnologías como COM, CORBA y EJB.

6. Paradigma de Desarrollo Ágil

Más que un paradigma de programación, es un enfoque para la gestión y ejecución de proyectos de software. Enfatiza la flexibilidad, la colaboración con el cliente, y la entrega continua de incrementos de software funcional.

- **Ejemplo:** Metodologías ágiles como Scrum, Kanban y XP (Extreme Programming).

7. Paradigma de Desarrollo Basado en Modelos (MDD)

Enfatiza la creación de modelos abstractos del sistema que pueden ser transformados automáticamente en código ejecutable mediante herramientas de generación de código.

- **Ejemplo:** UML (Unified Modeling Language) y herramientas de MDD como Enterprise Architect.

8. Paradigma Reactivo

Se enfoca en la construcción de sistemas que reaccionan a eventos, siendo altamente responsivos, resilientes y escalables. Utiliza conceptos como flujos de datos y programación orientada a eventos.

- **Ejemplo:** Programación reactiva con frameworks y bibliotecas como RxJava y Akka.

Cada paradigma tiene sus propias ventajas y desventajas, y la elección de uno u otro depende del tipo de problema que se esté tratando de resolver, el contexto del proyecto y las preferencias del equipo de desarrollo.

4.1. El enfoque estructurado

El enfoque estructurado en la ingeniería de software es un paradigma que se enfoca en mejorar la claridad, calidad y desarrollo del software mediante la división del programa en módulos más pequeños y manejables. Este paradigma surgió como respuesta a la creciente complejidad del software, ofreciendo una manera sistemática de diseñar y escribir programas. Aquí te detallo sus características, ventajas, desventajas y ejemplos:

Características del Enfoque Estructurado

1. **División del Programa en Módulos:**
 - El programa se divide en funciones o subrutinas más pequeñas y manejables.
 - Cada módulo tiene una única entrada y una única salida.
2. **Uso de Estructuras de Control:**
 - Utiliza estructuras de control básicas: secuencia, selección (if-else), e iteración (bucles).
 - Evita el uso de saltos incontrolados como el comando goto, que puede dificultar la comprensión del flujo del programa.
3. **Diseño Descendente (Top-Down):**
 - El desarrollo comienza con una visión general del sistema y se descompone en componentes más específicos.
 - Facilita la comprensión global del sistema y su estructura.
4. **Claridad y Simplicidad:**
 - Promueve la escritura de código claro y sencillo, facilitando el mantenimiento y la evolución del software.
 - Hace hincapié en la documentación y en los comentarios para aumentar la legibilidad.

Ventajas del Enfoque Estructurado

1. **Facilita el Mantenimiento:**
 - La división en módulos independientes facilita la localización y corrección de errores.
 - Los cambios en una parte del programa suelen tener un impacto mínimo en otras partes.
2. **Mejora la Comprensión:**
 - La estructura clara y modular hace que el código sea más fácil de entender y seguir.
 - La documentación y los comentarios ayudan a nuevos desarrolladores a comprender rápidamente el funcionamiento del programa.
3. **Reutilización de Código:**
 - Los módulos pueden ser reutilizados en diferentes programas, reduciendo el esfuerzo de desarrollo.
4. **Desarrollo Colaborativo:**
 - Los equipos pueden trabajar en diferentes módulos de manera simultánea, lo que puede acelerar el proceso de desarrollo.

Desventajas del Enfoque Estructurado

1. **Rigidez:**
 - Puede ser menos flexible que otros enfoques modernos, como la programación orientada a objetos.
 - No es tan adecuado para sistemas muy grandes o complejos.
2. **Dificultad para Manejar Datos Complejos:**

- Manejar estructuras de datos complejas puede ser más complicado y menos intuitivo que en otros paradigmas como el orientado a objetos.

3. Escalabilidad Limitada:

- A medida que el tamaño y la complejidad del software aumentan, la gestión de múltiples módulos puede volverse más difícil.

Ejemplos de Uso

- **Lenguajes de Programación:** Lenguajes como C, Pascal y Fortran son ejemplos clásicos donde se aplica el enfoque estructurado.
- **Sistemas Operativos y Software de Sistemas:** Muchas partes de sistemas operativos y software de sistemas críticos se desarrollaron inicialmente utilizando este enfoque debido a su claridad y control sobre el flujo del programa.
- **Aplicaciones de Procesamiento de Datos:** Programas que requieren procesamiento secuencial y análisis de datos.

Ejemplo de Código en C

```
c
Copiar código
#include <stdio.h>

// Función para sumar dos números
int suma(int a, int b) {
    return a + b;
}

// Función principal
int main() {
    int num1, num2, resultado;

    // Solicitar al usuario dos números
    printf("Ingrese el primer número: ");
    scanf("%d", &num1);
    printf("Ingrese el segundo número: ");
    scanf("%d", &num2);

    // Llamar a la función suma
    resultado = suma(num1, num2);

    // Mostrar el resultado
    printf("El resultado de la suma es: %d\n", resultado);

    return 0;
}
```

En este ejemplo, el programa se estructura en una función principal (main) y una función auxiliar (suma), demostrando la división del programa en módulos más pequeños y manejables, característica fundamental del enfoque estructurado.

El enfoque estructurado sigue siendo relevante en muchos contextos, especialmente en proyectos donde la simplicidad y la claridad son prioritarias. Sin embargo, para proyectos más complejos, puede ser beneficioso considerar otros paradigmas que ofrecen mayor flexibilidad y mejores herramientas para manejar la complejidad.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A4.1.1.

4.2. El enfoque orientado a objetos

El enfoque orientado a objetos (POO) es un paradigma de programación que organiza el software en torno a objetos, que son instancias de clases. Este paradigma se centra en encapsular datos y comportamientos en unidades cohesivas, facilitando la modularidad, la reutilización y el mantenimiento del código. A continuación, se detallan las características, ventajas, desventajas y ejemplos del enfoque orientado a objetos:

Características del Enfoque Orientado a Objetos

1. Clases y Objetos:

- **Clases:** Definiciones de tipos de datos que incluyen atributos (datos) y métodos (funciones) que operan sobre esos datos.
- **Objetos:** Instancias de clases que representan entidades concretas con estado y comportamiento.

2. Encapsulamiento:

- Los datos y los métodos se agrupan dentro de las clases, ocultando los detalles internos y exponiendo solo lo necesario a través de interfaces públicas.

3. Herencia:

- Permite crear nuevas clases basadas en clases existentes, reutilizando y extendiendo su comportamiento.
- Facilita la creación de jerarquías de clases.

4. Polimorfismo:

- Permite que diferentes clases respondan a la misma interfaz o método de diferentes maneras.
- Incluye polimorfismo de sobrecarga y polimorfismo de inclusión (a través de la herencia).

5. Abstracción:

- Facilita la creación de modelos simplificados del sistema real, centrándose en los aspectos más relevantes y omitiendo los detalles innecesarios.

Ventajas del Enfoque Orientado a Objetos

1. **Reutilización de Código:**
 - La herencia y la modularidad permiten reutilizar clases y objetos en diferentes partes de un programa o en distintos proyectos.
2. **Facilidad de Mantenimiento:**
 - La encapsulación y la modularidad hacen que el código sea más fácil de mantener y actualizar, ya que los cambios en una parte del sistema suelen tener un impacto limitado en otras partes.
3. **Modelado Natural del Mundo Real:**
 - Los objetos permiten representar entidades del mundo real de manera intuitiva, facilitando el diseño y la comprensión del sistema.
4. **Flexibilidad y Extensibilidad:**
 - Las clases pueden ser extendidas y modificadas sin afectar el código existente, lo que facilita la evolución del software.

Desventajas del Enfoque Orientado a Objetos

1. **Complejidad Inicial:**
 - Puede ser más difícil de aprender y dominar en comparación con enfoques más simples, especialmente para programadores novatos.
2. **Sobrecarga de Rendimiento:**
 - La abstracción y el manejo de objetos pueden introducir sobrecargas en el rendimiento en comparación con enfoques más directos.
3. **Diseño Complejo:**
 - El diseño de jerarquías de clases y la implementación de relaciones entre objetos pueden volverse complejas en sistemas grandes y complicados.

Ejemplos de Uso

- **Lenguajes de Programación:** Lenguajes como Java, C++, Python, C#, y Ruby son ampliamente utilizados en la programación orientada a objetos.
- **Desarrollo de Aplicaciones Empresariales:** Muchas aplicaciones empresariales utilizan POO para manejar la complejidad y facilitar el mantenimiento.
- **Desarrollo de Juegos:** La POO permite modelar entidades del juego como personajes, objetos y escenarios de manera intuitiva y modular.

Ejemplo de Código en Java

```
java
Copiar código
// Definición de la clase Persona
public class Persona {
    // Atributos
    private String nombre;
```

```

private int edad;

// Constructor
public Persona(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
}

// Método para obtener el nombre
public String getNombre() {
    return nombre;
}

// Método para obtener la edad
public int getEdad() {
    return edad;
}

// Método para mostrar información de la persona
public void mostrarInformacion() {
    System.out.println("Nombre: " + nombre + ", Edad: " + edad);
}
}

// Clase principal para probar la clase Persona
public class Main {
    public static void main(String[] args) {
        // Crear un objeto de la clase Persona
        Persona persona1 = new Persona("Juan", 30);

        // Usar métodos del objeto persona1
        persona1.mostrarInformacion();
    }
}

```

En este ejemplo, se define una clase Persona con atributos nombre y edad, un constructor, y métodos para acceder a los atributos y mostrar información. En la clase Main, se crea un objeto persona1 y se utiliza para mostrar su información.

El enfoque orientado a objetos es ampliamente utilizado debido a su capacidad para manejar la complejidad del software y facilitar su mantenimiento y evolución. Aunque puede tener una curva de aprendizaje más pronunciada, sus beneficios en términos de modularidad y reutilización de código son significativos en proyectos a gran escala.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A4.2.1 y A4.2.2

4.3 Enfoques de vanguardia

Los enfoques de vanguardia en la ingeniería de software están en constante evolución, impulsados por los avances tecnológicos y las nuevas necesidades del mercado. Aquí te presento algunos de los enfoques más modernos y relevantes:

1. Desarrollo Basado en DevOps

DevOps es un enfoque que integra las prácticas de desarrollo de software (Dev) y operaciones de TI (Ops) para mejorar la colaboración y productividad. Este enfoque promueve la automatización y monitoreo continuo a lo largo de todo el ciclo de vida del software, desde la integración y las pruebas hasta la entrega y el despliegue.

- **Principales características:**
 - **Integración Continua/Entrega Continua (CI/CD):** Automatización de la integración y entrega del código.
 - **Infraestructura como Código (IaC):** Gestión y aprovisionamiento de la infraestructura mediante código.
 - **Monitoreo y Logging:** Seguimiento continuo del rendimiento y la salud del sistema.
 -

2. Desarrollo Basado en Microservicios

El desarrollo basado en **microservicios** implica descomponer una aplicación monolítica en servicios más pequeños, independientes y desplegables de manera autónoma. Cada microservicio es responsable de una funcionalidad específica y se comunica con otros microservicios a través de interfaces bien definidas, generalmente APIs RESTful.

- **Principales características:**
 - **Independencia:** Los microservicios pueden ser desarrollados, desplegados y escalados de manera independiente.
 - **Resiliencia:** Aislamiento de fallos; el fallo de un microservicio no afecta a los demás.
 - **Escalabilidad:** Escalado horizontal de microservicios según la demanda.

3. Desarrollo Basado en la Nube (Cloud-Native Development)

El **desarrollo basado en la nube** se centra en crear aplicaciones que se diseñan específicamente para operar en entornos de nube. Este enfoque utiliza servicios y tecnologías proporcionados por proveedores de nube como AWS, Azure y Google Cloud.

- **Principales características:**
 - **Elasticidad:** Capacidad para escalar recursos según la demanda.
 - **Resiliencia:** Diseño para tolerancia a fallos y alta disponibilidad.
 - **Automatización:** Uso de herramientas de automatización para el despliegue y gestión de aplicaciones.

4. Desarrollo Basado en Inteligencia Artificial y Machine Learning (AI/ML)

La integración de **inteligencia artificial y machine learning** en el desarrollo de software está revolucionando la manera en que se crean y operan las aplicaciones. Las aplicaciones pueden adaptarse y mejorar automáticamente basándose en los datos y el aprendizaje.

- **Principales características:**
 - **Personalización:** Aplicaciones que pueden adaptar su comportamiento a las preferencias del usuario.
 - **Automatización Inteligente:** Uso de algoritmos para automatizar tareas complejas.
 - **Análisis Predictivo:** Capacidad para predecir tendencias y comportamientos futuros basándose en datos históricos.

5. Desarrollo de Aplicaciones Híbridas y Multiplataforma

El **desarrollo híbrido y multiplataforma** permite crear aplicaciones que funcionan en múltiples plataformas (iOS, Android, web) a partir de una única base de código. Herramientas como React Native, Flutter y Xamarin son populares en este enfoque.

- **Principales características:**
 - **Reutilización de Código:** Un solo código base para múltiples plataformas.
 - **Consistencia de la Experiencia del Usuario:** Interfaces y comportamientos consistentes en diferentes dispositivos.
 - **Reducción de Costos:** Menor tiempo y esfuerzo en desarrollo y mantenimiento.

6. Desarrollo Basado en Contenedores

El **desarrollo basado en contenedores** utiliza tecnologías como Docker y Kubernetes para empaquetar aplicaciones y sus dependencias en contenedores, asegurando que funcionen de manera consistente en diferentes entornos.

- **Principales características:**
 - **Portabilidad:** Los contenedores pueden ejecutarse en cualquier entorno que soporte Docker.
 - **Aislamiento:** Cada contenedor opera en su propio espacio aislado.

- **Escalabilidad:** Fácil escalado horizontal mediante la orquestación de contenedores.

7. Desarrollo Guiado por Pruebas (TDD) y Desarrollo Guiado por Comportamiento (BDD)

TDD y **BDD** son enfoques que enfatizan la importancia de las pruebas desde las primeras etapas del desarrollo. TDD se centra en escribir pruebas unitarias antes de escribir el código de producción, mientras que BDD se centra en la especificación del comportamiento de la aplicación en un lenguaje comprensible para todas las partes interesadas.

- **Principales características:**
 - **Calidad del Código:** Mejora la calidad y reducen los errores.
 - **Documentación Viva:** Las pruebas actúan como documentación del comportamiento del sistema.
 - **Feedback Rápido:** Identificación temprana de errores y problemas de diseño.

Estos enfoques de vanguardia están transformando la ingeniería de software, proporcionando nuevas herramientas y metodologías para abordar la complejidad y los desafíos del desarrollo moderno. La elección del enfoque adecuado depende del contexto del proyecto, las necesidades específicas del negocio y las habilidades del equipo de desarrollo.

Referencias Generales

1. **Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach.** Este libro cubre una amplia gama de paradigmas y metodologías en la ingeniería de software, proporcionando una base sólida para entender sus principios y aplicaciones.
2. **Sommerville, I. (2015). Software Engineering.** Ofrece una visión detallada de los diferentes paradigmas de desarrollo de software y su evolución a lo largo del tiempo.
3. **Larman, C. (2004). Agile and Iterative Development: A Manager's Guide.** Un recurso clave para entender los principios y prácticas de los métodos ágiles en la ingeniería de software.
4. **Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software.** Este libro es fundamental para el paradigma orientado a objetos y proporciona soluciones probadas a problemas comunes de diseño de software.
5. **Van Roy, P., & Haridi, S. (2004). Concepts, Techniques, and Models of Computer Programming.** Una referencia exhaustiva que abarca múltiples paradigmas de programación, incluyendo la programación funcional y basada en componentes.

UNIDAD V GESTIÓN DE PROYECTOS DE SISTEMAS DE INFORMACIÓN

La gestión de proyectos de sistemas de información (SI) implica la planificación, ejecución y supervisión de todas las actividades necesarias para desarrollar e implementar sistemas de información que satisfagan las necesidades de una organización. Aquí te presento los componentes clave, las metodologías y las mejores prácticas para la gestión de proyectos de SI:

Componentes Clave de la Gestión de Proyectos de SI

1. **Alcance del Proyecto:**
 - Definición clara y detallada de los objetivos, entregables y límites del proyecto.
 - Incluye la identificación de las necesidades del negocio y los requisitos funcionales y no funcionales.
2. **Planificación:**
 - Creación de un plan de proyecto detallado que incluya cronogramas, recursos, presupuestos y riesgos.
 - Establecimiento de hitos y fechas de entrega.
3. **Ejecución:**
 - Implementación del plan de proyecto mediante la asignación de tareas y la gestión del equipo de trabajo.
 - Monitoreo continuo del progreso y ajustes necesarios para mantener el proyecto en curso.
4. **Monitoreo y Control:**
 - Seguimiento del desempeño del proyecto en términos de tiempo, costo y calidad.
 - Identificación y gestión de riesgos y problemas que puedan surgir durante el desarrollo.
5. **Cierre del Proyecto:**
 - Verificación de que todos los entregables cumplen con los requisitos establecidos.
 - Documentación de las lecciones aprendidas y formalización de la aceptación del proyecto.

Metodologías de Gestión de Proyectos de SI

1. **Metodologías Tradicionales (Waterfall):**
 - Enfoque secuencial donde cada fase del proyecto debe completarse antes de pasar a la siguiente.
 - Adecuado para proyectos con requisitos bien definidos y poco propensos a cambios.
2. **Metodologías Ágiles:**
 - Enfoques iterativos e incrementales que permiten adaptarse a cambios en los requisitos y prioridades del proyecto.
 - Ejemplos: Scrum, Kanban, Extreme Programming (XP).
3. **Metodologías Híbridas:**

- Combinan elementos de metodologías tradicionales y ágiles para aprovechar las ventajas de ambos enfoques.
- Ejemplo: Scrumban.
- 4. **Metodologías Basadas en el Valor:**
 - Enfocadas en maximizar el valor entregado al cliente mediante la priorización de características y funcionalidades.
 - Ejemplo: Lean Software Development.

Herramientas de Gestión de Proyectos de SI

1. **Software de Gestión de Proyectos:**
 - Herramientas como Microsoft Project, Jira, Trello, Asana y Monday.com para planificar, rastrear y colaborar en proyectos.
2. **Herramientas de Control de Versiones:**
 - Sistemas como Git y Subversion (SVN) para gestionar el código fuente y las versiones del software.
3. **Herramientas de Comunicación y Colaboración:**
 - Plataformas como Slack, Microsoft Teams y Zoom para facilitar la comunicación y colaboración en equipo.
4. **Herramientas de Integración Continua/Entrega Continua (CI/CD):**
 - Jenkins, Travis CI, CircleCI para automatizar el proceso de integración y despliegue del software.

Mejores Prácticas en la Gestión de Proyectos de SI

1. **Definición Clara de Requisitos:**
 - Trabajar estrechamente con los stakeholders para entender y documentar los requisitos del proyecto.
2. **Comunicación Efectiva:**
 - Mantener una comunicación abierta y frecuente con todos los miembros del equipo y las partes interesadas.
3. **Gestión de Riesgos:**
 - Identificar, evaluar y planificar la mitigación de riesgos potenciales desde el inicio del proyecto.
4. **Flexibilidad y Adaptabilidad:**
 - Estar preparado para adaptarse a cambios en los requisitos y el entorno del proyecto.
5. **Revisión y Mejora Continua:**
 - Realizar revisiones periódicas del proyecto y utilizar las lecciones aprendidas para mejorar futuros proyectos.
6. **Enfoque en la Calidad:**
 - Implementar prácticas de aseguramiento de la calidad y realizar pruebas rigurosas para garantizar que el sistema cumple con los estándares de calidad.

Fases del Ciclo de Vida del Proyecto de SI

- 1. Iniciación:**
 - Definición del proyecto, establecimiento de objetivos y obtención de la aprobación de los stakeholders.
- 2. Planificación:**
 - Detallado de las actividades, cronogramas, recursos, presupuestos y riesgos del proyecto.
- 3. Ejecución:**
 - Implementación del plan, asignación de tareas y gestión del equipo de trabajo.
- 4. Monitoreo y Control:**
 - Seguimiento del progreso del proyecto y realización de ajustes según sea necesario.
- 5. Cierre:**
 - Finalización del proyecto, verificación de entregables, documentación de lecciones aprendidas y formalización de la aceptación del proyecto.

La gestión de proyectos de sistemas de información es una disciplina compleja que requiere una combinación de habilidades técnicas y de gestión. Al aplicar metodologías adecuadas y seguir mejores prácticas, los equipos de proyecto pueden aumentar sus probabilidades de éxito y entregar sistemas de información que satisfagan las necesidades de sus organizaciones.

5.1. Actividades de gestión.

Las actividades de gestión de proyectos de sistemas de información (SI) abarcan una serie de tareas y responsabilidades clave que aseguran la planificación, ejecución, monitoreo y cierre exitoso del proyecto. Aquí se detallan las principales actividades de gestión que se llevan a cabo en cada fase del ciclo de vida del proyecto:

1. Iniciación

- **Definición del Proyecto:**
 - Identificar y documentar el propósito, objetivos y beneficios del proyecto.
 - Establecer el alcance preliminar del proyecto.
- **Identificación de Stakeholders:**
 - Identificar a todas las partes interesadas (stakeholders) y sus expectativas.
 - Crear un registro de stakeholders.
- **Desarrollo del Acta de Constitución del Proyecto:**
 - Documentar la aprobación formal del proyecto.
 - Definir el gerente del proyecto y su autoridad.

- **Análisis de Viabilidad:**
 - Evaluar la viabilidad técnica, económica y operativa del proyecto.
 - Realizar un análisis de costo-beneficio.

2. Planificación

- **Desarrollo del Plan de Proyecto:**
 - Crear un plan detallado que cubra todas las áreas de gestión del proyecto (alcance, tiempo, costo, calidad, recursos humanos, comunicaciones, riesgos y adquisiciones).
- **Definición del Alcance:**
 - Elaborar una declaración detallada del alcance.
 - Crear la estructura de desglose del trabajo (WBS).
- **Planificación de Cronograma:**
 - Definir actividades, secuenciar tareas, estimar la duración y desarrollar el cronograma del proyecto.
- **Planificación de Recursos:**
 - Identificar y asignar recursos necesarios (humanos, materiales, equipos).
 - Crear el plan de gestión de recursos.
- **Planificación del Presupuesto:**
 - Estimar costos y determinar el presupuesto.
 - Crear el plan de gestión de costos.
- **Planificación de la Calidad:**
 - Definir los estándares de calidad y cómo se cumplirán.
 - Crear el plan de gestión de la calidad.
- **Planificación de Riesgos:**
 - Identificar riesgos potenciales, evaluar su impacto y probabilidad.
 - Desarrollar estrategias de mitigación y un plan de gestión de riesgos.

3. Ejecución

- **Dirección y Gestión del Trabajo del Proyecto:**
 - Coordinar a las personas y recursos.
 - Ejecutar el plan de proyecto y gestionar las actividades de trabajo.
- **Aseguramiento de la Calidad:**
 - Realizar auditorías de calidad y revisar los resultados.
 - Implementar mejoras en los procesos.
- **Gestión de Recursos Humanos:**
 - Desarrollar y gestionar el equipo de proyecto.
 - Realizar evaluaciones de desempeño y resolver conflictos.
- **Gestión de Comunicaciones:**
 - Distribuir la información del proyecto de manera efectiva.
 - Gestionar las expectativas y necesidades de los stakeholders.
- **Adquisiciones del Proyecto:**
 - Realizar adquisiciones necesarias (compras, contrataciones).
 - Gestionar relaciones con proveedores y contratistas.

4. Monitoreo y Control

- **Monitoreo del Desempeño del Proyecto:**
 - Realizar seguimiento y medición del progreso del proyecto.
 - Utilizar indicadores clave de rendimiento (KPI) y métricas de proyecto.
- **Control de Cambios:**
 - Gestionar solicitudes de cambio y su impacto en el proyecto.
 - Actualizar documentos de proyecto y comunicar cambios.
- **Control de Calidad:**
 - Inspeccionar y verificar los entregables para asegurar que cumplen con los estándares de calidad.
- **Gestión de Riesgos:**
 - Monitorear riesgos identificados y nuevos riesgos.
 - Implementar planes de respuesta a riesgos cuando sea necesario.
- **Control de Cronograma:**
 - Comparar el progreso real con el cronograma planificado.
 - Tomar acciones correctivas para mantener el proyecto en curso.
- **Control de Costos:**
 - Comparar costos reales con el presupuesto.
 - Tomar acciones correctivas para gestionar desviaciones de costos.

5. Cierre

- **Cierre del Proyecto o Fase:**
 - Confirmar que todos los entregables cumplen con los requisitos.
 - Obtener la aceptación formal del cliente o patrocinador del proyecto.
- **Documentación de Lecciones Aprendidas:**
 - Recopilar y documentar experiencias y lecciones aprendidas durante el proyecto.
 - Facilitar reuniones de cierre para discutir lo que funcionó bien y lo que se puede mejorar.
- **Cierre Administrativo:**
 - Completar toda la documentación necesaria.
 - Archivar todos los documentos del proyecto para referencias futuras.
- **Liberación de Recursos:**
 - Despedir al equipo del proyecto y reasignar recursos a otros proyectos o tareas.
 - Realizar evaluaciones finales del equipo.
- **Entrega y Transición:**
 - Transferir la propiedad del producto final al cliente o a las operaciones.
 - Asegurar una transición sin problemas al soporte y mantenimiento.

Estas actividades de gestión son esenciales para asegurar que los proyectos de sistemas de información se completen a tiempo, dentro del presupuesto y con la calidad requerida, satisfaciendo las necesidades y expectativas de los stakeholders.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A5.1.1.

5.2. Estudio de viabilidad del proyecto.

El estudio de viabilidad del proyecto es una evaluación exhaustiva que se realiza para determinar si un proyecto es viable y vale la pena emprenderlo. Este estudio analiza varios aspectos del proyecto, incluyendo su viabilidad técnica, económica, legal, operativa y de programación. A continuación, se describen las etapas y componentes clave de un estudio de viabilidad de un proyecto de sistemas de información (SI):

1. Identificación del Proyecto

- **Definición del Proyecto:**
 - Describir el proyecto de manera clara y concisa.
 - Identificar los objetivos principales y los entregables esperados.
- **Contexto del Proyecto:**
 - Proporcionar el contexto y la justificación del proyecto.
 - Identificar las necesidades del negocio y los problemas que el proyecto pretende resolver.

2. Análisis de Viabilidad Técnica

- **Evaluación de Requisitos Técnicos:**
 - Identificar los requisitos técnicos necesarios para el proyecto.
 - Analizar la disponibilidad de la tecnología necesaria y su compatibilidad con los sistemas existentes.
- **Capacidades del Equipo:**
 - Evaluar las habilidades y competencias del equipo de proyecto.
 - Identificar si se necesitan formaciones adicionales o contratación de personal especializado.
- **Soluciones Alternativas:**
 - Explorar y evaluar diferentes soluciones tecnológicas que puedan satisfacer los requisitos del proyecto.
 - Comparar las ventajas y desventajas de cada alternativa.

3. Análisis de Viabilidad Económica

- **Estimación de Costos:**
 - Identificar y estimar todos los costos asociados con el proyecto (costos de desarrollo, implementación, mantenimiento, etc.).
- **Análisis de Beneficios:**
 - Identificar y cuantificar los beneficios esperados del proyecto (incremento de ingresos, reducción de costos, mejoras en eficiencia, etc.).
- **Análisis de Retorno de Inversión (ROI):**

- Calcular el retorno de inversión proyectado para evaluar la rentabilidad del proyecto.
- Realizar análisis de costo-beneficio y análisis de punto de equilibrio.
- **Análisis de Riesgos Financieros:**
 - Identificar y evaluar los posibles riesgos financieros y sus impactos en el proyecto.

4. Análisis de Viabilidad Operativa

- **Evaluación de la Capacidad Operativa:**
 - Evaluar si la organización tiene la capacidad operativa para implementar y mantener el sistema.
- **Impacto en la Organización:**
 - Analizar cómo el proyecto afectará las operaciones actuales de la organización.
 - Evaluar la aceptación y resistencia al cambio por parte del personal.
- **Requisitos de Soporte:**
 - Identificar los requisitos de soporte y mantenimiento una vez que el sistema esté en funcionamiento.

5. Análisis de Viabilidad Legal y Normativa

- **Cumplimiento Legal:**
 - Evaluar el cumplimiento del proyecto con las leyes y regulaciones aplicables.
- **Consideraciones Contractuales:**
 - Identificar posibles problemas contractuales con proveedores y socios.
- **Aspectos de Seguridad y Privacidad:**
 - Evaluar cómo el proyecto abordará la seguridad de los datos y la privacidad.

6. Análisis de Viabilidad de Programación

- **Planificación del Cronograma:**
 - Desarrollar un cronograma preliminar del proyecto.
 - Identificar hitos y fechas clave.
- **Disponibilidad de Recursos:**
 - Evaluar la disponibilidad de recursos (humanos, tecnológicos y financieros) durante todo el ciclo de vida del proyecto.
- **Análisis de Riesgos de Programación:**
 - Identificar riesgos relacionados con el tiempo y desarrollar planes de contingencia.

7. Recomendaciones y Conclusiones

- **Resumen de Resultados:**

- Resumir los hallazgos de cada análisis de viabilidad.
- **Recomendaciones:**
 - Hacer recomendaciones sobre la viabilidad del proyecto basadas en el análisis realizado.
 - Sugerir posibles mejoras o ajustes para aumentar la viabilidad del proyecto.
- **Decisión Final:**
 - Proporcionar una conclusión sobre si el proyecto debe proceder, necesitar más estudios o ser descartado.

8. Presentación del Informe de Viabilidad

- **Informe de Viabilidad:**
 - Documentar todos los análisis y conclusiones en un informe de viabilidad detallado.
- **Presentación a los Stakeholders:**
 - Presentar el informe a los stakeholders clave para obtener su feedback y aprobación.

Ejemplo de Estudio de Viabilidad

Supongamos que una empresa quiere desarrollar un nuevo sistema de gestión de inventarios. El estudio de viabilidad podría incluir:

1. **Viabilidad Técnica:**
 - Evaluar si la empresa tiene la tecnología adecuada (servidores, bases de datos, etc.).
 - Determinar si el personal técnico tiene las habilidades necesarias para desarrollar y mantener el sistema.
2. **Viabilidad Económica:**
 - Estimar el costo del desarrollo del sistema (\$100,000) y los beneficios proyectados (reducción de costos de inventario en un 10% anual).
 - Calcular el ROI y el tiempo de recuperación de la inversión.
3. **Viabilidad Operativa:**
 - Evaluar cómo el nuevo sistema afectará los procesos de la empresa.
 - Determinar si el personal está dispuesto a adoptar el nuevo sistema.
4. **Viabilidad Legal:**
 - Asegurarse de que el sistema cumple con las regulaciones de protección de datos.
 - Evaluar cualquier contrato existente con proveedores de software.
5. **Viabilidad de Programación:**
 - Desarrollar un cronograma preliminar y evaluar la disponibilidad del equipo de desarrollo.

El estudio de viabilidad es crucial para tomar decisiones informadas y asegurar que los recursos de la empresa se utilicen de manera efectiva para proyectos que realmente añadan valor a la organización.

5.3. Análisis Costo/Beneficio.

El análisis costo/beneficio (ACB) es una técnica utilizada para evaluar los costos totales de un proyecto frente a los beneficios totales que generará, con el objetivo de determinar si el proyecto es económicamente viable y rentable. Este análisis es crucial en la toma de decisiones de proyectos de sistemas de información (SI) porque proporciona una base cuantitativa para justificar la inversión.

Pasos para Realizar un Análisis Costo/Beneficio

Identificación de Costos y Beneficios

Costos

- **Costos de Desarrollo:**
 - **Costos de Personal:** Salarios del equipo de desarrollo, consultores y personal de soporte.
 - **Costos de Hardware y Software:** Compra o alquiler de servidores, licencias de software, herramientas de desarrollo.
 - **Costos de Infraestructura:** Espacio físico, electricidad, conexiones de red.
 - **Costos de Capacitación:** Entrenamiento del personal en nuevas tecnologías o herramientas.
- **Costos de Implementación:**
 - **Costos de Integración:** Integración del nuevo sistema con los sistemas existentes.
 - **Costos de Migración:** Migración de datos desde sistemas antiguos al nuevo sistema.
 - **Costos de Pruebas:** Realización de pruebas de calidad, seguridad y rendimiento.
- **Costos de Operación y Mantenimiento:**
 - **Costos de Soporte Técnico:** Soporte continuo para usuarios y mantenimiento del sistema.
 - **Costos de Actualización:** Actualizaciones de hardware y software.
 - **Costos de Infraestructura Continuos:** Gastos continuos de operación como electricidad y espacio físico.

Beneficios

- **Beneficios Directos:**
 - **Incremento en Ingresos:** Aumento de ventas, acceso a nuevos mercados.
 - **Reducción de Costos:** Disminución de costos operativos, reducción de errores y retrabajos.

- **Beneficios Indirectos:**
 - **Mejora en la Productividad:** Mayor eficiencia y eficacia del personal.
 - **Mejora en la Toma de Decisiones:** Acceso a información más precisa y oportuna.
- **Beneficios Intangibles:**
 - **Mejora en la Satisfacción del Cliente:** Mejor experiencia del usuario, mayor fidelidad.
 - **Ventaja Competitiva:** Diferenciación frente a la competencia.

Cuantificación de Costos y Beneficios

- **Monetización:** Convertir todos los costos y beneficios en valores monetarios.
- **Horizonte Temporal:** Determinar el período de tiempo sobre el cual se evaluarán los costos y beneficios.

Cálculo del Valor Actual Neto (VAN) y la Tasa Interna de Retorno (TIR)

- **Valor Actual Neto (VAN):**
 - Descontar los costos y beneficios futuros al valor presente usando una tasa de descuento apropiada.
 - Fórmula del VAN:

$$VAN = \sum \left(\frac{\text{Beneficios}_t - \text{Costos}_t}{(1 + r)^t} \right) - \text{Inversión Inicial}$$

$$VAN = \sum ((1+r)^t \text{Beneficios}_t - \text{Costos}_t) - \text{Inversión Inicial}$$
Donde:
 - Beneficios_t = Beneficios en el año t
 - Costos_t = Costos en el año t
 - r = Tasa de descuento
 - t = Año del flujo de efectivo
- **Tasa Interna de Retorno (TIR):**
 - La tasa de descuento que hace que el VAN sea igual a cero.
 - Es una medida de la rentabilidad del proyecto.

Análisis de Sensibilidad

- Evaluar cómo cambian los resultados del ACB ante cambios en las principales variables (costos, beneficios, tasa de descuento).

Presentación de Resultados

- Resumir los hallazgos en un informe claro y conciso.
- Incluir recomendaciones basadas en los resultados del análisis.

Ejemplo de Análisis Costo/Beneficio

Supongamos que una empresa está considerando implementar un nuevo sistema de gestión de inventarios. Se han identificado los siguientes costos y beneficios:

Costos:

- Desarrollo: \$50,000
- Hardware y Software: \$20,000
- Capacitación: \$10,000
- Mantenimiento anual: \$5,000

Beneficios:

- Reducción de costos operativos: \$15,000/año
- Aumento en la precisión del inventario que resulta en menores pérdidas: \$10,000/año
- Mejor toma de decisiones: valor intangible estimado en \$5,000/año

Horizonte Temporal: 5 años **Tasa de Descuento:** 10%

Cálculo del VAN

$$VAN = \sum_{t=1}^n \left(\frac{\text{Beneficios}_t - \text{Costos}_t}{(1 + 0.10)^t} \right) - \text{Inversión Inicial}$$
$$VAN = \sum_{t=1}^n ((1 + 0.10)^t \text{Beneficios}_t - \text{Costos}_t) - \text{Inversión Inicial}$$

Año 1:

$$\frac{(15,000 + 10,000 + 5,000) - 5,000}{(1 + 0.10)^1} = \frac{25,000 - 5,000}{1.10} = \frac{20,000}{1.10} = 18,182$$
$$\frac{25,000 - 5,000}{1.10} = 18,182$$

Año 2:

$$\frac{25,000 - 5,000}{(1 + 0.10)^2} = \frac{20,000}{1.21} = 16,529$$
$$\frac{25,000 - 5,000}{1.21} = 16,529$$

Año 3:

$$\frac{20,000}{(1 + 0.10)^3} = \frac{20,000}{1.331} = 15,028$$
$$\frac{20,000}{1.331} = 15,028$$

Año 4:

$$\frac{20,000}{(1 + 0.10)^4} = \frac{20,000}{1.4641} = 13,660$$
$$\frac{20,000}{1.4641} = 13,660$$

Año 5:

$$20,000(1+0.10)^5=20,000(1.61051)=12,418 \frac{20,000}{1.61051} = 12,418$$

Sumamos estos valores y restamos la inversión inicial:

$$VAN=18,182+16,529+15,028+13,660+12,418-80,000=-4,183$$

$$VAN = 18,182 + 16,529 + 15,028 + 13,660 + 12,418 - 80,000 = -4,183$$

En este ejemplo, el VAN es negativo, lo que sugiere que el proyecto no es económicamente viable con los supuestos dados.

Tasa Interna de Retorno (TIR)

La TIR es la tasa que hace que el VAN sea cero. En este caso, debemos encontrar la tasa rrr tal que:

$$0 = \sum (20,000(1+r)^{-t}) - 80,000 = \sum \left(\frac{20,000}{(1+r)^t} \right) - 80,000$$

Resolver esta ecuación requiere un proceso iterativo o el uso de software financiero.

Análisis de Sensibilidad

Podríamos evaluar cómo cambios en la tasa de descuento, costos o beneficios impactan el VAN y la TIR para entender mejor las condiciones bajo las cuales el proyecto podría ser viable.

Realizar un análisis costo/beneficio detallado y riguroso ayuda a las organizaciones a tomar decisiones informadas sobre la viabilidad económica de sus proyectos de sistemas de información.

5.4. Gestión de riesgos.

La gestión de riesgos en proyectos de sistemas de información (SI) es un proceso crítico que implica identificar, evaluar y mitigar los riesgos que podrían afectar el éxito del proyecto. A continuación, se describen las etapas clave y las actividades involucradas en la gestión de riesgos:

1. Planificación de la Gestión de Riesgos

- **Definición del Plan de Gestión de Riesgos:**
 - Establecer el enfoque, las actividades y los criterios para gestionar los riesgos del proyecto.
 - Definir roles y responsabilidades para la gestión de riesgos.
- **Identificación de Stakeholders:**

- Involucrar a los stakeholders clave para identificar y evaluar riesgos potenciales.

2. Identificación de Riesgos

- **Revisión de Documentación del Proyecto:**
 - Revisar los documentos del proyecto, como el acta de constitución, el plan de proyecto y los requisitos, para identificar posibles riesgos.
- **Técnicas de Identificación:**
 - **Lluvia de Ideas:** Reuniones del equipo para generar una lista de posibles riesgos.
 - **Entrevistas:** Conversaciones con expertos y stakeholders para identificar riesgos.
 - **Análisis FODA:** Identificación de fortalezas, oportunidades, debilidades y amenazas.
 - **Listas de Verificación:** Uso de listas predefinidas de riesgos comunes.
- **Registro de Riesgos:**
 - Documentar todos los riesgos identificados en un registro de riesgos, incluyendo descripciones, causas y posibles consecuencias.

3. Análisis Cualitativo de Riesgos

- **Evaluación de Probabilidad e Impacto:**
 - Asignar una probabilidad de ocurrencia y un impacto potencial a cada riesgo.
 - Clasificar los riesgos en categorías (alto, medio, bajo) según su probabilidad e impacto.
- **Matriz de Riesgos:**
 - Crear una matriz de riesgos para visualizar la severidad de cada riesgo en función de su probabilidad e impacto.
- **Priorización de Riesgos:**
 - Identificar los riesgos más críticos que requieren atención inmediata.

4. Análisis Cuantitativo de Riesgos (Opcional)

- **Modelos de Simulación:**
 - Usar técnicas como la simulación de Monte Carlo para cuantificar el impacto potencial de los riesgos en el cronograma y el presupuesto.
- **Árboles de Decisión:**
 - Utilizar árboles de decisión para evaluar diferentes escenarios y sus implicaciones financieras.
- **Valor Monetario Esperado (VME):**
 - Calcular el VME para cada riesgo, multiplicando su probabilidad por el impacto monetario estimado.

5. Planificación de la Respuesta a los Riesgos

- **Estrategias de Respuesta a Riesgos:**
 - **Evitar:** Modificar el plan del proyecto para eliminar el riesgo.
 - **Mitigar:** Tomar acciones para reducir la probabilidad o el impacto del riesgo.
 - **Transferir:** Transferir el riesgo a un tercero (por ejemplo, mediante seguros o contratos).
 - **Aceptar:** Aceptar el riesgo y desarrollar un plan de contingencia.
- **Planes de Contingencia:**
 - Desarrollar planes de acción detallados para implementar en caso de que se materialicen los riesgos.
- **Asignación de Responsabilidades:**
 - Asignar un responsable para cada riesgo y su respectiva respuesta.

6. Monitoreo y Control de Riesgos

- **Seguimiento de Riesgos:**
 - Monitorear continuamente los riesgos y las acciones de respuesta.
 - Revisar y actualizar el registro de riesgos regularmente.
- **Informes de Riesgos:**
 - Preparar informes periódicos sobre el estado de los riesgos y su gestión para los stakeholders.
- **Revisión de Efectividad:**
 - Evaluar la efectividad de las respuestas a los riesgos y ajustar las estrategias según sea necesario.

7. Comunicación de Riesgos

- **Comunicación Transparente:**
 - Mantener una comunicación clara y abierta sobre los riesgos y su gestión con todo el equipo del proyecto y los stakeholders.
- **Reuniones de Riesgos:**
 - Realizar reuniones periódicas de revisión de riesgos para discutir el estado y las acciones de mitigación.

Ejemplo de Gestión de Riesgos

1. **Identificación de Riesgos:**
 - Riesgo 1: Falta de habilidades técnicas en el equipo.
 - Riesgo 2: Retrasos en la entrega del hardware necesario.
 - Riesgo 3: Resistencia al cambio por parte del personal.
 - Riesgo 4: Problemas de integración con el sistema existente.
2. **Análisis Cualitativo:**
 - Riesgo 1: Probabilidad alta, impacto alto.
 - Riesgo 2: Probabilidad media, impacto alto.
 - Riesgo 3: Probabilidad alta, impacto medio.

- Riesgo 4: Probabilidad media, impacto medio.
- 3. Planificación de Respuestas:**
 - Riesgo 1: Mitigar. Proveer formación adicional y contratar consultores externos.
 - Riesgo 2: Mitigar. Establecer contratos con penalizaciones por retrasos y buscar proveedores alternativos.
 - Riesgo 3: Mitigar. Realizar sesiones de sensibilización y formación para el personal.
 - Riesgo 4: Mitigar. Realizar pruebas de integración tempranas y planificar tiempo adicional para la integración.
- 4. Monitoreo y Control:**
 - Revisar semanalmente el progreso de las respuestas a los riesgos.
 - Actualizar el registro de riesgos con cualquier nuevo riesgo identificado y el estado de los riesgos existentes.
- 5. Comunicación:**
 - Informar mensualmente a la alta dirección sobre el estado de los riesgos y las acciones de mitigación.
 - Realizar reuniones quincenales con el equipo del proyecto para discutir los riesgos y las estrategias de respuesta.

Registro de Riesgos (Ejemplo)

ID	Descripción	Probabilidad	Impacto	Estrategia de Respuesta	Responsable	Estado
R1	Falta de habilidades técnicas	Alta	Alto	Mitigar	Gerente TI	Activo
R2	Retrasos en entrega de hardware	Media	Alto	Mitigar	Proveedor	Activo
R3	Resistencia al cambio	Alta	Medio	Mitigar	RRHH	Activo
R4	Problemas de integración	Media	Medio	Mitigar	Líder Técnico	Activo

Implementar una gestión de riesgos efectiva permite a las organizaciones de SI anticipar problemas, minimizar su impacto y aumentar las probabilidades de éxito del proyecto.

5.5. Planificación del proyecto.

La planificación del proyecto es una fase crucial en la gestión de proyectos de sistemas de información (SI) que implica definir claramente los objetivos, establecer un cronograma detallado, asignar recursos y delinear las actividades necesarias para completar el proyecto. A continuación, se detallan los pasos y componentes esenciales para una planificación efectiva del proyecto.

1. Definición del Proyecto

Acta de Constitución del Proyecto

- **Objetivos del Proyecto:**
 - Describir los objetivos específicos y medibles del proyecto.
- **Alcance del Proyecto:**
 - Definir claramente el alcance del proyecto, incluyendo lo que está dentro y fuera del alcance.
- **Stakeholders:**
 - Identificar a todas las partes interesadas y sus roles.

Requisitos del Proyecto

- **Requisitos Funcionales:**
 - Detallar lo que el sistema debe hacer.
- **Requisitos No Funcionales:**
 - Especificar las cualidades del sistema, como rendimiento, seguridad y usabilidad.

2. Estructura de Desglose del Trabajo (EDT)

Descomposición del Proyecto

- Dividir el proyecto en tareas más pequeñas y manejables.
- Cada tarea debe ser clara y específica.

Jerarquía de Tareas

- Organizar las tareas en una estructura jerárquica.
- Crear paquetes de trabajo que se puedan asignar y gestionar.

3. Cronograma del Proyecto

Identificación de Tareas

- Listar todas las tareas y actividades necesarias para completar el proyecto.

Dependencias de Tareas

- Determinar las relaciones y dependencias entre las tareas.
- Utilizar diagramas de red (como el Diagrama de Pert) para visualizar las dependencias.

Duración de las Tareas

- Estimar la duración de cada tarea.
- Utilizar técnicas como el método Delphi o la técnica de tres estimaciones (optimista, pesimista y más probable).

Diagrama de Gantt

- Crear un diagrama de Gantt para visualizar el cronograma del proyecto.
- Asignar fechas de inicio y fin para cada tarea.

4. Asignación de Recursos

Identificación de Recursos

- Listar todos los recursos necesarios, incluidos personal, equipo, software y otros materiales.

Asignación de Tareas a Recursos

- Asignar cada tarea a los recursos apropiados.
- Asegurar que los recursos estén disponibles cuando se necesiten.

5. Presupuesto del Proyecto

Estimación de Costos

- Estimar los costos asociados con cada tarea y recurso.
- Considerar costos directos (materiales, mano de obra) e indirectos (infraestructura, servicios).

Elaboración del Presupuesto

- Crear un presupuesto detallado que incluya todos los costos estimados.
- Asegurar que el presupuesto sea realista y alcanzable.

6. Gestión de Riesgos

Identificación de Riesgos

- Identificar posibles riesgos que podrían afectar el proyecto.

Evaluación de Riesgos

- Evaluar la probabilidad e impacto de cada riesgo.
- Priorizar los riesgos en función de su severidad.

Plan de Respuesta a Riesgos

- Desarrollar estrategias para mitigar, transferir, evitar o aceptar los riesgos.
- Crear planes de contingencia para riesgos críticos.

7. Plan de Calidad

Criterios de Calidad

- Definir los estándares de calidad que debe cumplir el proyecto.

Plan de Aseguramiento de Calidad

- Desarrollar procesos para asegurar que el proyecto cumple con los estándares de calidad.

Plan de Control de Calidad

- Establecer procedimientos para monitorear y controlar la calidad durante el proyecto.

8. Comunicación del Proyecto

Plan de Comunicación

- Definir cómo y cuándo se comunicarán los avances del proyecto a los stakeholders.
- Establecer canales y frecuencias de comunicación.

Informes de Progreso

- Crear informes regulares para mantener a los stakeholders informados sobre el progreso del proyecto.

9. Gestión del Cambio

Plan de Gestión del Cambio

- Desarrollar un plan para gestionar los cambios en el proyecto.
- Definir el proceso para solicitar, evaluar y aprobar cambios.

Control de Cambios

- Implementar un sistema para registrar y rastrear todos los cambios aprobados.

10. Revisión y Aprobación del Plan

Revisión del Plan

- Revisar todo el plan del proyecto para asegurarse de que sea completo y coherente.

Aprobación del Plan

- Obtener la aprobación del plan por parte de los stakeholders clave.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A5.5.1

5.6. Calendarización del proyecto

La calendarización de proyectos de software tiene características específicas debido a la naturaleza de desarrollo de software, que incluye fases de planificación, diseño, desarrollo, pruebas, y despliegue. A continuación, se presenta una guía detallada para la calendarización de un proyecto de software:

1. Definición del Proyecto

- **Objetivos:** Claros y específicos, como desarrollar una aplicación móvil, un sistema de gestión, etc.
- **Alcance:** Define los módulos, funcionalidades y limitaciones del proyecto.
- **Entregables:** Listar los productos o resultados específicos que se entregarán.

2. Fases del Proyecto

- **Fase de Planificación**
- **Fase de Diseño**
- **Fase de Desarrollo**
- **Fase de Pruebas**
- **Fase de Despliegue**
- **Fase de Mantenimiento**

3. Desglose del Trabajo

- **Work Breakdown Structure (WBS):** Divide el proyecto en tareas más pequeñas y manejables.

4. Estimación de Tiempos y Recursos

- **Duración de Tareas:** Estima el tiempo necesario para completar cada tarea.
- **Recursos:** Asigna desarrolladores, diseñadores, testers, etc., a cada tarea.

5. Identificación de Dependencias

- **Dependencias entre Tareas:** Identifica qué tareas dependen de la finalización de otras tareas.

6. Creación del Cronograma

- **Herramientas de Gestión:** Utiliza herramientas como Jira, Trello, Asana, Microsoft Project.
- **Diagrama de Gantt:** Visualiza la programación de tareas y sus dependencias.

7. Implementación del Cronograma

- **Asignación de Responsabilidades:** Asegura que todos los miembros del equipo conozcan sus tareas y plazos.
- **Calendario Detallado:** Define fechas de inicio y fin para cada tarea y fase del proyecto.

8. Seguimiento y Control

- **Reuniones de Seguimiento:** Programa reuniones regulares para revisar el progreso.
- **Actualización del Cronograma:** Ajusta el cronograma según sea necesario.
- **Gestión de Riesgos:** Monitorea y gestiona riesgos que puedan afectar el cronograma.

Ver ANEXO 1. Unidad I Actividad de aprendizaje A5.6.1.

Referencias

1. Project Management Institute (PMI) (2021). A Guide to the Project Management Body of Knowledge (PMBOK® Guide). Este libro es una referencia estándar en la gestión de proyectos, proporcionando un marco de trabajo y mejores prácticas ampliamente reconocidas.
2. Schwalbe, K. (2018). Information Technology Project Management. Ofrece una visión detallada de la gestión de proyectos de TI, incluyendo estudios de caso y aplicaciones prácticas.
3. Kerzner, H. (2017). Project Management: A Systems Approach to Planning, Scheduling, and Controlling. Proporciona una cobertura completa de los principios y técnicas de gestión de proyectos.
4. Wysocki, R. K. (2019). Effective Project Management: Traditional, Agile, Extreme. Explora diferentes enfoques de gestión de proyectos y cómo aplicarlos en distintos contextos.
5. Gido, J., & Clements, J. P. (2020). Successful Project Management. Una guía práctica que cubre todas las etapas de la gestión de proyectos, desde la iniciación hasta el cierre.
6. Meredith, J. R., & Mantel, S. J. (2014). Project Management: A Managerial Approach. Enfoca la gestión de proyectos desde una perspectiva gerencial, proporcionando estrategias para liderar y coordinar proyectos complejos.

Estos libros y recursos proporcionan una comprensión profunda de los principios y prácticas de la gestión de proyectos de sistemas de información y son esenciales tanto para estudiantes como para profesionales en el campo.

Bibliografía:

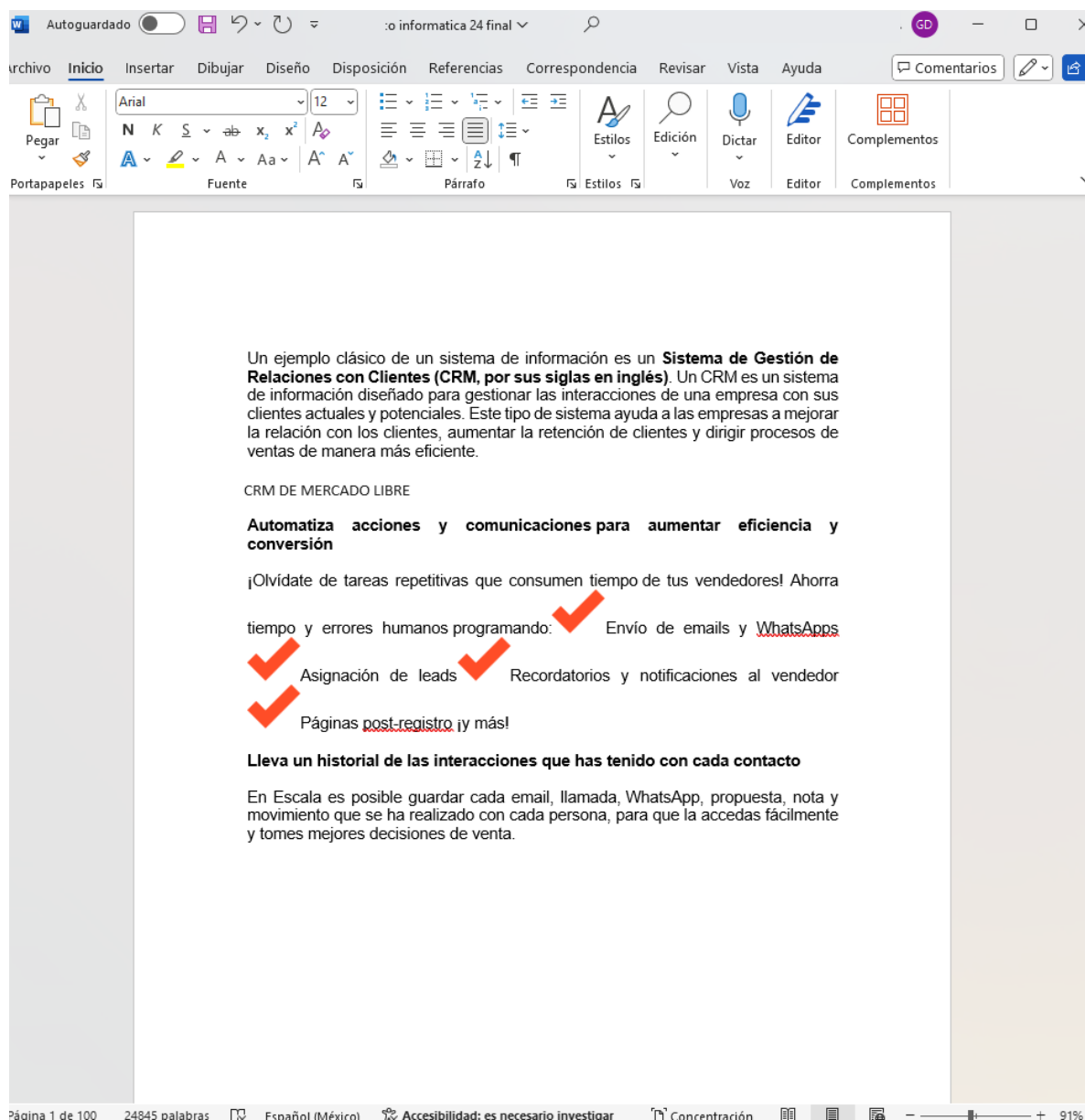
1. Laudon, K.C., & Laudon, J.P. (2020). Management Information Systems: Managing the Digital Firm. Este libro proporciona una visión comprensiva de cómo los sistemas de información se utilizan para apoyar las operaciones y la toma de decisiones en las organizaciones.
2. Stair, R., & Reynolds, G. (2019). Principles of Information Systems. Una introducción a los conceptos fundamentales de los sistemas de información, incluyendo su diseño, desarrollo y gestión.
3. Turban, E., Sharda, R., & Delen, D. (2020). Decision Support and Business Intelligence Systems. Este texto profundiza en cómo los sistemas de soporte de decisiones y las tecnologías de inteligencia empresarial se utilizan para mejorar la toma de decisiones empresariales.
4. O'Brien, J.A., & Marakas, G.M. (2011). Introduction to Information Systems. Proporciona una visión general de los sistemas de información y cómo se utilizan en el entorno empresarial actual.
5. Beynon-Davies, P. (2009). Business Information Systems. Este libro cubre los aspectos técnicos y organizacionales de los sistemas de información en el contexto empresarial.
6. Cohen y Asin.; Sistemas de Información un enfoque de toma de decisiones. 3ª Edición. Mc Graw Hill.2000.
7. EDWARDS, CHRIS; JOHN WARD y ANDY BYTHEWAY. Fundamentos de Sistemas de Información. 2da. Edición. Ed. Prentice Hall. 1998.
8. KENDALL, KENNETH E. Y KENDALL, JULIE E. Análisis y Diseño de Sistemas. 6ª Edición; Ed. Pearson Educación México. 2005.
9. PRESSMAN, ROGER S.; Ingeniería de software un Enfoque práctico; Ed. Mc. Graw. Hill. 2007.
10. SOMMERVILLE, IAN; Ingeniería de Software, Edit. Addison Wesley; 2005.

ANEXO 1. Actividades de Aprendizaje

Unidad I

A1.1.1 Estudio de caso: Sistema de Gestión de Relaciones con Clientes

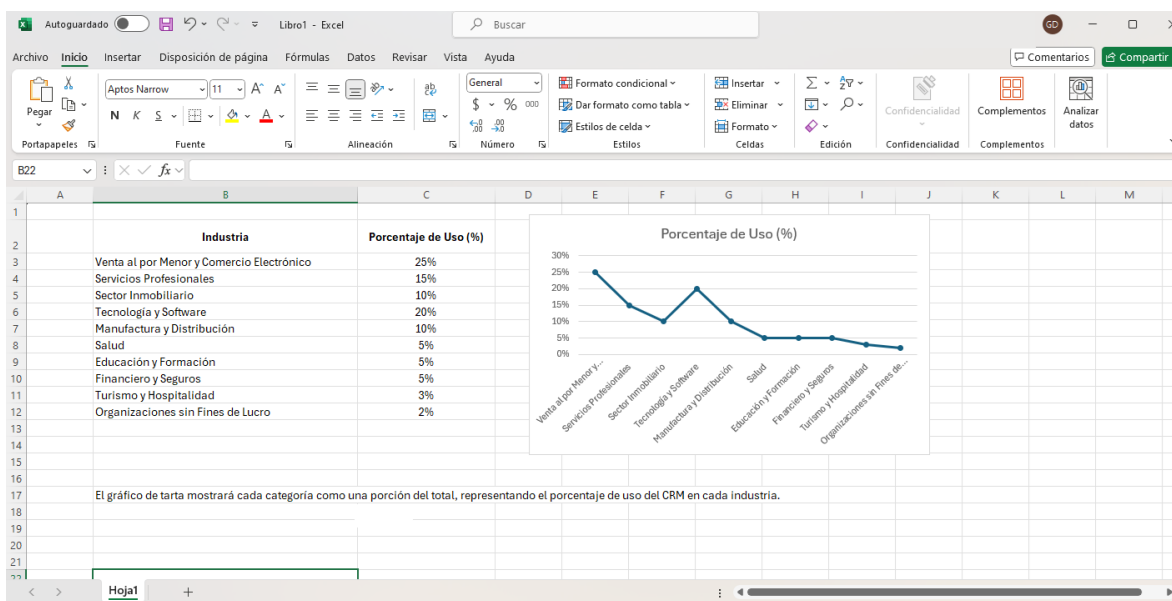
En Word realizar un resumen de un caso de estudio basado en el sistema de gestión de clientes de mercado libre.



A1.2.1 Ejercicio propuesto: Grafica en Excel de empresas que utilizan CRM.

Se requiere una gráfica que muestre las diferentes industrias de acuerdo al porcentaje de uso del CRM en cada tipo de empresa. La grafica debe apoyarse de la siguiente tabla:

Industria	Porcentaje de Uso (%)
Venta al por Menor y Comercio Electrónico	25%
Servicios Profesionales	15%
Sector Inmobiliario	10%
Tecnología y Software	20%
Manufactura y Distribución	10%
Salud	5%
Educación y Formación	5%
Financiero y Seguros	5%
Turismo y Hospitalidad	3%
Organizaciones sin Fines de Lucro	2%



A1.3.1 Aprendizaje basado en problemas: Uso de un sistema CRM

Realizar una presentación en power point en donde se represente una empresa llamada "TechSolutions" que vende software y servicios de TI. TechSolutions implementa un sistema CRM para mejorar su interacción con los clientes y gestionar sus procesos de ventas y marketing de manera más eficiente.

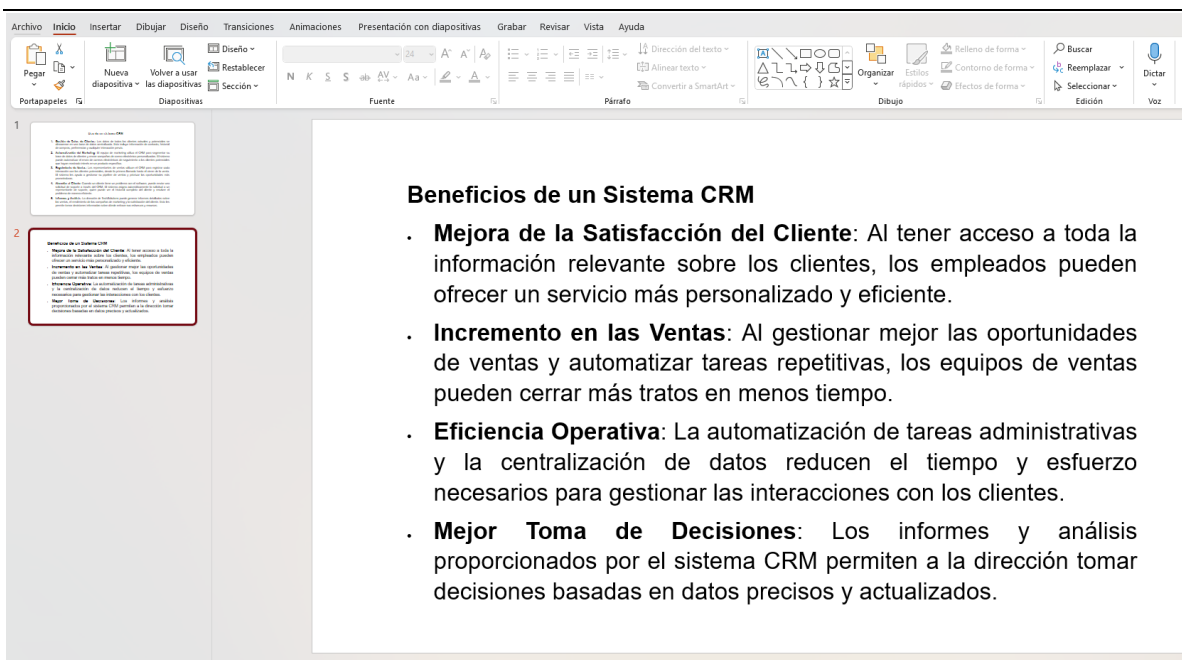
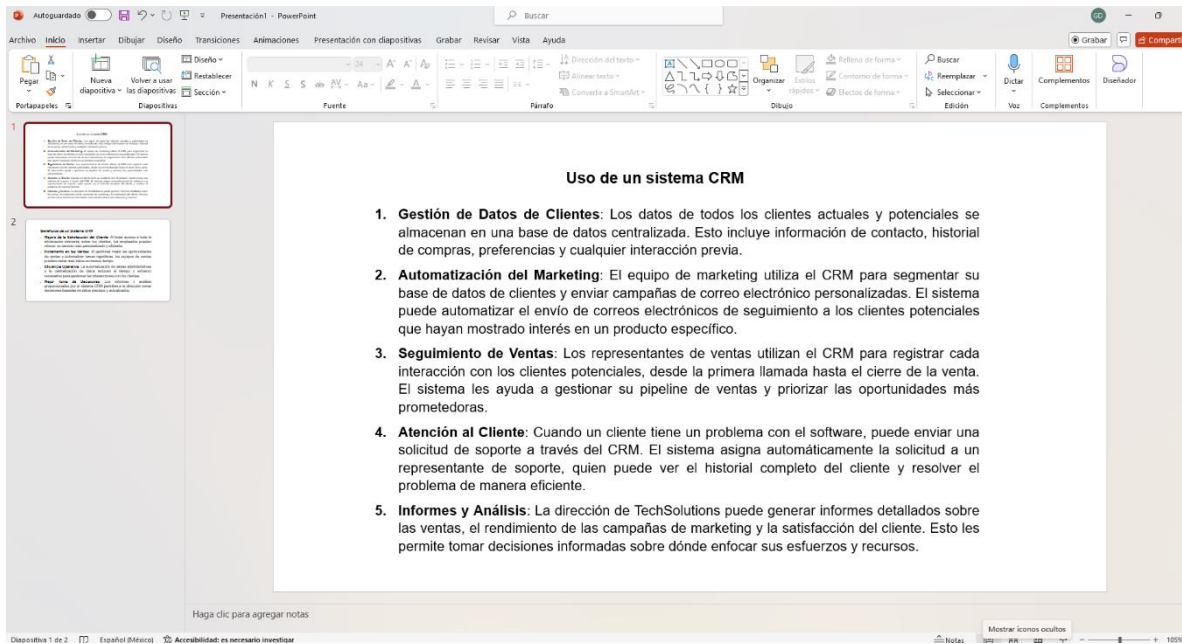
Solución propuesta:

1. **Gestión de Datos de Clientes:** Los datos de todos los clientes actuales y potenciales se almacenan en una base de datos centralizada. Esto incluye información de contacto, historial de compras, preferencias y cualquier interacción previa.
2. **Automatización del Marketing:** El equipo de marketing utiliza el CRM para segmentar su base de datos de clientes y enviar campañas de correo electrónico personalizadas. El sistema puede automatizar el envío de correos electrónicos de seguimiento a los clientes potenciales que hayan mostrado interés en un producto específico.
3. **Seguimiento de Ventas:** Los representantes de ventas utilizan el CRM para registrar cada interacción con los clientes potenciales, desde la primera llamada hasta el cierre de la venta. El sistema les ayuda a gestionar su pipeline de ventas y priorizar las oportunidades más prometedoras.
4. **Atención al Cliente:** Cuando un cliente tiene un problema con el software, puede enviar una solicitud de soporte a través del CRM. El sistema asigna automáticamente la solicitud a un representante de soporte, quien puede ver el historial completo del cliente y resolver el problema de manera eficiente.
5. **Informes y Análisis:** La dirección de TechSolutions puede generar informes detallados sobre las ventas, el rendimiento de las campañas de marketing y la satisfacción del cliente. Esto les permite tomar decisiones informadas sobre dónde enfocar sus esfuerzos y recursos.

Beneficios de un Sistema CRM

- **Mejora de la Satisfacción del Cliente:** Al tener acceso a toda la información relevante sobre los clientes, los empleados pueden ofrecer un servicio más personalizado y eficiente.
- **Incremento en las Ventas:** Al gestionar mejor las oportunidades de ventas y automatizar tareas repetitivas, los equipos de ventas pueden cerrar más tratos en menos tiempo.
- **Eficiencia Operativa:** La automatización de tareas administrativas y la centralización de datos reducen el tiempo y esfuerzo necesarios para gestionar las interacciones con los clientes.

- **Mejor Toma de Decisiones:** Los informes y análisis proporcionados por el sistema CRM permiten a la dirección tomar decisiones basadas en datos precisos y actualizados.



UNIDAD II

A2.3.1.1 Estudio de caso: Realizar un resumen de un caso real basado en el desarrollo de un sistema informático para el control del servicio técnico de la empresa Telmex.

Solución propuesta:

Autoguardado | Buscar

Archivo Inicio Insertar Dibujar Diseño Disposición Referencias Correspondencia Revisar Vista Ayuda

Portapapeles Fuente Párrafo Estilos

Estudio de caso

Desarrollo de un sistema informático para el control del servicio técnico de la empresa Telmex.

1. Seguimiento de Incidencias:

- La empresa enfrentaba dificultades para rastrear y gestionar las incidencias de servicio, lo que resultaba en retrasos y falta de visibilidad en el estado de cada solicitud.

2. Integración de Datos:

- Los datos relacionados con el servicio técnico estaban dispersos en varios sistemas y formatos, dificultando la obtención de una visión consolidada del estado de las solicitudes y el rendimiento del equipo.

3. Eficiencia Operativa:

- El proceso manual de asignación de tareas y seguimiento del personal técnico resultaba ineficiente y propenso a errores.

4. Comunicación con el Cliente:

- La falta de un sistema centralizado impedía una comunicación efectiva con los clientes y la actualización oportuna sobre el estado de sus solicitudes.

Solución Implementada:

Telmex desarrolló e implementó un sistema informático integral para la gestión del servicio técnico con los siguientes componentes clave:

1. Desarrollo del Sistema:

- Análisis de Requisitos:**

Se realizó un análisis detallado para identificar las necesidades y requisitos del sistema, incluyendo la gestión de incidencias, y comunicación con el cliente.

Diseño y Arquitectura:

- El sistema fue diseñado para incluir un panel de control de incidencias, asignación automática de tareas y herramientas de comunicación.

2. Características del Sistema:

- Gestión de Incidencias:**
 - Permite registrar, clasificar y desde la apertura hasta la resolución, optimizando así la eficiencia.
- Asignación de Tareas:**
 - Automatiza la asignación de tareas basado en la ubicación, disponiendo optimizando así la eficiencia.
- Base de Datos Centralizada:**
 - Integra datos de diversas fuentes centralizada, proporcionando una visión unificada de todas las solicitudes de servicio técnico.
- Comunicación con el Cliente:**
 - Incluye herramientas para actualizar sobre el estado de sus solicitudes de comunicación bidireccional efectiva.

3. Implementación:

- Fases del Despliegue:**

Página 1 de 103 | 25329 palabras | Español (México) | Accesibilidad: todo correcto

Telmex, una de las principales empresas de telecomunicaciones en América Latina, enfrentaba desafíos significativos en la gestión de su servicio técnico. La empresa proporcionaba soporte para una amplia gama de servicios de telecomunicaciones, incluidos telefonía fija, móvil, Internet y TV por cable. La gestión eficiente del servicio técnico era crucial para mantener la satisfacción del cliente y asegurar la continuidad del servicio.

1. Seguimiento de Incidencias:

- La empresa enfrentaba dificultades para rastrear y gestionar las incidencias de servicio, lo que resultaba en retrasos y falta de visibilidad en el estado de cada solicitud.

2. Integración de Datos:

- Los datos relacionados con el servicio técnico estaban dispersos en varios sistemas y formatos, dificultando la obtención de una visión consolidada del estado de las solicitudes y el rendimiento del equipo.

3. Eficiencia Operativa:

- El proceso manual de asignación de tareas y seguimiento del personal técnico resultaba ineficiente y propenso a errores.

4. Comunicación con el Cliente:

- La falta de un sistema centralizado impedía una comunicación efectiva con los clientes y la actualización oportuna sobre el estado de sus solicitudes.

Solución Implementada:

Telmex desarrolló e implementó un sistema informático integral para la gestión del servicio técnico con los siguientes componentes clave:

1. Desarrollo del Sistema:

- **Análisis de Requisitos:**
 - Se realizó un análisis detallado para identificar las necesidades específicas del servicio técnico, incluyendo la gestión de incidencias, asignación de tareas, y comunicación con el cliente.
- **Diseño y Arquitectura:**

- El sistema fue diseñado para integrar varios módulos, incluyendo un panel de control centralizado, gestión de incidencias, asignación automática de tareas, y herramientas de comunicación con el cliente.

2. Características del Sistema:

○ Gestión de Incidencias:

- Permite registrar, clasificar y rastrear incidencias desde la apertura hasta la resolución. Incluye herramientas para priorización y seguimiento del estado.

○ Asignación de Tareas:

- Automatiza la asignación de tareas al personal técnico basado en la ubicación, disponibilidad y habilidades, optimizando así la eficiencia.

○ Base de Datos Centralizada:

- Integra datos de diversas fuentes en una base de datos centralizada, proporcionando una visión unificada de todas las solicitudes y el rendimiento del servicio técnico.

○ Comunicación con el Cliente:

- Incluye herramientas para actualizar a los clientes sobre el estado de sus solicitudes y permitir una comunicación bidireccional efectiva.

3. Implementación:

○ Fases del Despliegue:

- La implementación se realizó en fases para asegurar una transición suave. Inicialmente se desplegó en áreas piloto antes de una implementación a gran escala.

○ Capacitación del Personal:

- Se proporcionó capacitación extensiva al personal técnico y al equipo de soporte para garantizar un uso efectivo del nuevo sistema.

4. Mantenimiento y Soporte:

- **Actualizaciones y Mejoras:**

- Se estableció un plan de mantenimiento continuo para actualizar el sistema y abordar cualquier problema emergente.

- **Soporte Técnico:**

- Se implementó un equipo de soporte para resolver problemas técnicos y proporcionar asistencia adicional.

Resultados:

1. Mejora en la Gestión de Incidencias:

- El sistema permitió un seguimiento más eficiente de las incidencias, reduciendo los tiempos de respuesta y mejorando la resolución de problemas.

2. Eficiencia Operativa Aumentada:

- La automatización de la asignación de tareas y la centralización de datos mejoraron significativamente la eficiencia operativa y redujeron el margen de error.

3. Mejora en la Comunicación con el Cliente:

- La capacidad de proporcionar actualizaciones en tiempo real y mantener una comunicación fluida con los clientes mejoró la satisfacción y confianza del cliente.

4. Visibilidad y Análisis:

- La integración de datos permitió un análisis más profundo del rendimiento del servicio técnico, identificando áreas de mejora y oportunidades para optimizar los procesos.

Conclusión:

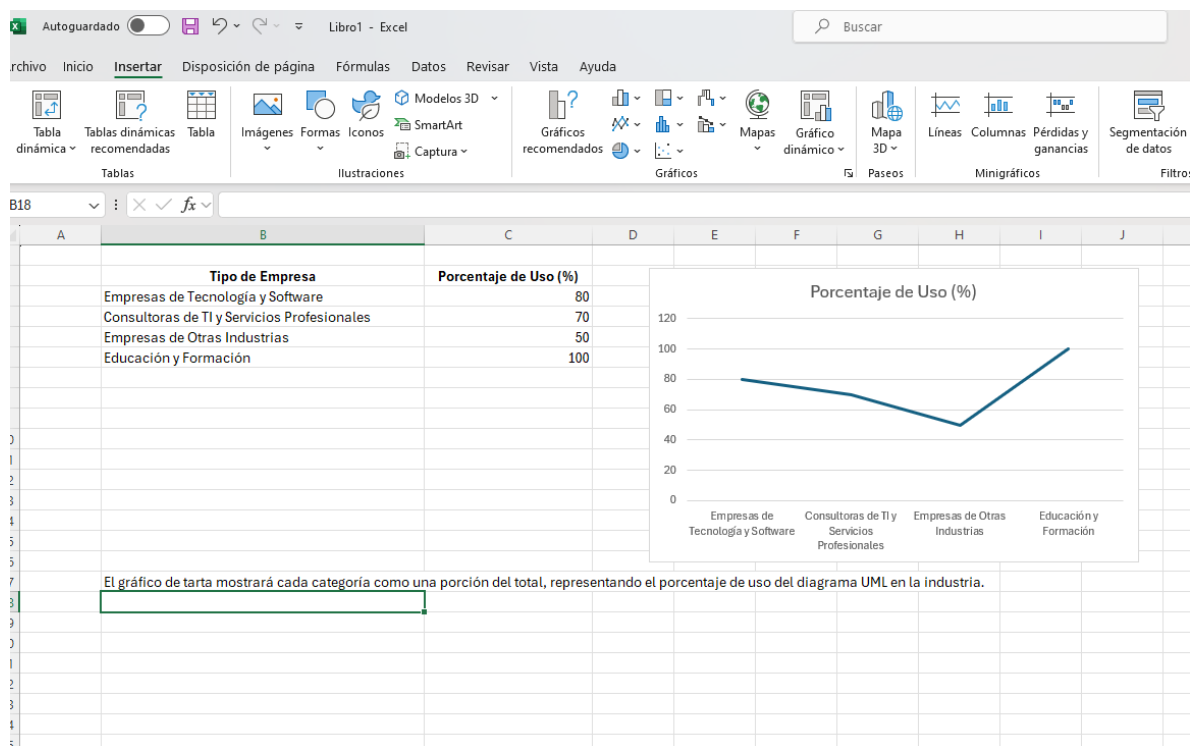
La implementación del sistema de control de servicio técnico en Telmex representó un avance significativo en la gestión de incidencias y la eficiencia operativa. La solución no solo mejoró la capacidad de la empresa para manejar solicitudes de servicio técnico de manera efectiva, sino que también fortaleció la comunicación con los clientes y proporcionó una visión integral de las operaciones. Este caso demuestra cómo una solución informática bien diseñada puede transformar la gestión del servicio técnico y contribuir a una mayor satisfacción del cliente.

A2.3.2.1 Ejercicio propuesto: Grafica en Excel presentar el porcentaje de uso por parte de las empresas que utilizan el diagrama UML en la actualidad.

Solución propuesta:

Datos:

Tipo de Empresa	Porcentaje de Uso (%)
Empresas de Tecnología y Software	70-80%
Consultoras de TI y Servicios Profesionales	60-70%
Empresas de Otras Industrias	20-50%
Educación y Formación	Alta (100%)



A2.3.2.2 Aprendizaje basado en problemas: Realizar un diagrama UML del sistema informático de Telmex en power point.

Solución propuesta:

1. Diagrama de Casos de Uso

El diagrama de casos de uso ilustra las principales funciones del sistema y cómo los actores interactúan con él. En el caso de Telmex, los actores principales podrían ser el **Cliente**, el **Técnico**, y el **Administrador**. Los casos de uso incluyen:

- **Cliente:**
 - **Reportar Incidencia**
 - **Consultar Estado de Incidencia**
 - **Recibir Actualizaciones**
- **Técnico:**
 - **Asignar Incidencias**
 - **Actualizar Estado de Incidencia**
 - **Registrar Resolución**
- **Administrador:**
 - **Gestionar Técnicos**
 - **Generar Reportes**
 - **Configurar Sistema**

2. Diagrama de Clases

El diagrama de clases representa las clases del sistema y sus relaciones. En el sistema de control de servicio técnico, podrías tener las siguientes clases:

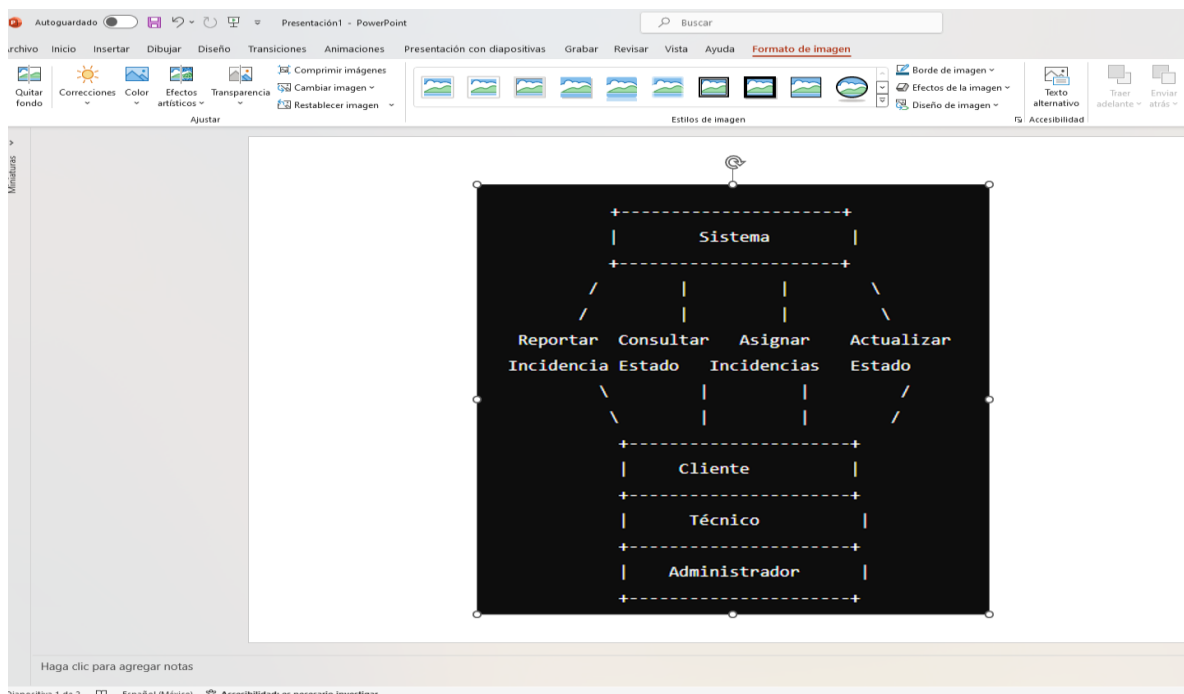
- **Incidencia**
 - Atributos: ID, Descripción, Prioridad, Estado
 - Métodos: Registrar(), Actualizar(), Resolver()
- **Técnico**
 - Atributos: ID, Nombre, Especialidad, Estado
 - Métodos: AsignarIncidencia(), ActualizarEstado()
- **Cliente**
 - Atributos: ID, Nombre, Contacto
 - Métodos: ReportarIncidencia(), ConsultarEstado()
- **Administrador**

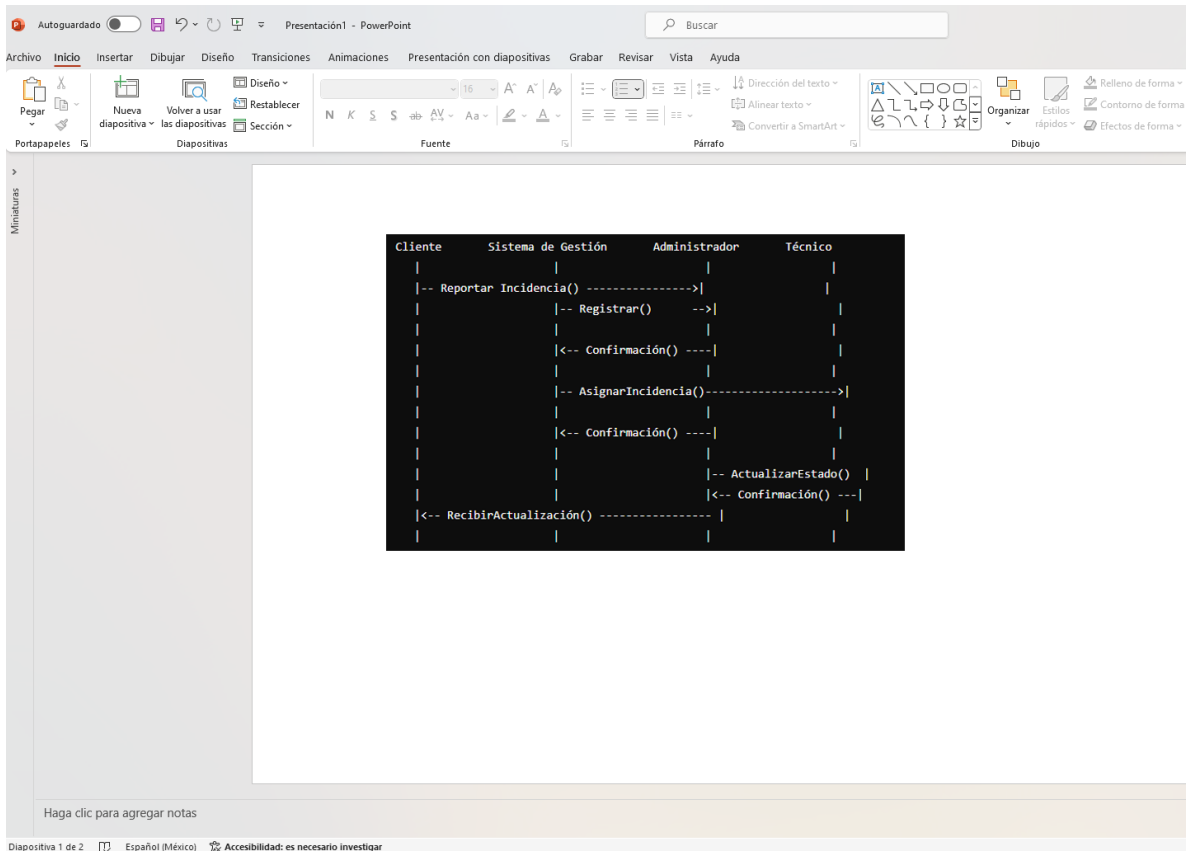
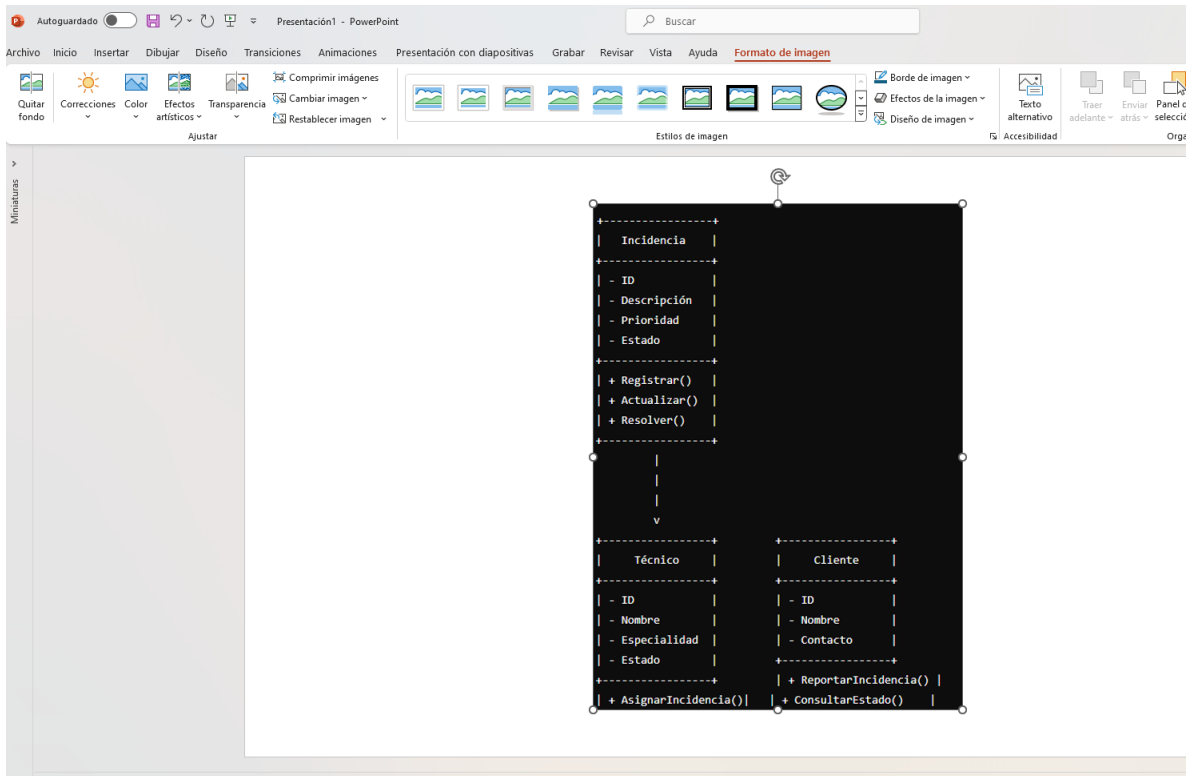
- Atributos: ID, Nombre
 - Métodos: GestionarTécnicos(), GenerarReportes()
- **Sistema de Gestión**
 - Métodos: GestionarIncidentes(), AsignarTareas(), EnviarActualizaciones()

3. Diagrama de Secuencia

El diagrama de secuencia muestra cómo los objetos interactúan en un proceso particular. Para el proceso de "**Reportar Incidencia**", el flujo sería:

1. **Cliente** inicia la solicitud.
2. **Sistema de Gestión** recibe y registra la incidencia.
3. **Administrador** revisa y asigna la incidencia a un **Técnico**.
4. **Técnico** actualiza el estado de la incidencia.
5. **Sistema de Gestión** envía actualizaciones al **Cliente**.





UNIDAD III

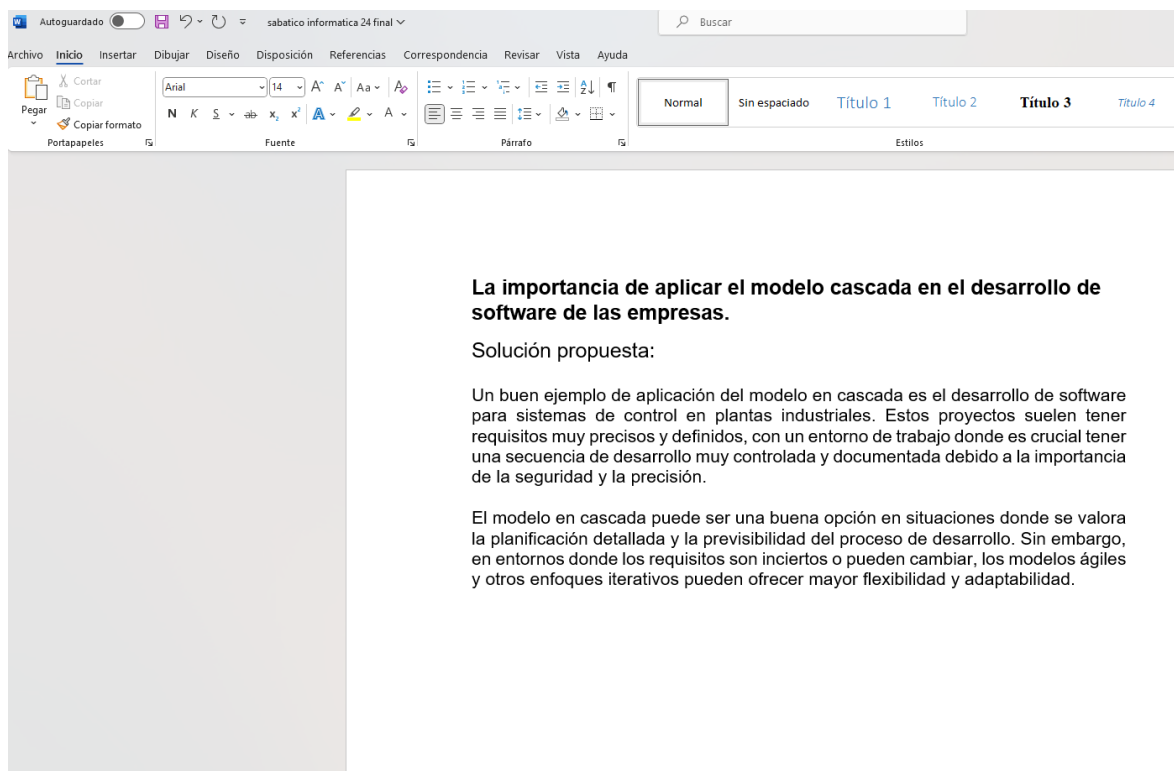
A3.1.1 Estudio de caso: La importancia de aplicar el modelo cascada en el desarrollo de software de las empresas.

Solución propuesta:

Un buen ejemplo de aplicación del modelo en cascada es el desarrollo de software para sistemas de control en plantas industriales. Estos proyectos suelen tener requisitos muy precisos y definidos, con un entorno de trabajo donde es crucial tener una secuencia de desarrollo muy controlada y documentada debido a la importancia de la seguridad y la precisión.

El modelo en cascada puede ser una buena opción en situaciones donde se valora la planificación detallada y la previsibilidad del proceso de desarrollo. Sin embargo, en entornos donde los requisitos son inciertos o pueden cambiar, los modelos ágiles y otros enfoques iterativos pueden ofrecer mayor flexibilidad y adaptabilidad.

El Modelo de Cascada es uno de los modelos prescriptivos más antiguos y utilizados en el desarrollo de software. Se caracteriza por un enfoque secuencial donde cada fase debe completarse antes de pasar a la siguiente.



A3.1.2 Ejercicio propuesto: Grafica en Excel que presente el porcentaje de empresas que trabajan con el modelo de cascada para el desarrollo de su software.

Solución propuesta:

El modelo en cascada, o "**waterfall**", es una metodología de desarrollo de software tradicional que sigue un enfoque secuencial y lineal. Aunque ha sido ampliamente utilizado en el pasado, su popularidad ha disminuido en favor de metodologías más ágiles y flexibles. A continuación, te proporciono un resumen de la adopción actual del modelo en cascada:

Uso del Modelo en Cascada

1. Industria del Software y Tecnología:

- **Baja Adopción:** En la industria del software y la tecnología, el modelo en cascada ha sido superado en gran medida por enfoques ágiles, como Scrum y Kanban, que ofrecen mayor flexibilidad y adaptabilidad. Se estima que solo **10-20%** de las empresas de desarrollo de software utilizan el modelo en cascada para nuevos proyectos. El modelo en cascada todavía puede ser utilizado en proyectos con requisitos muy claros y estables, pero su uso ha disminuido considerablemente.

2. Industria de Ingeniería y Construcción:

- **Uso Moderado:** En industrias como la ingeniería y la construcción, donde los requisitos tienden a ser más estables y bien definidos, el modelo en cascada sigue siendo relativamente popular. Aproximadamente **20-30%** de las empresas en estos sectores pueden seguir utilizando el modelo en cascada debido a la naturaleza de sus proyectos.

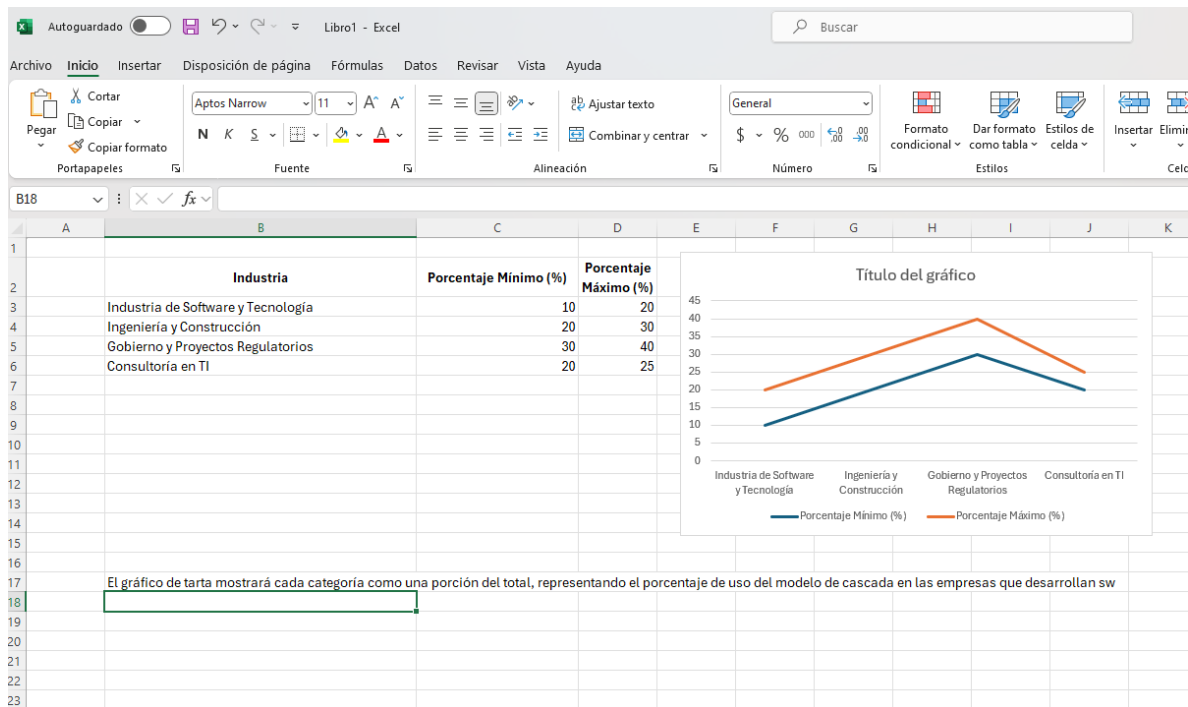
3. Empresas de Gobierno y Proyectos Regulatorios:

- **Uso Relativamente Alto:** En entornos gubernamentales y proyectos con requisitos regulatorios estrictos, el modelo en cascada todavía se utiliza con más frecuencia debido a su enfoque sistemático y documentado. Aproximadamente **30-40%** de las organizaciones gubernamentales y proyectos de gran escala pueden utilizar el modelo en cascada, aunque está siendo gradualmente reemplazado por enfoques más ágiles en algunos casos.

4. Empresas de Consultoría:

- **Uso Variado:** Las empresas de consultoría en TI pueden usar el modelo en cascada dependiendo del tipo de cliente y del proyecto. Sin embargo, el uso de metodologías ágiles ha ido aumentando, con alrededor de **20-25%** de las consultoras utilizando el modelo en cascada para proyectos específicos que lo requieran.

Industria	Porcentaje Mínimo (%)	Porcentaje Máximo (%)
Industria de Software y Tecnología	10	20
Ingeniería y Construcción	20	30
Gobierno y Proyectos Regulatorios	30	40
Consultoría en TI	20	25



A3.1.3 Aprendizaje basado en problemas: Basándose en la ingeniería de software, utilizar el modelo de cascada para desarrollar un sistema informático para una biblioteca, presentar en power point el resultado.

Solución propuesta:

Proyecto: Sistema de Gestión de Biblioteca

Objetivo: Desarrollar un sistema de gestión de biblioteca que permita a los usuarios buscar y reservar libros, y a los bibliotecarios gestionar inventarios y préstamos.

Fases del Modelo de Cascada

1. Requisitos

- **Objetivo:** Recopilar y documentar todos los requisitos del sistema de los diferentes stakeholders.
- **Actividades:**
 - Reuniones con bibliotecarios y usuarios.
 - Análisis de sistemas de biblioteca existentes.
 - Creación de un documento de requisitos del sistema (SRS).
- **Entregables:**
 - Documento de Especificación de Requisitos del Software (SRS).

2. Diseño del Sistema

- **Objetivo:** Definir la arquitectura del sistema y diseñar sus componentes.
- **Actividades:**
 - Diseño de la arquitectura del sistema.
 - Diseño de la base de datos.
 - Diseño de la interfaz de usuario.
 - Diagramas UML (diagramas de clases, diagramas de secuencia, etc.).
- **Entregables:**
 - Documento de Diseño del Sistema.
 - Diagramas UML.

3. Implementación

- **Objetivo:** Escribir el código para cada módulo y componente del sistema.
- **Actividades:**
 - Programación de los módulos del sistema.
 - Integración de los módulos.
 - Desarrollo de la interfaz de usuario.
- **Entregables:**
 - Código fuente del sistema.
 - Manual del programador.

4. Verificación

- **Objetivo:** Probar el sistema para asegurar que cumple con los requisitos especificados.
- **Actividades:**
 - Pruebas unitarias de cada módulo.
 - Pruebas de integración.
 - Pruebas de sistema.
 - Pruebas de aceptación del usuario.
- **Entregables:**
 - Plan de pruebas.
 - Casos de prueba y resultados.
 - Informe de pruebas.

5. Mantenimiento

- **Objetivo:** Corregir errores y mejorar el sistema tras su implementación.
- **Actividades:**
 - Corrección de errores reportados.
 - Actualizaciones y mejoras basadas en el feedback de los usuarios.
 - Mantenimiento preventivo.
- **Entregables:**
 - Actualizaciones del código fuente.
 - Informe de mantenimiento.

Detalle de Cada Fase

Requisitos

En esta fase, se identifican y documentan todos los requisitos funcionales y no funcionales del sistema. Los requisitos funcionales incluyen características específicas como la búsqueda de libros, la reserva de libros y la gestión de inventarios. Los requisitos no funcionales incluyen aspectos como la seguridad, la escalabilidad y el rendimiento del sistema.

Ejemplo de Requisitos Funcionales:

- El usuario debe poder buscar libros por título, autor o ISBN.
- El bibliotecario debe poder añadir, eliminar y actualizar información sobre los libros.
- El sistema debe enviar recordatorios de devolución a los usuarios.

Ejemplo de Requisitos No Funcionales:

- El sistema debe ser accesible a través de navegadores web modernos.
- El tiempo de respuesta para las búsquedas no debe exceder los 2 segundos.

Diseño del Sistema

En esta fase, se define cómo se implementarán los requisitos. Se elabora la arquitectura del sistema, que incluye el diseño de la base de datos y la definición de los componentes principales. También se crean los diagramas UML para visualizar la estructura y el comportamiento del sistema.

Ejemplo de Diagramas UML:

- Diagrama de Clases: Define las clases principales y sus relaciones.
- Diagrama de Secuencia: Muestra cómo los objetos interactúan en una secuencia temporal.

Implementación

Durante la implementación, los desarrolladores escriben el código para cada módulo del sistema según el diseño. Esto incluye la implementación de la lógica de negocio, la interfaz de usuario y las interacciones con la base de datos. La integración de los módulos asegura que todos los componentes funcionen juntos como un sistema cohesivo.

Ejemplo de Componentes:

- Módulo de Búsqueda: Implementa la funcionalidad de búsqueda de libros.
- Módulo de Gestión de Usuarios: Maneja el registro y la autenticación de usuarios.
- Interfaz de Usuario: Proporciona una interfaz web para interactuar con el sistema.

Verificación

Esta fase se enfoca en probar el sistema para garantizar que cumple con los requisitos especificados. Se realizan diferentes tipos de pruebas, incluyendo pruebas unitarias (cada módulo por separado), pruebas de integración (cómo funcionan los módulos juntos), pruebas de sistema (el sistema completo) y pruebas de aceptación del usuario (validación por parte de los usuarios finales).

Ejemplo de Pruebas:

- Prueba Unitaria: Verificar que la función de búsqueda retorna resultados correctos.
- Prueba de Integración: Asegurarse de que el módulo de búsqueda interactúa correctamente con la base de datos.
- Prueba de Sistema: Validar que todas las funcionalidades trabajan de manera conjunta sin errores.

- Prueba de Aceptación: Los usuarios finales prueban el sistema para asegurar que cumple con sus expectativas.

Mantenimiento

Después del despliegue, el sistema entra en la fase de mantenimiento. Esta fase incluye la corrección de errores que se descubren después del lanzamiento, la implementación de nuevas funcionalidades según sea necesario y el mantenimiento preventivo para asegurar que el sistema siga funcionando correctamente a lo largo del tiempo.

Ejemplo de Mantenimiento:

- Corrección de Errores: Solucionar un problema donde las búsquedas no devuelven resultados correctos.
- Nuevas Funcionalidades: Añadir una opción para que los usuarios reserven libros desde sus dispositivos móviles.
- Mantenimiento Preventivo: Actualizar bibliotecas de software para asegurar la compatibilidad y seguridad.

Conclusión

El Modelo de Cascada proporciona un enfoque estructurado y secuencial para el desarrollo de software, adecuado para proyectos donde los requisitos son bien entendidos desde el principio. Sin embargo, es menos flexible frente a cambios en los requisitos una vez que una fase ha sido completada. Este ejemplo de un Sistema de Gestión de Biblioteca muestra cómo el Modelo de Cascada puede ser aplicado de manera efectiva para desarrollar un sistema de información desde la recopilación de requisitos hasta el mantenimiento post-implementación.

The screenshot shows a PowerPoint presentation slide. The title is "Proyecto: Sistema de Gestión de Biblioteca" in blue. Below the title, the "Objetivo" (Objective) is stated: "Desarrollar un sistema de gestión de biblioteca que permita a los usuarios buscar y reservar libros, y a los bibliotecarios gestionar inventarios y préstamos." The "Fases del Modelo de Cascada" (Phases of the Cascade Model) are listed, with the first phase being "1. Requisitos". Under "1. Requisitos", there are two sub-points: "Objetivo: Recopilar y documentar todos los requisitos del sistema de los diferentes stakeholders." and "Actividades:" followed by a list of activities: "Reuniones con bibliotecarios y usuarios.", "Análisis de sistemas de biblioteca existentes.", and "Creación de un documento de requisitos del sistema (SRS)."

Proyecto: Sistema de Gestión de Biblioteca

Objetivo: Desarrollar un sistema de gestión de biblioteca que permita a los usuarios buscar y reservar libros, y a los bibliotecarios gestionar inventarios y préstamos.

Fases del Modelo de Cascada

1. Requisitos

- **Objetivo:** Recopilar y documentar todos los requisitos del sistema de los diferentes stakeholders.
- **Actividades:**
 - . Reuniones con bibliotecarios y usuarios.
 - . Análisis de sistemas de biblioteca existentes.
 - . Creación de un documento de requisitos del sistema (SRS).

The screenshot shows a PowerPoint presentation slide. The text explains that the Cascade Model provides a structured and sequential approach for software development, suitable for projects where requirements are well understood from the beginning. It notes that while it is less flexible to changes once a phase is completed, this example shows how the Cascade Model can be effectively applied to develop an information system from requirement gathering to post-implementation maintenance.

El Modelo de Cascada proporciona un enfoque estructurado y secuencial para el desarrollo de software, adecuado para proyectos donde los requisitos son bien entendidos desde el principio. Sin embargo, es menos flexible frente a cambios en los requisitos una vez que una fase ha sido completada. Este ejemplo de un Sistema de Gestión de Biblioteca muestra cómo el Modelo de Cascada puede ser aplicado de manera efectiva para desarrollar un sistema de información desde la recopilación de requisitos hasta el mantenimiento post-implementación.

UNIDAD IV

A4.1.1 Estudio de caso: Sistema de gestión de inventarios basado en un enfoque estructurado.

Solución propuesta:

El enfoque estructurado en ingeniería de software implica descomponer un sistema complejo en módulos o componentes más pequeños y manejables. Este método se centra en la planificación y el diseño antes de la implementación, asegurando que cada parte del sistema esté bien definida y documentada.

Sistema de Gestión de Inventarios

Objetivo: Desarrollar un sistema que permita a una empresa gestionar su inventario de productos, incluyendo la entrada y salida de productos, el seguimiento del inventario en tiempo real, y la generación de informes.

Fases del Enfoque Estructurado

1. **Análisis de Requisitos**
2. **Diseño del Sistema**
3. **Implementación**
4. **Pruebas**
5. **Mantenimiento**

Detalle de Cada Fase

Análisis de Requisitos

Objetivo: Identificar y documentar los requisitos del sistema.

Actividades:

- Reuniones con los stakeholders para entender sus necesidades.
- Análisis de sistemas de inventario existentes.
- Documentación de requisitos funcionales y no funcionales.

Ejemplo de Requisitos Funcionales:

- El sistema debe permitir la entrada de nuevos productos en el inventario.
- El sistema debe registrar la salida de productos del inventario.
- El sistema debe proporcionar informes sobre el estado del inventario.

Ejemplo de Requisitos No Funcionales:

- El sistema debe ser accesible desde múltiples ubicaciones.
- El tiempo de respuesta para consultar el inventario no debe exceder los 2 segundos.

Entregables:

- Documento de Especificación de Requisitos del Software (SRS).

Diseño del Sistema

Objetivo: Definir la estructura y componentes del sistema.

Actividades:

- Crear un diseño de alto nivel del sistema.
- Desarrollar diagramas de flujo de datos (DFD) para representar cómo la información se mueve dentro del sistema.
- Diseñar la base de datos.
- Crear diagramas de estructura para definir la jerarquía y las relaciones entre los módulos.

Ejemplo de Diagramas:

- **Diagrama de Flujo de Datos (DFD):**
 - **Nivel 0 (Contexto):** Representa el sistema como un único proceso interactuando con entidades externas (usuarios, sistemas de proveedores).
 - **Nivel 1 (Descomposición):** Descompone el sistema en procesos principales como "Gestión de Entradas", "Gestión de Salidas" y "Generación de Informes".
- **Diagrama de Estructura:**
 - **Módulo Principal:** Gestión de Inventarios.
 - **Submódulos:** Entrada de Productos, Salida de Productos, Consultas de Inventario, Generación de Informes.

Entregables:

- Documento de Diseño del Sistema.
- Diagramas de Flujo de Datos (DFD).
- Diagramas de Estructura.

Implementación

Objetivo: Codificar y desarrollar los módulos del sistema según el diseño especificado.

Actividades:

- Escribir el código para cada módulo.
- Implementar la base de datos y las conexiones.
- Desarrollar la interfaz de usuario.
- Integrar los módulos y realizar pruebas iniciales de integración.

Ejemplo de Componentes de Código:

- **Módulo de Entrada de Productos:**

```
python
Copiar código
def registrar_entrada(producto_id, cantidad, fecha):
    # Código para registrar la entrada de productos en la base de datos
```

- **Módulo de Salida de Productos:**

```
python
Copiar código
def registrar_salida(producto_id, cantidad, fecha):
    # Código para registrar la salida de productos en la base de datos
```

- **Interfaz de Usuario:**

```
html
Copiar código
<form action="/registrar_entrada" method="post">
    <!-- Campos para ingresar los detalles del producto -->
</form>
```

Entregables:

- Código fuente del sistema.
- Manual del programador.

Pruebas

Objetivo: Asegurar que el sistema cumple con los requisitos y funciona correctamente.

Actividades:

- Realizar pruebas unitarias de cada módulo.
- Realizar pruebas de integración para verificar que los módulos trabajan juntos.
- Realizar pruebas del sistema completo.
- Realizar pruebas de aceptación con usuarios finales.

Ejemplo de Pruebas:

- **Prueba Unitaria:** Verificar que la función registrar_entrada añade correctamente los productos a la base de datos.
- **Prueba de Integración:** Asegurarse de que el módulo de "Gestión de Entradas" se comunica correctamente con la base de datos y la interfaz de usuario.
- **Prueba de Sistema:** Validar que todas las funcionalidades del sistema trabajan juntas sin errores.
- **Prueba de Aceptación:** Los usuarios finales verifican que el sistema cumple con sus expectativas y requisitos.

Entregables:

- Plan de pruebas.
- Casos de prueba y resultados.
- Informe de pruebas.

Mantenimiento

Objetivo: Corregir errores y mejorar el sistema post-implementación.

Actividades:

- Corrección de errores reportados por los usuarios.
- Implementación de nuevas funcionalidades según las necesidades cambiantes de la empresa.
- Mantenimiento preventivo para asegurar la estabilidad del sistema.

Ejemplo de Mantenimiento:

- **Corrección de Errores:** Solucionar problemas donde el sistema no actualiza correctamente el inventario.
- **Nuevas Funcionalidades:** Añadir una funcionalidad para exportar informes de inventario a formato Excel.
- **Mantenimiento Preventivo:** Actualizar bibliotecas de software y asegurar la compatibilidad con nuevas versiones de sistemas operativos.

Entregables:

- Actualizaciones del código fuente.
- Documentación de mantenimiento.
- Informe de mantenimiento.

Conclusión

El enfoque estructurado proporciona una metodología clara y sistemática para el desarrollo de software, permitiendo una gestión eficiente de cada fase del proyecto. Este ejemplo de un Sistema de Gestión de Inventarios ilustra cómo se pueden aplicar las técnicas de análisis, diseño, implementación, pruebas y mantenimiento de manera ordenada y controlada para desarrollar un sistema robusto y eficiente.

Análisis de Requisitos

Objetivo: Identificar y documentar los requisitos del sistema.

Actividades:

- Reuniones con los stakeholders para entender sus necesidades.
- Análisis de sistemas de inventario existentes.
- Documentación de requisitos funcionales y no funcionales.

Ejemplo de Requisitos Funcionales:

- El sistema debe permitir la entrada de nuevos productos en el inventario.
- El sistema debe registrar la salida de productos del inventario.
- El sistema debe proporcionar informes sobre el estado del inventario.

Ejemplo de Requisitos No Funcionales:

- El sistema debe ser accesible desde múltiples ubicaciones.
- El tiempo de respuesta para consultar el inventario no debe exceder los 2 segundos.

Entregables:

- Documento de Especificación de Requisitos del Software (SRS).

Diseño del Sistema

Objetivo: Definir la estructura y componentes del sistema.

Página 1 de 100 24852 palabras Español (México) Accesibilidad: todo correcto

A4.2.2 Ejercicio propuesto: Grafica en Excel del desarrollo de sw basado en el enfoque estructurado y el enfoque orientado a objetos.

Solución propuesta:

Enfoque Estructurado vs. Orientado a Objetos en el Desarrollo de Software

1. Enfoque Estructurado

El enfoque estructurado, también conocido como método estructurado o enfoque en cascada, es una metodología tradicional que se basa en una secuencia lineal de fases (análisis, diseño, implementación, pruebas, y mantenimiento). Aunque fue ampliamente utilizado en el pasado, ha sido en gran parte reemplazado por metodologías más modernas.

- Estimación de Uso Actual:
 - 10-20% de las empresas de desarrollo de software utilizan enfoques estructurados para nuevos proyectos.
 - Este enfoque todavía se emplea en proyectos con requisitos fijos y donde la estabilidad y previsibilidad son críticas.
 - Es más común en industrias reguladas, como la ingeniería y el gobierno, donde la documentación y los procedimientos rigurosos son esenciales.

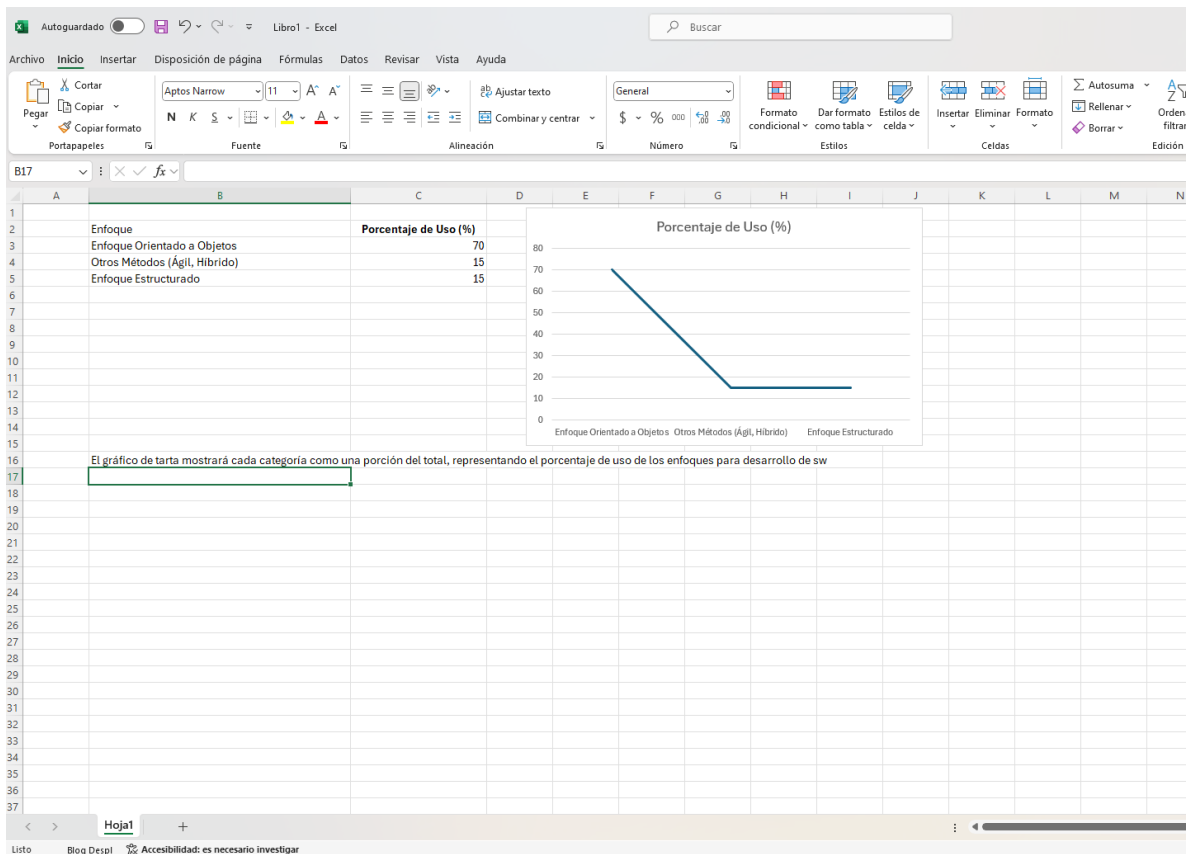
2. Enfoque Orientado a Objetos (OO)

El enfoque orientado a objetos (OO) se basa en la utilización de objetos y clases para modelar y desarrollar software. Es conocido por su modularidad, reusabilidad y flexibilidad. Este enfoque se ha vuelto más popular debido a sus ventajas en términos de mantenimiento y escalabilidad.

- Estimación de Uso Actual:
 - 60-80% de las empresas de desarrollo de software utilizan enfoques orientados a objetos, particularmente para proyectos nuevos.

- OO es ampliamente adoptado en la industria de software moderno debido a su capacidad para manejar la complejidad y facilitar el desarrollo ágil.
- Es prevalente en empresas de tecnología, startups, y en proyectos de software que requieren una arquitectura flexible y extensible.

Enfoque	Porcentaje de Uso (%)
Enfoque Estructurado	15
Enfoque Orientado a Objetos	70
Otros Métodos (Ágil, Híbrido)	15



A4.2.3 Aprendizaje basado en problemas: Establecer un Sistema de gestión de inventarios bajo el enfoque orientado a objetos, presentarlo en power point.

Solución propuesta:

Se decidió desarrollar un sistema de gestión de inventarios utilizando principios de **orientación a objetos (OO)** para abordar los problemas mencionados. La orientación a objetos permite modelar el sistema de manera más natural y modular, facilitando la escalabilidad y el mantenimiento.

Fases del Desarrollo del Sistema

1. Análisis y Diseño:

- **Identificación de Objetos:**

- Se identificaron los principales objetos en el dominio del problema, como **Producto**, **Categoría**, **Proveedor**, **Orden de Compra**, y **Inventario**.

- **Modelado UML:**

- Se crearon diagramas UML para definir la estructura y comportamiento del sistema, incluyendo diagramas de clases, casos de uso, y secuencia.

2. Implementación:

- **Clases y Objetos:**

- **Producto:** Representa un artículo en el inventario. Atributos incluyen ID, nombre, descripción, y precio. Métodos para actualizar precios y gestionar variantes.
- **Categoría:** Agrupa productos. Atributos incluyen ID y nombre.
- **Proveedor:** Información sobre proveedores de productos. Atributos incluyen ID, nombre, y contacto.
- **Orden de Compra:** Registro de compras de inventario. Atributos incluyen ID, fecha, y lista de productos.
- **Inventario:** Gestión del stock. Métodos para agregar o reducir inventario basado en órdenes de compra y ventas.

- **Interfaz de Usuario:**

- Se diseñó una interfaz gráfica intuitiva utilizando principios de diseño OO, permitiendo una gestión

eficiente de los objetos y una experiencia de usuario mejorada.

3. Pruebas:

- **Pruebas Unitarias:**

- Se realizaron pruebas unitarias en cada clase y método para asegurar que funcionaran correctamente.

- **Pruebas de Integración:**

- Se verificó la interacción entre objetos para asegurar que los componentes del sistema se integraran correctamente.

4. Despliegue:

- **Capacitación del Personal:**

- Se proporcionó capacitación a los usuarios finales sobre el nuevo sistema.

- **Implementación Gradual:**

- El sistema se desplegó en fases para minimizar el impacto en las operaciones diarias y facilitar la adaptación.

5. Mantenimiento y Soporte:

- **Actualizaciones Continuas:**

- Se estableció un plan para realizar actualizaciones periódicas y agregar nuevas funcionalidades según fuera necesario.

- **Soporte Técnico:**

- Se proporcionó soporte técnico continuo para resolver cualquier problema que surgiera.

Beneficios del Sistema Basado en Orientación a Objetos

1. Modularidad y Reusabilidad:

- La orientación a objetos permitió crear módulos independientes (clases) que se podían reutilizar y actualizar de manera aislada sin afectar el resto del sistema.

2. Escalabilidad:

- El sistema es fácilmente escalable para manejar nuevos tipos de productos, categorías y funcionalidades adicionales.

3. Mejor Integración:

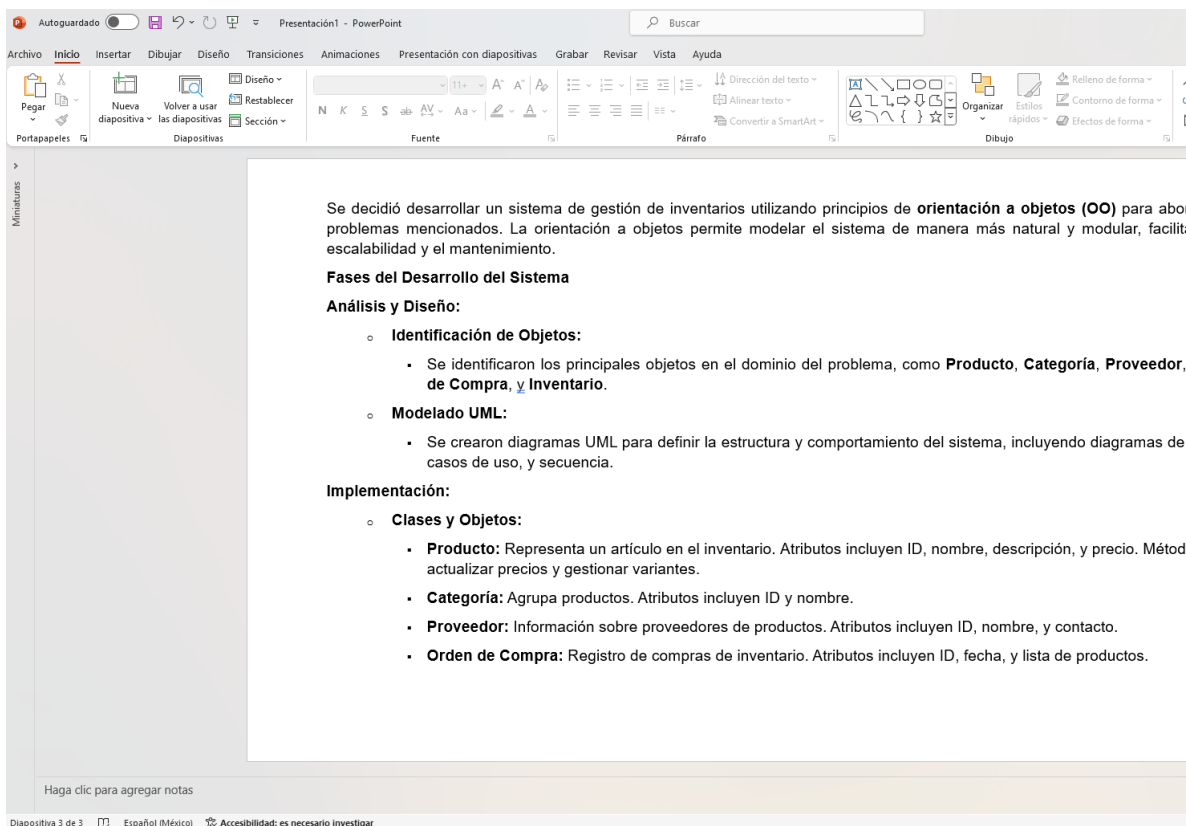
- La estructura basada en objetos facilitó la integración con otros sistemas, como el de contabilidad y ventas, mediante la creación de interfaces bien definidas.

4. Interfaz de Usuario Mejorada:

- La nueva interfaz gráfica, diseñada con principios de OO, es más intuitiva y facilita la gestión del inventario.

5. Facilidad de Mantenimiento:

- El sistema es más fácil de mantener y extender debido a su diseño modular y a la claridad de la estructura basada en objetos.



UNIDAD V

A5.1.1 Estudio de caso: Sistema de inventario de entradas y salida de productos de una empresa.

La planificación de un proyecto de sistemas de información es crucial para asegurar su éxito. Aquí se detalla un ejemplo de planificación para un proyecto de desarrollo de un Sistema de Gestión de Inventarios.

Objetivo: Desarrollar un sistema que permita gestionar el inventario de una empresa, incluyendo la entrada y salida de productos, el seguimiento del inventario en tiempo real y la generación de informes.

Alcance: El sistema debe incluir módulos para la gestión de productos, gestión de entradas y salidas, consultas de inventario y generación de informes.

Fases del Proyecto

1. Inicio del Proyecto
2. Planificación
3. Ejecución
4. Monitoreo y Control
5. Cierre del Proyecto

Detalle de la Planificación

Inicio del Proyecto

Objetivos:

- Definir el proyecto y obtener la aprobación inicial.
- Identificar los stakeholders y formar el equipo del proyecto.

Actividades:

- Creación del Acta de Constitución del Proyecto.
- Identificación de stakeholders.
- Definición del alcance inicial del proyecto.
- Formación del equipo del proyecto.

Entregables:

- Acta de Constitución del Proyecto.
- Lista de stakeholders.
- Definición del Alcance Inicial.

Planificación

Objetivos:

- Desarrollar un plan detallado para guiar la ejecución y control del proyecto.
- Identificar y gestionar los riesgos.
- Definir el cronograma y presupuesto.

Actividades:

- Definir el alcance detallado del proyecto.
- Crear el cronograma del proyecto.
- Estimar costos y desarrollar el presupuesto.
- Identificar riesgos y planificar su gestión.
- Planificar la calidad, comunicación y recursos humanos.
- Definir los criterios de éxito y las métricas de desempeño.

Entregables:

- Plan de Proyecto.
- Cronograma del Proyecto.
- Presupuesto del Proyecto.
- Plan de Gestión de Riesgos.
- Plan de Gestión de Calidad.
- Plan de Gestión de Comunicación.
- Plan de Gestión de Recursos Humanos.

Detalle del Plan de Proyecto

Definición del Alcance

Objetivos:

- Describir detalladamente lo que incluirá el proyecto.
- Asegurar que todas las partes interesadas estén alineadas con los objetivos del proyecto.

Actividades:

- Reuniones con stakeholders para entender sus necesidades y expectativas.
- Documentar los requisitos del sistema.
- Crear la Declaración de Alcance del Proyecto.

Entregables:

- Documento de Requisitos.

- Declaración de Alcance del Proyecto.

Cronograma del Proyecto

Objetivos:

- Establecer una línea de tiempo clara para las actividades del proyecto.
- Asegurar que todas las actividades estén debidamente secuenciadas y programadas.

Actividades:

- Identificar las tareas y actividades necesarias.
- Estimar la duración de cada tarea.
- Definir las dependencias entre las tareas.
- Crear un diagrama de Gantt.

Ejemplo de Cronograma:

Tarea	Duración	Inicio	Fin	Dependencias
Definir requisitos	2 semanas	01/08/2024	14/08/2024	
Diseño del sistema	3 semanas	15/08/2024	04/09/2024	Definir requisitos
Implementación	8 semanas	05/09/2024	30/10/2024	Diseño del sistema
Pruebas	4 semanas	31/10/2024	27/11/2024	Implementación
Capacitación y despliegue	2 semanas	28/11/2024	11/12/2024	Pruebas

Presupuesto del Proyecto

Objetivos:

- Estimar los costos del proyecto y asegurarse de que se ajuste al presupuesto disponible.

Actividades:

- Identificar los recursos necesarios (humanos, tecnológicos, etc.).
- Estimar los costos de cada recurso.
- Crear un presupuesto detallado.

Ejemplo de Presupuesto:

Concepto	Costo Estimado
Recursos Humanos	\$50,000
Hardware y Software	\$20,000
Pruebas y Calidad	\$10,000
Capacitación y Despliegue	\$5,000
Contingencias	\$5,000
Total	\$90,000

Gestión de Riesgos

Objetivos:

- Identificar posibles riesgos y planificar cómo manejarlos.
- Minimizar el impacto de los riesgos en el proyecto.

Actividades:

- Realizar un análisis de riesgos.
- Desarrollar un plan de respuesta a los riesgos.
- Asignar responsables para la gestión de riesgos.

Ejemplo de Análisis de Riesgos:

Riesgo	Probabilidad	Impacto	Estrategia	Responsable
Retrasos en la implementación	Alta	Alto	Plan contingencia	Jefe de Proyecto
Fallos técnicos	Media	Medio	Pruebas exhaustivas	Equipo de Desarrollo
Sobrecostos	Baja	Alto	Control costos	Financiero

Plan de Calidad

Objetivos:

- Asegurar que el sistema cumpla con los estándares de calidad requeridos.

Actividades:

- Definir los criterios de calidad.
- Planificar actividades de control y aseguramiento de calidad.
- Asignar responsabilidades de calidad.

Entregables:

- Plan de Gestión de Calidad.
- Criterios de Calidad.

Plan de Comunicación

Objetivos:

- Asegurar una comunicación efectiva entre todos los stakeholders.

Actividades:

- Definir los canales de comunicación.
- Establecer la frecuencia y formato de las comunicaciones.
- Asignar responsables de comunicación.

Entregables:

- Plan de Gestión de Comunicación.

g. Plan de Recursos Humanos

Objetivos:

- Definir y gestionar los recursos humanos necesarios para el proyecto.

Actividades:

- Identificar las habilidades y competencias necesarias.
- Asignar roles y responsabilidades.
- Planificar la capacitación y el desarrollo del equipo.

Entregables:

- Plan de Gestión de Recursos Humanos.
- Matriz de Roles y Responsabilidades.

3. Ejecución

Objetivos:

- Llevar a cabo las actividades del proyecto según el plan.

Actividades:

- Desarrollar el sistema.
- Coordinar al equipo de proyecto.
- Gestionar los recursos y presupuesto.
- Mantener la comunicación con los stakeholders.

Entregables:

- Sistema desarrollado.
- Informes de progreso.

4. Monitoreo y Control

Objetivos:

- Supervisar el progreso del proyecto y asegurar que se mantenga en línea con el plan.

Actividades:

- Monitorear el desempeño del proyecto.
- Gestionar los cambios y riesgos.
- Realizar revisiones de calidad.
- Informar a los stakeholders sobre el progreso.

Entregables:

- Informes de desempeño.
- Actualizaciones del plan de proyecto.
- Informes de calidad.

5. Cierre del Proyecto

Objetivos:

- Completar todas las actividades del proyecto y formalizar su cierre.

Actividades:

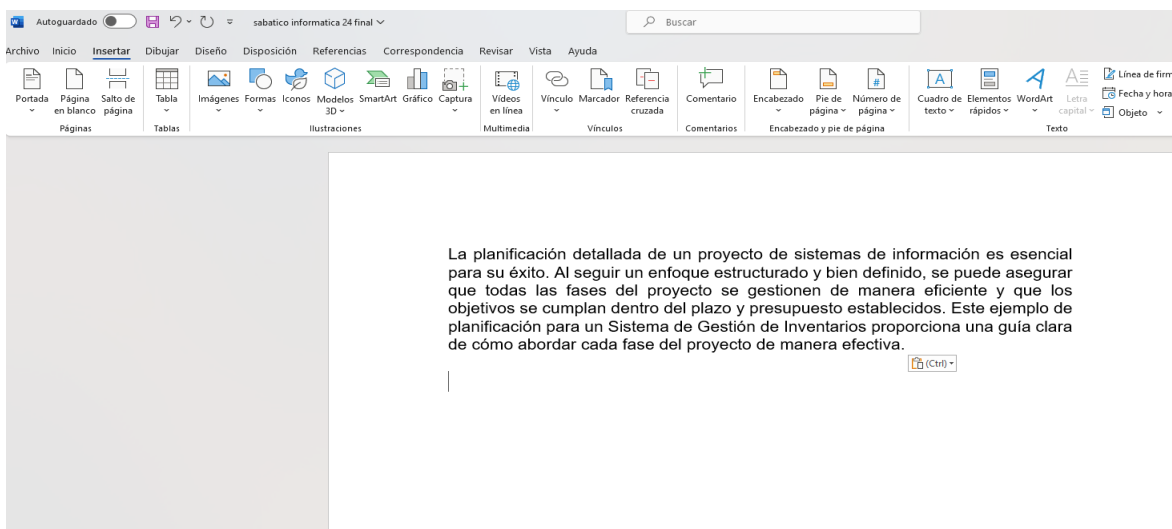
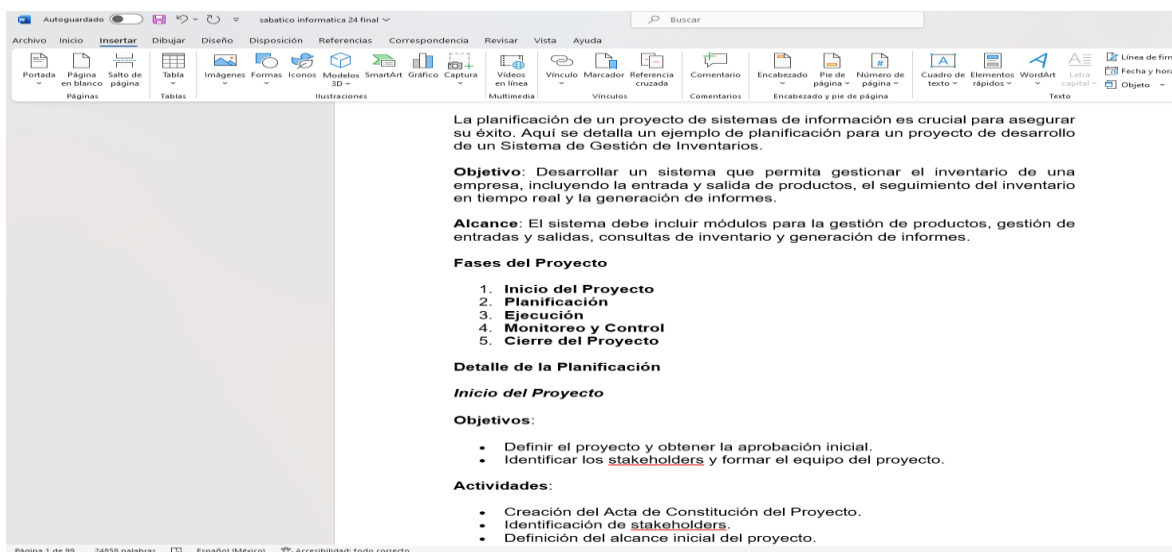
- Verificar que todos los entregables estén completos y aceptados.
- Documentar las lecciones aprendidas.
- Liberar los recursos del proyecto.
- Realizar la entrega formal del sistema.

Entregables:

- Informe de cierre del proyecto.
- Documentación de lecciones aprendidas.
- Entrega formal del sistema.

Conclusión

La planificación detallada de un proyecto de sistemas de información es esencial para su éxito. Al seguir un enfoque estructurado y bien definido, se puede asegurar que todas las fases del proyecto se gestionen de manera eficiente y que los objetivos se cumplan dentro del plazo y presupuesto establecidos. Este ejemplo de planificación para un Sistema de Gestión de Inventarios proporciona una guía clara de cómo abordar cada fase del proyecto de manera efectiva.



A5.5.1 Ejercicio propuesto: Grafica en Excel para representar el desarrollo de un sistema de gestión de inventarios.

Solución propuesta:

Proyecto: Desarrollo de un Sistema de Gestión de Inventarios

- 1. Definición del Proyecto**
 - Objetivo: Desarrollar un sistema de gestión de inventarios para mejorar la precisión y eficiencia.
 - Alcance: Incluye el desarrollo del sistema, capacitación del personal y migración de datos. No incluye el mantenimiento post-implementación.
- 2. EDT**
 - Nivel 1: Iniciación del Proyecto, Planificación, Ejecución, Monitoreo y Control, Cierre.
 - Nivel 2: Dentro de Planificación -> Requisitos, Cronograma, Presupuesto, Gestión de Riesgos, Plan de Calidad, Plan de Comunicación, Gestión del Cambio.
- 3. Cronograma del Proyecto**
 - Tareas: Recolección de Requisitos, Diseño del Sistema, Desarrollo, Pruebas, Implementación, Capacitación.
 - Dependencias: Desarrollo depende del Diseño, Pruebas dependen del Desarrollo, etc.
 - Diagrama de Gantt: Visualización de tareas con fechas de inicio y fin.
- 4. Asignación de Recursos**
 - Recursos: Equipo de desarrollo (5 personas), Hardware, Software de gestión de proyectos, Consultores externos.
 - Asignaciones: Equipo de desarrollo -> Desarrollo y Pruebas, Consultores -> Capacitación.
- 5. Presupuesto del Proyecto**
 - Estimación de Costos: \$150,000 (Desarrollo: \$100,000, Hardware: \$20,000, Capacitación: \$10,000, Otros: \$20,000).
 - Presupuesto Detallado: Incluye costos de personal, equipos, formación y contingencias.
- 6. Gestión de Riesgos**
 - Identificación de Riesgos: Falta de habilidades técnicas, Retrasos en entrega de hardware, Resistencia al cambio.
 - Evaluación: Falta de habilidades (Alta probabilidad, Alto impacto), Retrasos hardware (Media probabilidad, Alto impacto).
 - Respuestas: Capacitación adicional, contratos con cláusulas de penalización por retrasos, sesiones de sensibilización para el personal.
- 7. Plan de Calidad**
 - Criterios: El sistema debe tener una precisión de inventario del 99.9%.
 - Aseguramiento: Revisiones de código, pruebas unitarias y de integración.
 - Control: Auditorías de calidad periódicas.
- 8. Comunicación del Proyecto**
 - Plan: Reuniones semanales del equipo, informes mensuales a la dirección.
 - Informes: Progreso, hitos alcanzados, riesgos identificados.
- 9. Gestión del Cambio**

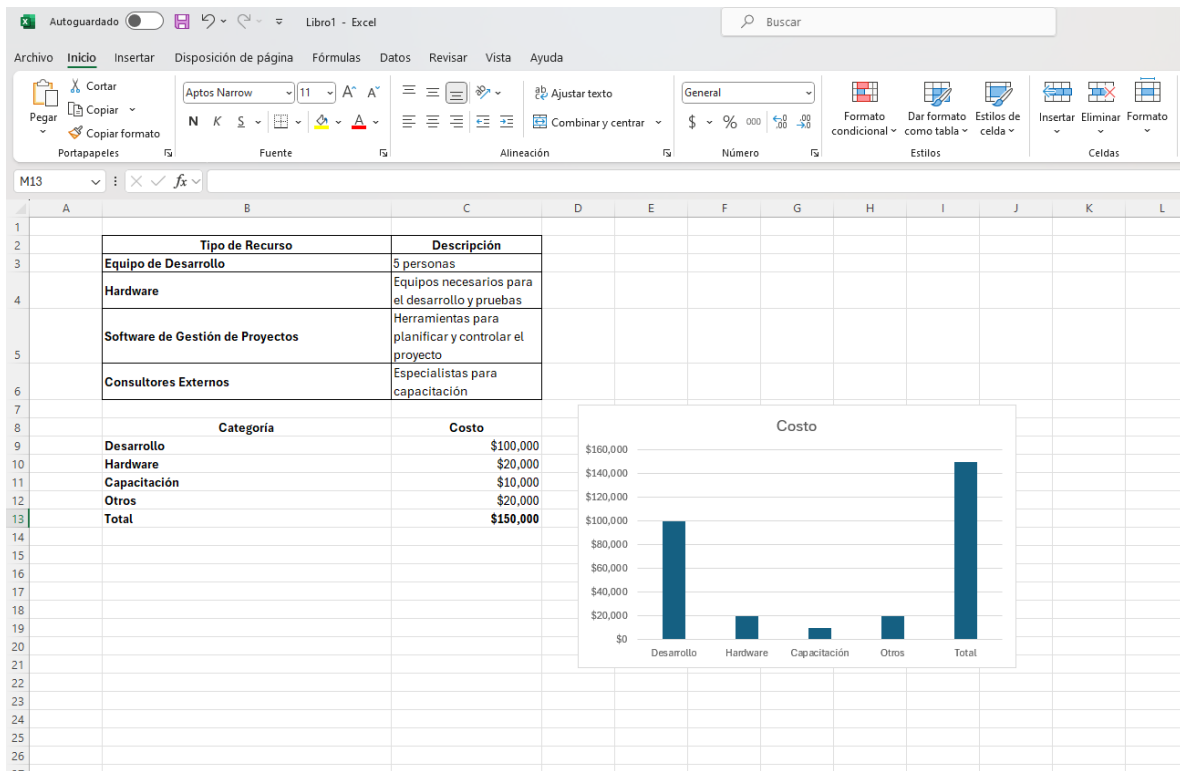
- Plan: Formularios de solicitud de cambio, comité de control de cambios.
- Control: Registro de cambios aprobados, seguimiento de la implementación de cambios.

10. Revisión y Aprobación

- Revisión: Revisión completa del plan por el equipo de gestión.
- Aprobación: Firma de los stakeholders clave antes de iniciar la ejecución.

Herramientas y Técnicas de Planificación

- **Software de Gestión de Proyectos:** Herramientas como Microsoft Project, JIRA, Asana, Trello.
- **Diagramas de Gantt y Pert:** Visualización y planificación del cronograma.
- **Matrices de Riesgos:** Evaluación y priorización de riesgos.
- **Modelos de Presupuesto:** Plantillas y software para estimar y controlar los costos.



A5.6.1 Aprendizaje basado en problemas: Cronograma del proyecto de software de un sistema de inventario de entradas y salida de productos de una empresa, presentar resultados en power point.

Solución propuesta:

Fase de Planificación

- **Reunión de Inicio del Proyecto:** 1 día
- **Definición de Requisitos:** 5 días
- **Análisis de Factibilidad:** 3 días

Fase de Diseño

- **Diseño de Arquitectura:** 5 días
- **Diseño de Base de Datos:** 3 días
- **Diseño de Interfaz de Usuario:** 5 días

Fase de Desarrollo

- **Desarrollo del Backend:** 15 días
- **Desarrollo del Frontend:** 15 días
- **Integración de Componentes:** 10 días

Fase de Pruebas

- **Pruebas Unitarias:** 5 días
- **Pruebas de Integración:** 5 días
- **Pruebas de Sistema:** 5 días
- **Pruebas de Aceptación de Usuario (UAT):** 5 días

Fase de Despliegue

- **Preparación del Entorno de Producción:** 3 días
- **Despliegue en Producción:** 2 días

Fase de Mantenimiento

- **Corrección de Errores Post-Despliegue:** Continuo
- **Actualizaciones y Mejoras:** Según necesidad

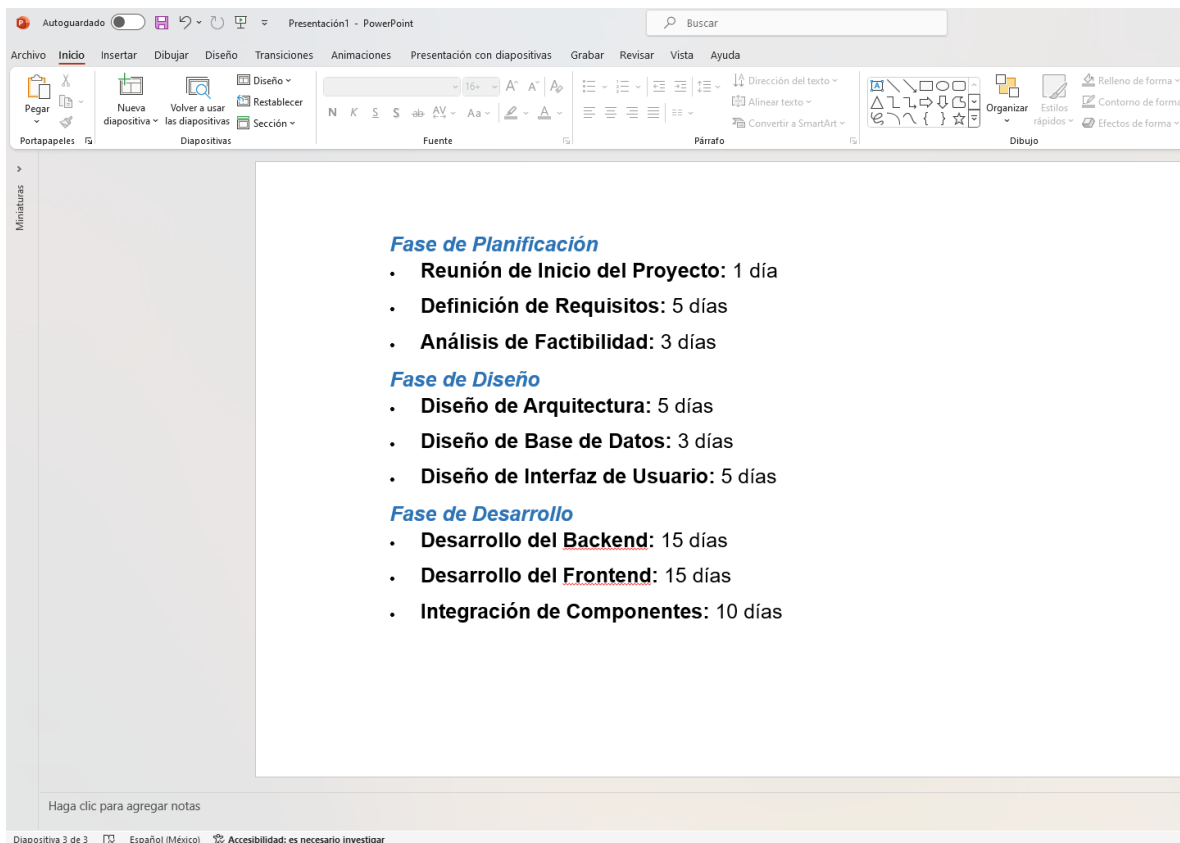
Ejemplo de Diagrama de Gantt para un Proyecto de Software

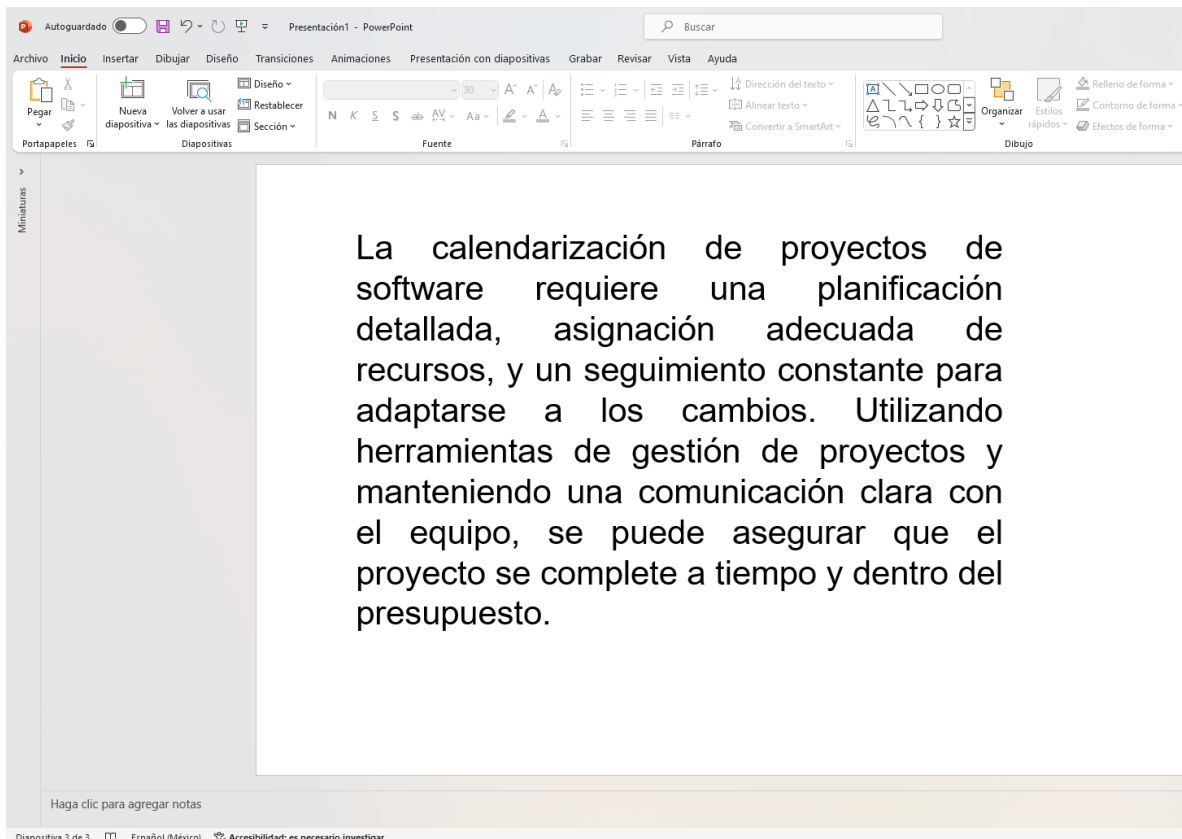
Herramientas Recomendadas

- **Jira:** Excelente para gestión ágil de proyectos de software con funcionalidades de seguimiento de problemas y gestión de sprints.
- **Trello:** Ideal para la gestión visual de tareas y trabajo en equipo.
- **Asana:** Útil para la gestión de proyectos colaborativos.
- **Microsoft Project:** Completa herramienta de gestión de proyectos, ideal para proyectos complejos con múltiples dependencias.

Conclusión

La calendarización de proyectos de software requiere una planificación detallada, asignación adecuada de recursos, y un seguimiento constante para adaptarse a los cambios. Utilizando herramientas de gestión de proyectos y manteniendo una comunicación clara con el equipo, se puede asegurar que el proyecto se complete a tiempo y dentro del presupuesto.





ANEXO 2. Plantilla del reporte de actividades

Unidad *[Número de la unidad]*

Actividad *[Número de actividad]. [Nombre de la actividad]*

Nombre del alumno *[Nombre del alumno en caso de ser individual la actividad]*

Integrantes: *[Lista de los alumnos que realizaron la actividad en caso de ser en equipo]*

Procedimiento (Descripción)

- a) Planteamiento del problema.
- b) Desarrollar la actividad indicada.
- c) Si la actividad es desarrollo de software, realizar lo siguiente:
 - 1. Codificar en prolog, java o c.
 - 2. Crear el programa en prolog o netbeans.
 - 3. Compilar y ejecutar el programa.

Resultados y Conclusiones

- a) En los resultados se deben agregar las vistas del código y de la ejecución o en su caso, evidencia fotográfica del desarrollo de la actividad o video.
- b) En las conclusiones establecer en máximo 5 renglones la experiencia adquirida.

Nota.- Lo deberá entregar en formato PDF.

ANEXO 3. Rúbrica

Para la evaluación de las actividades, se debe considerar la siguiente rúbrica, que va desde el 1% hasta el 100%.

Criterios	Deficiente	Regular	Bueno	Muy Bueno
Reporte (20%)	1% a 5% El documento no cumple con la estructura y formato explicitado por el docente.	6% a 10% El documento cumple con la estructura y formato explicitado por el docente.	11% a 15% Además, no tiene errores ortográficos y gramaticales.	16% a 20% Además, está bien redactado y tiene coherencia de contenido.
Codificación (40%)	1% a 10% Utiliza instrucciones que no resuelven el ejercicio.	11% a 20% Utiliza instrucciones que no son acordes para resolver el ejercicio.	21% a 30% Utiliza las instrucciones necesarios para resolver el ejercicio, aunque no son los más adecuados.	31% a 40% Utiliza las instrucciones más adecuados para resolver el ejercicio. Usa buenas prácticas de codificación.
Funcionamiento (40%)	1% a 10% No funciona.	11% a 20% El funcionamiento tiene fallos importantes.	21% a 30% El funcionamiento tiene fallos sin importancia.	31% a 40% El funcionamiento es completo.

ANEXO 4. Instrumentación Didáctica

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 1 de 13

Tecnológico Nacional de México
Subdirección Académica
Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales

Nombre de la asignatura: FUNDAMENTOS DE SISTEMAS DE INFORMACIÓN
Plan de estudios: IFE-1015
Clave de la asignatura: AGG3
Periodo: Agosto-diciembre/2023
Horas teoría - horas prácticas - créditos: 3 - 1 - 4
Atributos de Egreso:

1. Caracterización de la asignatura

Esta asignatura aporta al perfil del Ingeniero en Informática en las siguientes competencias:

- . Formula, desarrolla y gestiona el desarrollo de proyectos de software para incrementar la competitividad en las organizaciones, considerando las normas de calidad vigentes.
- . Realiza consultorías relacionadas con la función informática para la mejora continua de la organización.
- . Se desempeña profesionalmente con ética, respetando el marco legal, la pluralidad y la conservación del medio ambiente.

Se sabe que las economías de los países desarrollados dependen en gran parte del software y más sistemas son actualmente controlados por éste. La Ingeniería de Software concierne a teorías, métodos y herramientas para el desarrollo profesional de software por lo que el gasto en la Ingeniería de Software, representa un alto porcentaje del PIB de los países desarrollados.

Fundamentos de Sistemas de Información es una asignatura que comprende la conceptualización de los sistemas de información en su campo dimensional y tipos en que éstos se presentan, además de proporcionar con exactitud y amplitud el contexto de las etapas del proceso

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 2 de 13

de Ingeniería del Software y las herramientas empleadas para éste. Lo anterior, hace importante que las organizaciones en colaboración con profesionales del área aprendan a adoptar un enfoque orientado a brindar servicios utilizando software que cumplan con los estándares de calidad.

Así, la asignatura de Fundamentos de Sistemas de Información proporciona al estudiante el marco referencial para conocer y analizar los procesos inherentes a la ingeniería del software, mismos que se desarrollarán en asignaturas posteriores.

Esta asignatura se relaciona con las asignaturas de: Administración de los Recursos y Función Informática, Análisis y modelado de Sistemas de Información, Desarrollo e implementación de Sistemas de Información y Calidad de los Sistemas de Información.

2. Intención didáctica

La asignatura está diseñada en cinco temas. El primer tema es una introducción a los conceptos fundamentales de los Sistemas de Información, su importancia, alcances y tipos. El segundo tema se enfoca a estudiar a detalle los procesos de la ingeniería del software referentes a la elaboración de sistemas de información. En el tercer tema se analizan los modelos prescriptivos para el desarrollo de Sistemas de Información valorando la permanencia con evolución de éstos. En el cuarto tema se revisan los paradigmas de la ingeniería del software, estructurado, orientado a objetos y de vanguardia, centrándose en las características de cada uno para que el estudiante llegado el momento pueda distinguir el más conveniente al sistema en desarrollo. En el último tema se aborda la importancia de la gestión de proyectos de sistemas de información haciendo énfasis desde la viabilidad del posible proyecto hasta la calendarización del mismo.

El enfoque sugerido para la asignatura requiere que las actividades prácticas promuevan el desarrollo de habilidades para todo el proceso de ingeniería de software, debido a que en las asignaturas subsecuentes aplicará los conocimientos adquiridos al desarrollo de un sistema de información que responda a un problema del entorno, y entonces requerirá la parte conceptual de la ingeniería de software así como las herramientas de vanguardia que existen para analizar, diseñar, construir, probar e implementar un sistema de información.

3. Competencia de la asignatura

Conoce e identifica las metodologías y procesos de la ingeniería de software relacionados con los sistemas de información.

4. Análisis por competencias específicas

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 3 de 13

Competencia No. : 1 Introducción a los sistemas de Información

Descripción:

Conoce el contexto de los sistemas de información.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
1.1 Concepto de Sistemas de Información 1.2 Dimensiones de los sistemas de información. 1.3 Clasificación de los sistemas de información.	- Elaborar un glosario con la terminología de los sistemas de información (SI). - Discutir en plenaria los resultados. - Elaborar un resumen de las dimensiones de los SI. - Presentar los resultados en plenaria. - Ejemplificar los tipos de SI. Presentar los resultados en un cuadro sinóptico. computadora.	- DAR CONCEPTO DE SISTEMAS DE INFORMACIÓN - DAR LA CLASIFICACIÓN DE LOS SISTEMAS DE INFORMACIÓN	* Capacidad de análisis y síntesis * Comunicación oral y escrita * Habilidades básicas de manejo de la computadora * Habilidad para buscar y analizar información proveniente de fuentes diversas. * Trabajo en equipo * Compromiso ético * Capacidad de aprender * Habilidad para trabajar en forma autónoma * Búsqueda del logro	15

Indicadores de alcance	Valor del indicador
A. Elabore ejemplos de la Clasificación de los Sist. de Información	90
B. Elabore la Evidencia correspondiente	10

Niveles de desempeño

--	--	--	--

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 4 de 13

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	B	Evaluación formativa de la competencia	Atributos de Egreso
1_ Entregar la tarea solicitada	90	90		Revisar la Tarea encargada	
2_ Entregar el Portafolio de evidencia	10		10	Revisar el Portafolio de evidencia	
Total:	100	90	10		

Competencia No. : 2 Ingeniería de software

Descripción:

Conoce los fundamentos de la ingeniería de software asociados a los sistemas de información.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
				15

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.		Código: ITSLP-SAC-PG-001-02
			Revisión: 4
			Página: 5 de 13
2.1 Definición de Ingeniería de software 2.2 Capas de la ingeniería de software 2.3 Etapas de la ingeniería de software 2.3.1 Análisis 2.3.2 Diseño 2.3.3 Desarrollo 2.3.4 Pruebas 2.3.5 Implementación 2.3.6 Mejora continua	. Gestionar información sobre las capas de la ingeniería de software y plasmar los resultados en un cuadro sinóptico ejemplificando cada capa. . Gestionar información sobre las etapas de la ingeniería de software y ejemplificarlas entrevistando a profesionistas que desempeñen roles asociados con ellas. Presentar los resultados en un informe técnico por equipo.	2.1 Definición de Ingeniería de software 2.2 Capas de la ingeniería de software 2.3 Etapas de la ingeniería de software 2.3.1 Análisis 2.3.2 Diseño 2.3.3 Desarrollo 2.3.4 Pruebas 2.3.5 Implementación 2.3.6 Mejora continua	* Capacidad de análisis y síntesis * Comunicación oral y escrita * Habilidades básicas de manejo de la computadora * Habilidad para buscar y analizar información proveniente de fuentes diversas. * Trabajo en equipo * Compromiso ético * Capacidad de aprender * Habilidad para trabajar en forma autónoma * Búsqueda del logro

Indicadores de alcance	Valor del indicador
A. Conoce los fundamentos de la ingeniería de software	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.		Código: ITSLP-SAC-PG-001-02
			Revisión: 4
			Página: 6 de 13

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	60	60	Conocimiento del concepto	
2._ Examen	40	40	Conocimiento del concepto	
Total:	100	100		

Competencia No. : 3 Modelos prescriptivos del desarrollo de sistemas de información

Descripción:

Identifica los modelos prescriptivos relacionados al desarrollo de sistemas de información.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
3.1. Modelo en Cascada. 3.2. Modelos Evolutivos. 3.3. Modelos Especiales. 3.4. El Proceso Unificado de Desarrollo de software. 3.5. Modelo de Proceso de Software IEEE. 3.6. Herramientas CASE	. Gestionar información sobre los modelos prescriptivos del desarrollo de software y plasmarlos los resultados en un resumen. . Elaborar un cuadro comparativo entre los modelos descriptivos. . Elaborar un cuadro comparativo de herramientas case	Gestionar información sobre los modelos prescriptivos del desarrollo de software y plasmarlos los resultados en un resumen. . Elaborar un cuadro comparativo entre los modelos descriptivos. . Elaborar un cuadro comparativo de herramientas case	* Capacidad de análisis y síntesis * Comunicación oral y escrita * Habilidades básicas de manejo de la computadora * Habilidad para buscar y analizar información proveniente de fuentes diversas. * Trabajo en equipo * Compromiso ético * Capacidad de aprender * Habilidad para trabajar en forma autónoma * Búsqueda del logro	10

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 7 de 13

Indicadores de alcance	Valor del indicador
A. Identifica los modelos prescriptivos	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	60	60	Conocimiento del concepto	
2._ Examen	40	40	Conocimiento del concepto	
Total:	100	100		

Competencia No. : 4 Paradigmas de la Ingeniería de Software

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 8 de 13

Descripción:

Conoce los paradigmas de ingeniería de software.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
4.1. El enfoque estructurado 4.2. El enfoque orientado a objetos 4.3 Enfoques de vanguardia	. Gestionar información sobre los paradigmas de ingeniería de software. Presentar los resultados en una línea del tiempo. . Buscar ejemplos de SI orientados a objetos. Presentar los resultados obtenidos en un informe técnico.	. Gestionar información sobre los paradigmas de ingeniería de software. Presentar los resultados en una línea del tiempo. . Buscar ejemplos de SI orientados a objetos. Presentar los resultados obtenidos en un informe técnico.	* Capacidad de análisis y síntesis * Comunicación oral y escrita * Habilidades básicas de manejo de la computadora * Habilidad para buscar y analizar información proveniente de fuentes diversas. * Trabajo en equipo * Compromiso ético * Capacidad de aprender * Habilidad para trabajar en forma autónoma * Búsqueda del logro	10

Indicadores de alcance	Valor del indicador
A. Conoce los paradigmas de ingeniería de software.	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.		Código: ITSLP-SAC-PG-001-02
			Revisión: 4
			Página: 9 de 13

	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	60	60	Conocimiento del concepto	
2._ Examen	40	40	Conocimiento del concepto	
Total:	100	100		

Competencia No. : 5 Gestión de proyectos de sistemas de información.

Descripción:

Comprende las fases que integran la gestión de proyectos de sistemas de información.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
5.1. Actividades de gestión. 5.2. Estudio de viabilidad del proyecto. 5.3. Análisis Costo/Beneficio. 5.4. Gestión de riesgos. 5.5. Planificación del proyecto. 5.6. Calendarización del proyecto.	. Gestionar información sobre los elementos de la gestión de proyectos de SI y plasmar resultados en un resumen. . Acudir a una empresa del entorno, entrevistar a un	. Gestionar información sobre los elementos de la gestión de proyectos de SI y plasmar resultados en un resumen. . Acudir a una empresa del entorno, entrevistar a	* Capacidad de análisis y síntesis * Comunicación oral y escrita * Habilidades básicas de manejo de la computadora * Habilidad para buscar y analizar información proveniente de fuentes diversas.	30

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.		Código: ITSLP-SAC-PG-001-02
			Revisión: 4
			Página: 10 de 13

	líder de proyecto de SI e identificar el proceso en la gestión de algún proyectos SI. . Identificar un área de oportunidad para el desarrollo de un SI, formular el problema, estudiar la viabilidad y planearlo. Presentar los resultados en informe de proyecto.	un líder de proyecto de SI e identificar el proceso en la gestión de algún proyectos SI. . Identificar un área de oportunidad para el desarrollo de un SI, formular el problema, estudiar la viabilidad y planearlo. Presentar los resultados en informe de proyecto.	* Trabajo en equipo * Compromiso ético * Capacidad de aprender * Habilidad para trabajar en forma autónoma * Búsqueda del logro	
--	---	--	---	--

Indicadores de alcance	Valor del indicador
A. Comprende las fases que integran la gestión de proyectos de sistemas de información	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 11 de 13

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1. Tarea	60	60	Conocimiento del concepto	
2. Examen	40	40	Conocimiento del concepto	
Total:	100	100		

5. Fuentes de información y Apoyos didácticos

Fuentes de información: 1. Cohen y Asin.; Sistemas de Información un enfoque de toma de decisiones. 3ª Edición. Mc Graw Hill.2000. 2. EDWARDS, CHRIS; JOHN WARD y ANDY BYTHEWAY. Fundamentos de Sistemas de Información. 2da. Edición. Ed. Prentice Hall. 1998. 3. KENDALL, KENNETH E. Y KENDALL, JULIE E. Análisis y Diseño de Sistemas. 6ª Edición; Ed. Pearson Educación México. 2005. 4. Larman, Craig. UML y Patrones. 2ª. edición. Pearson 5. Laudon K. Laudon, J.; Sistema de Información Gerencial. Administración de la Empresa Digital. 10ª Edición; Ed. Pearson Prentice Hall. 2008. 6. PRESSMAN, ROGER S.; Ingeniería de software un Enfoque práctico; Ed. Mc. Graw. Hill. 2007. 7. SOMMERVILLE, IAN; Ingeniería de Software, Edit. Addison Wesley; 2005.	Apoyos didácticos: Proporcionados por el docente.
--	---

6. Calendarización de evaluación en semanas (6):

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 12 de 13

Semana:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
TP	ED		EF1			EF2		EF3		EF4						EF5	ES	
TR	ED					EF2		EF3		EF4						EF5	ES	
SD					SD1				SD2					SD3				

TP=tiempo planeado

TR=tiempo real

SD=seguimiento departamental

ED=evaluación diagnóstica

EFn=evaluación formativa(competencia específica n)

ES=evaluación sumativa

Fechas de evaluación por unidad

Tema	Evaluación diagnóstica	Evaluación formativa	Inicio Tema	Fin Tema	Sumativa
1	21/08/2023	08/09/2023	21/08/2023	08/09/2023	
2		29/09/2023	11/09/2023	29/09/2023	
3		13/10/2023	02/10/2023	13/10/2023	
4		27/10/2023	16/10/2023	27/10/2023	
5		08/12/2023	30/10/2023	08/12/2023	11/12/2023

Fecha de elaboración:25/07/2024

ITSLP-SAC-PG-001-02

Rev. 4

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 13 de 13

GUILLERMO GÁMEZ GALEANA

Nombre del(de la) profesor(a)

MC. ANTONIO MAYER SÁNCHEZ

Nombre del(de la) Jefe(a) de Departamento Académico

ITSLP-SAC-PG-001-02

Rev. 4



TECNOLÓGICO
NACIONAL DE MÉXICO®



INSTITUTO TECNOLÓGICO
DE SAN LUIS POTOSÍ

TECNOLÓGICO NACIONAL DE MÉXICO INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSÍ

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

**Reporte Final de trabajo de período sabático del:
ISC. GUILLERMO GÁMEZ GALEANA**

**B.1 Elaboración de Apuntes para la materia
Algoritmos y Lenguajes de Programación INC-1005**

**De acuerdo con el proyecto calendarizado en el documento PS-02
"B.1 Elaboración de Apuntes elaborado durante el ejercicio del periodo
Sabático 1/2024".**

Soledad de Graciano Sánchez, S.L.P., a 1 de febrero de 2025

CARTA DE RECONOCIMIENTO DEL AUTOR DE LOS DERECHOS A FAVOR DEL TECN

Ciudad de México, 01/08/2024

TECNOLÓGICO NACIONAL DE MÉXICO.

PRESENTE.

Bajo protesta de decir verdad, GUILLERMO GÁMEZ GALEANA, personal docente adscrito al INSTITUTO TECNOLÓGICO DE SAN LUIS POTOSI del Tecnológico Nacional de México, manifiesto que en cumplimiento de mis actividades relacionadas con el Año Sabático, elaboré la obra titulada "B.1 ELABORACIÓN DE APUNTES PARA LA MATERIA ALGORITMOS Y LENGUAJES DE PROGRAMACIÓN INC-1005".

Con base en lo anterior, y con fundamento en los artículos 83 de la Ley Federal del Derecho de Autor y 46 de su Reglamento, reconozco que el Tecnológico Nacional de México es titular de los derechos patrimoniales sobre la misma y le corresponden las facultades relativas a la divulgación, integridad de la obra y de colección, conservando el derecho a figurar como autor. Asimismo, respondo por la autoría y originalidad de la citada obra; y relevo de toda responsabilidad al Tecnológico Nacional de México de cualquier demanda o reclamación que llegara a formular alguna persona física o moral que considere que con esta obra es afectado en alguno de los derechos protegidos por la Ley en cita, asumiendo todas las consecuencias legales y económicas.

ATENTAMENTE



ISC GUILLERMO GÁMEZ GALEANA

Contribución académica

La contribución de este documento a los objetivos educativos del programa educativo de la materia de Algoritmos y lenguajes de programación la cual forma parte de la retícula de la Carrera de Ingeniería Industrial del Tecnológico Nacional de México, es lograr que el estudiante desarrolle un pensamiento lógico, que entienda y aplique las metodologías para resolver problemas a partir de lenguajes de programación, con el fin de que sea capaz de resolver problemas basados en procesos de las áreas de producción para la toma de decisiones. Se busca presentar situaciones problemáticas, asociando éstas con la realidad del estudiante, mejorando su aprendizaje. Con el uso de este material, permitirá clarificar los conceptos básicos de las tecnologías de la información y comunicación utilizadas en el ambiente laboral, con el fin de establecer las bases para dar propuestas de solución que resuelvan problemas de ingeniería en la industria y establecer el marco conceptual donde el estudiante y el docente podrán tener una comunicación efectiva. El docente debe hacer ver al estudiante las implicaciones o consecuencias del concepto revisado ya que en la actualidad las implicaciones ciencia Tecnología- Sociedad son muy importantes; traduciendo todo esto en un esfuerzo integrador de conocimiento

Introducción

Una de las preocupaciones más notables y que debe estar presente en cualquier organismo es la mejora continua. Para el caso de las instituciones Educativas, es necesario buscar mecanismos que coadyuven a elevar la calidad y eficiencia educativa. Uno de estos mecanismos lo proporciona precisamente la elaboración de apuntes, ya que contribuye de manera tangible a la adquisición del conocimiento por parte de los estudiantes, siendo, además, un parámetro de control de avance en los programas de estudio al estar unificado en todos los grupos de la misma materia. Con la elaboración de apuntes de la materia de Algoritmos y lenguajes de programación se pretende consolidar los conocimientos adquiridos por el estudiante con la teoría, concretando en un problema real las explicaciones recibidas. Estos conceptos son de gran importancia y deben ser seguidos por todos los estudiantes en forma individual o en equipos reducidos. Con estos apuntes se permite al alumno tener la capacidad de pensar para abordar y resolver nuevos problemas reales, además, representan una herramienta útil de apoyo para el docente, ya que además de contener el material requerido, le permitirá dosificarlo de acuerdo con tiempos y temas.

Contenido

Contribución académica	3
Introducción	4
CONCEPTOS GENERALES.....	8
UNIDAD I.....	8
INTRODUCCIÓN A LA COMPUTACIÓN Y HOJA DE CÁLCULO	8
1.1 Introducción a la Computación	8
1.2 Sistemas Operativos	9
1.3 Elementos de Excel.....	9
1.4 Fórmulas y Funciones.....	10
1.5 Macros	11
1.6 Aplicaciones.....	12
UNIDAD II	13
DESARROLLO DE LÓGICA ALGORÍTMICA.....	13
2.1. Metodología para la solución de problemas.....	13
2.2. Metodología para el diseño de software	14
2.3. Elementos y reglas de los lenguajes algorítmicos	15
2.4. Implementación de algoritmos	16
UNIDAD III.....	18
INTRODUCCIÓN A LA PROGRAMACIÓN DE UN LENGUAJE ESTRUCTURADO	18
3.1 Introducción y estructura del entorno de un lenguaje de programación	18
3.2 Estructura básica de un programa	18
3.3 Tipos de datos	19
3.4 Identificadores	19
3.5 Almacenamiento, direccionamiento y representación en memoria	20
3.6 Proposición de asignación	20
3.7 Operadores, operandos y expresiones	20
3.8 Prioridad de operadores, evaluación de expresiones	20
3.9 Elaboración de programas.....	21
3.10 Pruebas y depuración	21
UNIDAD IV	22
ESTRUCTURAS SELECTIVAS Y DE REPETICIÓN	22
4.1 Selectiva simple	22

4.2 Selectiva doble.....	22
4.3 Selectiva anidada.....	23
4.4 Selectiva múltiple	23
4.5 Repetir mientras.....	24
4.6 Repetir hasta	24
4.7 Repetir desde	25
4.8 Elaboración de programas.....	25
UNIDAD V.....	26
Arreglos y archivos.....	26
5.1 Arreglos unidimensionales	26
5.2 Arreglos bidimensionales y multidimensionales	26
5.3 Apertura, entrada-salida de datos, y cierre de archivos	27
5.4 Elaboración de programas.....	28
6.1 Introducción.....	28
6.2 Funciones estándar	28
6.3 Entrada y salida de datos.....	29
UNIDAD VI	29
FUNCIONES	29
6.4 Funciones definidas por el usuario	29
6.5 Pase por valor.....	30
6.6 Pase por referencia.....	30
6.7 Elaboración de programas.....	30
PRACTICAS.....	31
UNIDAD I.....	31
U.1 Practica 1 Introducción a la computación	31
U.1 Practica 2 Elementos de Excel	33
U.1 Practica 3 Introducción a la computación	35
U.1 Practica 4 Aplicaciones	36
UNIDAD II	39
U.2 Practica 1 Metodología para el diseño de software.....	39
U.2 Practica 2 Diagramas de Flujo	40
U.2 Practica 3 Pseudocódigo.....	45
U.2 Practica 4 Implementación de algoritmos.....	46

UNIDAD III.....	49
U.3 Practica 1: Conversión de temperatura.....	49
U.3 Practica 2: Cálculo del promedio.....	50
U.3 Practica 4: Calculadora de descuento.....	52
U.3 Practica 5: Conversión de tiempo	53
U.3 Practica 6: Comparación de números	54
UNIDAD IV	55
U.4 Practica 5 Estructuras selectivas y de repetición	55
U.4 Practica 1 Búsqueda Lineal	55
U.4 Practica 2 Ordenamiento por Selección	56
U.4 Practica 3 Números Primos hasta un Límite	57
U.4 Practica 4 Verificar un Palíndromo	58
U.4 Practica 5 Validación de Contraseña Fuerte.....	59
UNIDAD V.....	60
U.5 Practica 1: Promedio de números almacenados en un arreglo	60
U.5 Practica 2: Buscar un valor en un arreglo.....	61
U.5 Practica 3: Guardar nombres en un archivo.....	62
U.5 Practica 4: Leer nombres desde un archivo y ordenarlos.....	63
U.5 Practica 5: Leer y analizar un archivo CSV	64
UNIDAD 6.....	66
U.6 Practica 1: Simulador de cajero automático	66
U.6 Practica 2: Pago de nómina a empleados.....	69
BIBLIOGRAFIA	73

CONCEPTOS GENERALES

UNIDAD I

INTRODUCCIÓN A LA COMPUTACIÓN Y HOJA DE CÁLCULO.

1.1 Introducción a la Computación

La computación se refiere al uso de computadoras para resolver problemas, realizar cálculos y procesar información. Combina hardware (componentes físicos) y software (programas) para realizar tareas específicas de manera eficiente.

Componentes de la Computación

1. **Hardware:** La parte tangible de una computadora.
 - **Dispositivos de entrada:** Teclado, ratón, escáner.
 - **Dispositivos de salida:** Monitor, impresora.
 - **Procesadores:** CPU (Unidad Central de Procesamiento), GPU (Procesador Gráfico).
 - **Almacenamiento:** Discos duros, SSD, unidades USB.
2. **Software:** La parte intangible, como programas y sistemas operativos.
 - **Sistemas operativos:** Windows, Linux, macOS.
3. **Aplicaciones:** Procesadores de texto, hojas de cálculo, navegadores.
4. **Datos y Procesamiento:**
 - Entrada de datos → Procesamiento → Salida de información.
5. **Importancia de la Computación**
 - Automatización de procesos.
 - Almacenamiento y análisis de grandes cantidades de datos.
 - Conexión global mediante redes como Internet.
 - Innovación en diversas áreas como la medicina, educación y negocios.

1.2 Sistemas Operativos

Un sistema operativo (SO) es un programa esencial que actúa como intermediario entre el hardware de la computadora y el usuario. Gestiona los recursos del sistema y facilita la ejecución de aplicaciones.

Funciones Principales

Gestión de procesos:

- Asigna tiempo de CPU y recursos a las aplicaciones.
- Ejemplo: Multitarea.

Gestión de memoria:

- Controla el uso de la memoria RAM.

Gestión de dispositivos:

- Coordina la comunicación entre hardware y software.
- Ejemplo: Impresoras, cámaras.

Gestión de archivos:

- Permite la creación, lectura y almacenamiento de datos en discos.

Seguridad:

- Protege contra accesos no autorizados.

Tipos de Sistemas Operativos

- **De escritorio:** Windows, macOS, Linux.
- **Móviles:** Android, iOS.
- **De servidor:** Windows Server, Ubuntu Server.
- **Tiempo real:** Sistemas embebidos en dispositivos médicos o de automatización industrial.

1.3 Elementos de Excel

Excel es una herramienta de hoja de cálculo que permite organizar, analizar y presentar datos en formato de tabla, con funcionalidades avanzadas como gráficos, tablas dinámicas y macros.

Elementos Principales

1. **Celdas:** La unidad básica donde se ingresan datos.
2. **Filas y Columnas:** Organización horizontal y vertical para estructurar información.
3. **Barra de fórmulas:** Muestra el contenido o las fórmulas de una celda seleccionada.
4. **Hojas de cálculo:** Páginas dentro de un libro de trabajo.
5. **Gráficos:** Representaciones visuales de datos.

Tipos de Datos en Excel

- **Texto:** Etiquetas o descripciones.
- **Números:** Valores numéricos para cálculos.
- **Fechas y horas:** Formatos especiales.
- **Fórmulas:** Operaciones matemáticas

1.4 Fórmulas y Funciones

Fórmulas

Son expresiones que permiten realizar cálculos personalizados. Inician con el signo igual (=).

- Ejemplo: =A1+B1 suma los valores de las celdas A1 y B1.
- Funciones

Son fórmulas predefinidas que simplifican cálculos.

Matemáticas:

- =SUMA(A1:A10) suma los valores de un rango.
- =PROMEDIO(A1:A10) calcula el promedio.

Lógicas:

- =SI(A1>10, "Alto", "Bajo") evalúa condiciones.

Texto:

- =CONCATENAR(A1, B1) combina texto de varias celdas.

Fecha y hora:

- =HOY() devuelve la fecha actual.

Errores Comunes

- #DIV/0!: División por cero.
- #N/A: Valor no disponible.
- #REF!: Referencia no válida

1.5 Macros

Las macros son conjuntos de instrucciones programadas que automatizan tareas repetitivas en Excel. Se pueden grabar o escribir utilizando VBA (Visual Basic for Applications).

Elementos Básicos de las Macros

1. **Grabador de Macros:** Herramienta para registrar acciones realizadas por el usuario.
2. **VBA:** Lenguaje de programación integrado para personalizar macros.

Ejemplo:

- Macro para cambiar el formato de una tabla:

```
Sub FormatearTabla()  
    Columns("A:C").AutoFit  
    Range("A1:C1").Font.Bold = True  
End Sub
```

Aplicaciones Prácticas

- Automatización de reportes.
- Generación de gráficos.
- Procesamiento masivo de datos.

1.6 Aplicaciones

Uso de Hojas de Cálculo

Gestión Financiera:

- Creación de presupuestos.
- Cálculo de préstamos con la función PAGO.

Análisis de Datos:

- Uso de tablas dinámicas para resumir información.
- Representación visual mediante gráficos.

Inventarios:

- Seguimiento de existencias.
- Cálculo de costos totales.

Educación:

- Registro de calificaciones.
- Seguimiento de asistencia.

Automatización

Uso de macros para realizar tareas repetitivas como:

- Actualizar formatos.
- Generar reportes automatizados.

Conexión con Otras Herramientas

- Importación y exportación de datos desde otras plataformas como Access o SQL Server.
- Integración con PowerPoint para presentaciones.

UNIDAD II

DESARROLLO DE LÓGICA ALGORÍTMICA.

2.1. Metodología para la solución de problemas

La **metodología para la solución de problemas** se refiere a un conjunto de pasos sistemáticos y lógicos para abordar un problema y encontrar la solución más efectiva. Existen diversas metodologías, pero la más comúnmente aceptada sigue las siguientes etapas:

Definición del problema:

- El primer paso consiste en entender completamente el problema. Esto implica analizar y describir el problema de manera precisa. Para hacer esto, debes preguntar qué, por qué, cuándo, cómo y quién está involucrado en el problema.

Análisis del problema:

- Una vez que el problema ha sido definido, el siguiente paso es desglosarlo en sus partes componentes. Aquí, debes identificar las causas subyacentes, los factores que influyen y cualquier restricción existente.

Generación de posibles soluciones:

- En esta fase, se deben idear diferentes enfoques o soluciones para abordar el problema. Aquí es importante la creatividad y la lluvia de ideas. Cada solución debe ser evaluada con base en su viabilidad, costo, tiempo de implementación y otros criterios relevantes.

Selección de la mejor solución:

- De todas las soluciones propuestas, se elige la más efectiva o adecuada para el contexto. Para tomar una decisión, se analizan las ventajas y desventajas de cada opción.

Implementación de la solución:

- Esta fase involucra poner en práctica la solución seleccionada. Puede implicar la programación de un algoritmo, la construcción de un producto, la toma de decisiones operativas, etc.

Evaluación de la solución:

- Una vez implementada la solución, es crucial evaluar su efectividad. Se verifica si se resolvió el problema inicial y si la solución sigue siendo viable a largo plazo.

Mantenimiento:

- Después de evaluar y verificar el éxito de la solución, puede ser necesario realizar ajustes para optimizar la solución, o adaptarla a nuevos cambios en las condiciones o contexto.

2.2. Metodología para el diseño de software

El diseño de software se refiere a los procesos utilizados para crear una solución de software que cumpla con los requisitos del usuario de manera efectiva y eficiente. Existen varias metodologías que se utilizan para guiar el desarrollo de un sistema software:

Top Down (De arriba hacia abajo)

- En este enfoque, se empieza con una visión global del sistema y luego se divide en partes más pequeñas. Es como una estrategia de desglose jerárquico.
- Este enfoque es útil cuando se tiene un concepto general del sistema y se quiere detallar a medida que se avanza en su diseño.
- Ejemplo: El diseño de un sistema de gestión de usuarios comenzaría definiendo las funcionalidades generales (administración de usuarios, seguridad, interfaz, etc.) y luego se desglosaría en detalles más específicos (gestión de contraseñas, asignación de permisos).

Bottom Up (De abajo hacia arriba)

- El enfoque Bottom Up comienza con el diseño de los componentes más pequeños o más simples del sistema, que luego se combinan para formar el sistema completo.

- Este enfoque es adecuado cuando ya se tiene experiencia con los componentes básicos y es más fácil construir a partir de ellos.
- Ejemplo: En lugar de definir la arquitectura global primero, puedes comenzar desarrollando módulos o clases individuales y luego integrarlos en el sistema global.

Modular

- La programación modular divide un sistema en componentes o módulos más pequeños, cada uno con una función o responsabilidad bien definida. Los módulos interactúan entre sí a través de interfaces bien definidas.
- La modularización facilita la reutilización del código, el mantenimiento y la prueba de partes individuales del sistema sin afectar el todo.
- Ejemplo: En un sistema de venta en línea, los módulos pueden incluir: gestión de productos, gestión de usuarios, pagos, y envíos.

Programación estructurada

- La programación estructurada promueve el uso de estructuras de control como secuencias, condicionales y bucles, en lugar de utilizar instrucciones de salto, como el GOTO.
- El objetivo es crear un flujo de ejecución claro y fácil de seguir, que haga que el código sea más comprensible y fácil de depurar.
- Ejemplo: Se usa if-else en lugar de goto para las decisiones dentro del código.

2.3. Elementos y reglas de los lenguajes algorítmicos

Los lenguajes algorítmicos permiten expresar soluciones a problemas en términos de instrucciones claras para la máquina. Para facilitar su comprensión, se emplean ciertos elementos y reglas.

2.3.1. Diagramas de flujo

- Un diagrama de flujo es una representación visual de un algoritmo que muestra los pasos del proceso de forma secuencial.
- Utiliza símbolos estandarizados:
- Óvalo: Inicio o fin.

- Rectángulo: Acción o proceso (una tarea a realizar).
- Rombo: Decisión o condición (si/entonces).
- Flechas: Indican el flujo de ejecución entre los pasos.
- Los diagramas de flujo son útiles para representar visualmente la lógica de un algoritmo y facilitar su comprensión por parte de los desarrolladores.

2.3.2. Pseudocódigo

- El pseudocódigo es una forma intermedia entre el lenguaje natural y un lenguaje de programación. No está vinculado a ninguna sintaxis específica de programación, pero utiliza estructuras similares.
- Permite describir algoritmos de forma clara sin preocuparse por los detalles de la implementación en un lenguaje particular.
- Ejemplo de pseudocódigo:

```
css
Copiar código
Inicio
  Leer A
  Leer B
  Sumar A y B
  Imprimir Resultado
Fin
```

2.4. Implementación de algoritmos

Los algoritmos pueden ser implementados en diferentes formas, según sus características y estructuras de control.

2.4.1. Secuenciales

- Los algoritmos secuenciales ejecutan un conjunto de instrucciones de manera continua, una tras otra.
- No hay decisiones ni repeticiones involucradas.
- Ejemplo: Imprimir el resultado de una suma de dos números.

2.4.2. Selectivos

- Los algoritmos selectivos incluyen decisiones. Dependiendo de si una condición es verdadera o falsa, el flujo del programa puede seguir diferentes caminos.
- Usan estructuras de control condicional, como if, else, switch.

- Ejemplo: Si un número es mayor que 10, imprimir "Mayor"; de lo contrario, imprimir "Menor".

2.4.3. Repetitivos

- Los algoritmos repetitivos repiten un bloque de instrucciones mientras se cumpla una condición.
- Usan estructuras de control de repetición como for, while, o do-while.
- Ejemplo: Imprimir números del 1 al 10 usando un bucle for.

2.5. Pruebas y depuración

La prueba y depuración son procesos esenciales para asegurar que el software funcione correctamente y libre de errores.

Pruebas

- Las pruebas se realizan para verificar que el software cumpla con los requisitos y funcione correctamente.
- Existen diferentes tipos de pruebas:
 - **Pruebas unitarias:** Verifican que cada componente individual funcione como se espera.
 - **Pruebas de integración:** Aseguran que los diferentes componentes del sistema funcionen juntos de manera adecuada.
 - **Pruebas del sistema:** Validan el comportamiento del sistema completo.
 - **Pruebas de aceptación:** Se realizan para confirmar que el sistema cumple con las expectativas del usuario final.

Depuración

- La depuración es el proceso de identificar, rastrear y corregir errores en el código. Para hacerlo, se pueden utilizar herramientas automáticas de depuración o realizar análisis manuales.
- El proceso puede incluir técnicas como la impresión de mensajes de error, el uso de puntos de interrupción, o la revisión cuidadosa del código para localizar errores lógicos o de sintaxis.

UNIDAD III

INTRODUCCIÓN A LA PROGRAMACIÓN DE UN LENGUAJE ESTRUCTURADO.

3.1 Introducción y estructura del entorno de un lenguaje de programación

Un entorno de programación es el conjunto de herramientas que permiten a los desarrolladores escribir, ejecutar, depurar y mantener código en un lenguaje de programación.

Elementos comunes del entorno:

- **Editores de texto o IDEs:** Herramientas como Visual Studio Code, IntelliJ IDEA o PyCharm facilitan la escritura de código con funciones como resaltado de sintaxis y autocompletado.
- **Compiladores e intérpretes:** Transforman el código fuente en un formato ejecutable (ejemplo: GCC para C o Python Interpreter).
- **Depuradores:** Herramientas que ayudan a identificar y corregir errores en el código.
- **Sistemas de control de versiones:** Como Git, para gestionar cambios en el código.

Ejemplo: En Python, el entorno puede ser un IDE como PyCharm, que permite escribir y ejecutar scripts, depurar errores y administrar librerías.

3.2 Estructura básica de un programa

La estructura básica de un programa depende del lenguaje, pero generalmente incluye:

1. **Encabezados o importaciones:** Librerías y dependencias necesarias.
2. **Declaraciones:** Variables, constantes, y configuraciones iniciales.
3. **Cuerpo principal:** El flujo lógico y las operaciones del programa.

Ejemplo en Python:

```
# Importación de librerías  
import math
```

```
# Declaración de variables  
radio = 5
```

```
# Código principal  
area = math.pi * (radio ** 2)
```

```
print(f'El área del círculo es: {area}')
```

3.3 Tipos de datos

Los tipos de datos representan el formato y el tipo de información que puede manejar un programa.

Clasificación:

- **Primitivos:**
 - Enteros (int en Python, int en C++)
 - Flotantes (float en Python, double en C++)
 - Booleanos (True/False en Python, bool en C++)
 - Caracteres (str en Python, char en C++)
- **Compuestos:**
 - Listas, arrays, diccionarios o estructuras.

Ejemplo en Python:

```
numero = 10      # Tipo entero
precio = 19.99   # Tipo flotante
es_valido = True # Tipo booleano
nombre = "Juan"  # Tipo cadena
```

3.4 Identificadores

Un identificador es el nombre que se asigna a elementos del programa, como variables, funciones o clases.

Reglas:

1. Deben comenzar con una letra o un guión bajo (_).
2. No pueden incluir espacios ni caracteres especiales.
3. Son sensibles a mayúsculas y minúsculas en la mayoría de los lenguajes (ej.: Python).

Ejemplo:

```
_variable = 10
nombre_usuario = "Ana"
def sumar(a, b):
    return a + b
```

3.5 Almacenamiento, direccionamiento y representación en memoria

Los datos en un programa se almacenan en la memoria del ordenador. Cada dato ocupa un espacio de memoria identificado por una dirección.

Conceptos clave:

- **Variables:** Almacenan datos temporales en memoria.
- **Dirección de memoria:** Cada variable tiene una ubicación única.
- **Representación interna:** Los datos se representan en binario. Por ejemplo, int ocupa 4 bytes en muchos sistemas.

Ejemplo en Python:

```
x = 42
print(id(x)) # Devuelve la dirección de memoria del dato almacenado
```

3.6 Proposición de asignación

La asignación permite almacenar un valor en una variable utilizando el operador =.

Ejemplo:

```
x = 10 # Asigna el valor 10 a la variable x
y = x + 5 # Asigna a y el valor de x + 5
```

3.7 Operadores, operandos y expresiones

- **Operadores:** Símbolos que realizan operaciones (ej.: +, -, *, /).
- **Operandos:** Valores sobre los que actúan los operadores.
- **Expresión:** Combinación de operadores y operandos que devuelve un resultado.

Ejemplo:

```
a = 5
b = 3
resultado = a * b + 10 # Expresión
```

3.8 Prioridad de operadores, evaluación de expresiones

La prioridad determina el orden en que se ejecutan los operadores en una expresión.

Ejemplo de prioridades:

1. Paréntesis ()
2. Exponentes **

3. Multiplicación y división *, /
4. Suma y resta +, -

Ejemplo:

```
resultado = 2 + 3 * 4 # Resultado: 14  
resultado = (2 + 3) * 4 # Resultado: 20
```

3.6 Elaboración de programas

La elaboración de un programa incluye las siguientes etapas:

1. **Análisis del problema:** Comprender lo que se necesita resolver.
2. **Diseño del algoritmo:** Crear una serie de pasos lógicos para resolver el problema.
3. **Codificación:** Traducir el algoritmo a un lenguaje de programación.
4. **Pruebas y optimización:** Verificar que el programa cumple su objetivo y mejora de su eficiencia.

Ejemplo: Diseñar un programa que calcule la suma de dos números.

```
# Entrada de datos  
num1 = int(input("Ingrese el primer número: "))  
num2 = int(input("Ingrese el segundo número: "))  
  
# Procesamiento  
suma = num1 + num2  
  
# Salida  
print(f"La suma es: {suma}")
```

3.10 Pruebas y depuración

Las pruebas y la depuración son etapas fundamentales para garantizar la calidad del software.

Tipos de pruebas:

1. **Pruebas unitarias:** Verifican componentes individuales.
2. **Pruebas de integración:** Aseguran que los módulos funcionan en conjunto.
3. **Pruebas de sistema:** Evalúan el programa completo.

Depuración:

Proceso de identificar y corregir errores en el código.

Ejemplo en Python:

```
def dividir(a, b):  
    try:
```

```
        return a / b
    except ZeroDivisionError:
        print("Error: División por cero no permitida.")

resultado = dividir(10, 0)
```

UNIDAD IV

ESTRUCTURAS SELECTIVAS Y DE REPETICIÓN

4.1 Selectiva simple

La selectiva simple es una estructura de control que evalúa una condición y, si esta es verdadera, ejecuta un bloque de código. Si la condición no se cumple, el programa simplemente continuo con el flujo siguiente.

Sintaxis general:

```
Si (condición) Entonces
    // Acciones a realizar si la condición es verdadera
Fin Si
```

Ejemplo:

Determinar si un número es positivo.

```
Escribir "Introduce un número:"
Leer numero

Si (numero > 0) Entonces
    Escribir "El número es positivo."
Fin Si
```

4.2 Selectiva doble

La selectiva doble permite evaluar una condición y ejecutar un bloque de código si esta es verdadera, o ejecutar otro bloque si es falsa.

Sintaxis general:

```
Si (condición) Entonces
    // Acciones si la condición es verdadera
Sino
    // Acciones si la condición es falsa
Fin Si
```

Ejemplo:

Verificar si una persona es mayor de edad.

```
Escribir "Introduce tu edad:"
```

Leer edad

```
Si (edad >= 18) Entonces
    Escribir "Eres mayor de edad."
Sino
    Escribir "Eres menor de edad."
Fin Si
```

4.3 Selectiva anidada

La selectiva anidada consiste en incluir una estructura selectiva dentro de otra, permitiendo evaluar condiciones más complejas.

Sintaxis general:

```
Si (condición1) Entonces
    Si (condición2) Entonces
        // Acciones si ambas condiciones son verdaderas
    Sino
        // Acciones si condición2 es falsa
    Fin Si
Sino
    // Acciones si condición1 es falsa
Fin Si
```

Ejemplo:

Determinar el rango de un número.

```
Escribir "Introduce un número:"
Leer numero

Si (numero > 0) Entonces
    Si (numero > 100) Entonces
        Escribir "El número es mayor que 100."
    Sino
        Escribir "El número es positivo pero menor o igual a 100."
    Fin Si
Sino
    Escribir "El número es negativo."
Fin Si
```

4.4 Selectiva múltiple

La selectiva múltiple permite evaluar varias opciones o casos de una variable y ejecutar el bloque correspondiente al caso que se cumpla.

Sintaxis general:

```
Segun (variable) Hacer
    Caso 1:
        // Acciones para el caso 1
    Caso 2:
        // Acciones para el caso 2
    ...
```

Caso Por Defecto:

// Acciones si no se cumple ningún caso anterior

Fin Segun

Ejemplo:

Determinar el día de la semana basado en un número.

Escribir "Introduce un número del 1 al 7:"

Leer dia

Segun (dia) Hacer

Caso 1:

Escribir "Lunes"

Caso 2:

Escribir "Martes"

Caso 3:

Escribir "Miércoles"

Caso 4:

Escribir "Jueves"

Caso 5:

Escribir "Viernes"

Caso 6:

Escribir "Sábado"

Caso 7:

Escribir "Domingo"

Caso Por Defecto:

Escribir "Día no válido."

Fin Segun

4.5 Repetir mientras

El bucle "Repetir mientras" ejecuta un bloque de código mientras la condición especificada sea verdadera.

Sintaxis general:

Mientras (condición) Hacer

// Acciones a realizar mientras la condición sea verdadera

Fin Mientras

Ejemplo:

Imprimir los números del 1 al 5.

contador <- 1

Mientras (contador <= 5) Hacer

Escribir contador

contador <- contador + 1

Fin Mientras

4.6 Repetir hasta

El bucle "Repetir hasta" ejecuta un bloque de código al menos una vez y se repite hasta que la condición se cumpla.

Sintaxis general:

```
Repetir
  // Acciones a realizar
Hasta Que (condición)
```

Ejemplo:

Solicitar un número mayor que 10.

```
Repetir
  Escribir "Introduce un número mayor que 10:"
  Leer numero
Hasta Que (numero > 10)
```

4.7 Repetir desde

El bucle "Repetir desde" o "Para" ejecuta un bloque de código un número definido de veces, incrementando o decrementando una variable de control.

Sintaxis general:

```
Para (variable <- inicio Hasta fin Incremento paso) Hacer
  // Acciones a realizar en cada iteración
Fin Para
```

Ejemplo:

Imprimir los números del 1 al 10.

```
Para (i <- 1 Hasta 10 Incremento 1) Hacer
  Escribir i
Fin Para
```

4.8 Elaboración de programas

La elaboración de programas consiste en combinar estructuras selectivas y repetitivas para resolver problemas más complejos mediante algoritmos bien diseñados.

Ejemplo: Programa para calcular el promedio de calificaciones.

```
Escribir "Introduce la cantidad de calificaciones:"
Leer cantidad
suma <- 0
```

```
Para (i <- 1 Hasta cantidad Incremento 1) Hacer
  Escribir "Introduce la calificación ", i, ":"
  Leer calificacion
  suma <- suma + calificacion
Fin Para
```

```
promedio <- suma / cantidad
```

Escribir "El promedio es: ", promedio

Si (promedio >= 6) Entonces

Escribir "Aprobado."

Sino

Escribir "Reprobado."

Fin Si

UNIDAD V

Arreglos y archivos.

5.1 Arreglos unidimensionales

Un arreglo unidimensional es una estructura de datos que almacena un conjunto de elementos del mismo tipo, organizados en una secuencia lineal. Cada elemento del arreglo se accede mediante un índice que comienza en cero.

Sintaxis general:

tipo nombreArreglo[tamaño]

Ejemplo:

Almacenar las calificaciones de cinco estudiantes.

calificaciones[5] <- [85, 90, 78, 92, 88]

Para (i <- 0 Hasta 4 Incremento 1) Hacer

Escribir "Calificación del estudiante ", i+1, ": ", calificaciones[i]

Fin Para

5.2 Arreglos bidimensionales y multidimensionales

Un arreglo bidimensional es una estructura que organiza los datos en filas y columnas, similar a una tabla. Los arreglos multidimensionales extienden este concepto a más dimensiones.

Sintaxis general:

tipo nombreArreglo[filas][columnas]

Ejemplo de arreglo bidimensional:

Almacenar las calificaciones de tres estudiantes en dos materias.

calificaciones[3][2] <- [[85, 90], [78, 92], [88, 79]]

```

Para (i <- 0 Hasta 2 Incremento 1) Hacer
    Escribir "Estudiante ", i+1, ":"
    Para (j <- 0 Hasta 1 Incremento 1) Hacer
        Escribir " Materia ", j+1, ": ", calificaciones[i][j]
    Fin Para
Fin Para

```

Ejemplo de arreglo tridimensional:

Almacenar las temperaturas registradas en tres ciudades durante dos semanas, con lecturas diarias.

```

temperaturas[3][2][7]
// Arreglo tridimensional: ciudades, semanas, días

```

5.3 Apertura, entrada-salida de datos, y cierre de archivos

El manejo de archivos permite leer y escribir datos de manera permanente en un almacenamiento externo, como un disco duro. Esto incluye las operaciones de apertura, lectura/escritura, y cierre del archivo.

Operaciones básicas:

1. **Apertura de archivo:** Se especifica el archivo y el modo de apertura (lectura, escritura, etc.).
2. **Entrada-salida de datos:** Se realizan las operaciones de lectura y escritura en el archivo.
3. **Cierre de archivo:** Se libera el archivo para evitar pérdida de datos.

Sintaxis general:

```

Abrir archivo con modo
Mientras (no fin del archivo) Hacer
    Leer/Escribir datos
Fin Mientras
Cerrar archivo

```

Ejemplo:

Guardar nombres en un archivo y luego leerlos.

```

// Escritura
Abrir archivo "nombres.txt" con modo "escritura"
Escribir archivo "Ana"
Escribir archivo "Juan"
Escribir archivo "Luis"
Cerrar archivo

```

```

// Lectura
Abrir archivo "nombres.txt" con modo "lectura"
Mientras (no fin del archivo) Hacer
    Leer nombre desde archivo
    Escribir "Nombre: ", nombre
Fin Mientras
Cerrar archivo

```

5.4 Elaboración de programas

La elaboración de programas con arreglos y archivos combina las estructuras previamente descritas para resolver problemas complejos. Esto incluye la organización de datos en arreglos, su almacenamiento en archivos, y la manipulación eficiente mediante bucles y estructuras de control.

Ejemplo: Programa para calcular promedios y guardar resultados.

Escribir "Introduce el número de estudiantes:"

Leer estudiantes

Escribir "Introduce el número de materias:"

Leer materias

calificaciones[estudiantes][materias]

Para (i <- 0 Hasta estudiantes-1 Incremento 1) Hacer

Escribir "Estudiante ", i+1

Para (j <- 0 Hasta materias-1 Incremento 1) Hacer

Escribir " Introduce la calificación de la materia ", j+1, ":"

Leer calificaciones[i][j]

Fin Para

Fin Para

Abrir archivo "promedios.txt" con modo "escritura"

Para (i <- 0 Hasta estudiantes-1 Incremento 1) Hacer

suma <- 0

Para (j <- 0 Hasta materias-1 Incremento 1) Hacer

suma <- suma + calificaciones[i][j]

Fin Para

promedio <- suma / materias

Escribir archivo "Estudiante ", i+1, ": ", promedio

Fin Para

Cerrar archivo

Escribir "Los promedios se han guardado en 'promedios.txt'"

6.1 Introducción

Las funciones son bloques de código diseñados para realizar tareas específicas. Permiten dividir un programa en partes más manejables y reutilizables, mejorando la legibilidad y facilitando el mantenimiento.

6.2 Funciones estándar

Las funciones estándar son proporcionadas por los lenguajes de programación para realizar operaciones comunes, como manejo de cadenas, cálculos matemáticos, y manipulación de archivos.

Ejemplo:

Uso de funciones matemáticas estándar.

```
numero <- -25  
Escribir "Valor absoluto: ", abs(numero)  
Escribir "Raíz cuadrada: ", sqrt(25)
```

6.3 Entrada y salida de datos

Las funciones de entrada y salida son fundamentales para la interacción con el usuario o con archivos externos. Permiten leer datos desde el teclado o archivos y mostrar información en pantalla o guardar resultados en archivos.

Ejemplo:

```
Escribir "Introduce tu nombre:"  
Leer nombre  
Escribir "Hola, ", nombre
```

UNIDAD VI

FUNCIONES.

6.4 Funciones definidas por el usuario

El usuario puede definir sus propias funciones para resolver problemas específicos, lo que fomenta la modularidad y la reutilización de código.

Sintaxis general:

```
Funcion nombreFuncion(parametros)  
  // Cuerpo de la función  
  Retornar valor  
Fin Funcion
```

Ejemplo:

Calcular el área de un círculo.

```
Funcion calcularArea(radio)  
  area <- 3.1416 * radio * radio  
  Retornar area  
Fin Funcion
```

```
Escribir "Introduce el radio del círculo:"  
Leer radio  
Escribir "El área es: ", calcularArea(radio)
```

6.5 Pase por valor

En el pase por valor, se envía una copia del valor de una variable a la función, por lo que los cambios realizados dentro de la función no afectan la variable original.

Ejemplo:

```
Funcion duplicar(numero)
  numero <- numero * 2
  Escribir "Dentro de la función: ", numero
Fin Funcion
```

```
valor <- 5
duplicar(valor)
Escribir "Fuera de la función: ", valor
```

6.6 Pase por referencia

En el pase por referencia, se envía la dirección de memoria de una variable a la función, permitiendo modificar su valor original.

Ejemplo:

```
Funcion duplicarPorReferencia(Referenciar numero)
  numero <- numero * 2
  Escribir "Dentro de la función: ", numero
Fin Funcion
```

```
valor <- 5
duplicarPorReferencia(valor)
Escribir "Fuera de la función: ", valor
```

6.7 Elaboración de programas

La combinación de funciones estándar, definidas por el usuario, y distintos métodos de paso de parámetros permite crear programas complejos y eficientes.

Ejemplo: Programa para calcular el promedio usando funciones.

```
Funcion calcularPromedio(arreglo, tamaño)
  suma <- 0
  Para (i <- 0 Hasta tamaño-1 Incremento 1) Hacer
    suma <- suma + arreglo[i]
  Fin Para
  Retornar suma / tamaño
Fin Funcion
```

```
Escribir "Introduce la cantidad de calificaciones:"
Leer n
calificaciones[n]
```

```
Para (i <- 0 Hasta n-1 Incremento 1) Hacer
```

```
Escribir "Introduce la calificación ", i+1, ":"  
Leer calificaciones[i]  
Fin Para
```

```
promedio <- calcularPromedio(calificaciones, n)  
Escribir "El promedio es: ", promedio
```

PRACTICAS

UNIDAD I

U.1 Practica 1 Introducción a la computación.

Parte A: Identificación y descripción de los componentes físicos

1. **Objetivo:** Familiarizarse con los componentes físicos de una computadora y entender su función.
2. **Material necesario:**
 - Una computadora de escritorio o portátil.
 - Acceso a las partes internas de la computadora (si es posible).
3. **Instrucciones:**
 - Desmontar la carcasa de la computadora (si es una PC de escritorio) y observar sus componentes internos.
 - Identificar los siguientes componentes:
 - CPU
 - Placa base (Motherboard)
 - Memoria RAM
 - Disco duro o SSD
 - Tarjeta gráfica
 - Fuentes de alimentación
 - Describir brevemente la función de cada uno.

Parte B: Configuración y uso del sistema operativo

1. **Objetivo:** Familiarizarse con el uso de un sistema operativo y entender su papel en el manejo de recursos de hardware.
2. **Material necesario:**
 - Una computadora con un sistema operativo instalado (Windows, macOS, Linux, etc.).
3. **Instrucciones:**
 - Encender la computadora e iniciar sesión en el sistema operativo.
 - Explorar las configuraciones del sistema operativo (panel de control, configuraciones del sistema, administración de dispositivos, etc.).
 - Identificar cómo se gestionan los dispositivos de entrada y salida.
 - Abrir el "Administrador de tareas" (en Windows) o "Monitor de actividad" (en macOS) y observar los procesos que se están ejecutando y cómo el sistema operativo gestiona la memoria y el CPU.

Parte C: Exploración de aplicaciones

1. **Objetivo:** Comprender cómo el software de aplicación interactúa con el hardware y el sistema operativo.
2. **Material necesario:**
 - Una computadora con varias aplicaciones instaladas (navegador web, procesador de textos, reproductor multimedia, etc.).
3. **Instrucciones:**
 - Abrir diferentes aplicaciones (por ejemplo, un navegador web, un editor de texto, y un reproductor de video).
 - Observar cómo el sistema operativo gestiona las aplicaciones y cómo se interactúa con los recursos del hardware.
 - Analizar el rendimiento del sistema al ejecutar múltiples aplicaciones simultáneamente (gestión de memoria, uso de CPU, etc.).

U.1 Practica 2 Elementos de Excel.

Parte A: Introducción a la interfaz de Excel

1. **Objetivo:** Familiarizarse con los elementos de la interfaz de Excel.
2. **Material necesario:**
 - Una computadora con Microsoft Excel instalado.
3. **Instrucciones:**
 - Abre Excel y observa la **Cinta de opciones**. Haz clic en cada una de las pestañas (Inicio, Insertar, Datos, etc.) y explora las herramientas disponibles en cada una.
 - En una nueva hoja de trabajo, escribe algunos datos simples en las celdas (por ejemplo, nombres, números, fechas).
 - Selecciona una celda y observa la **Barra de fórmulas**. Aquí puedes ver el contenido de la celda o ingresar datos manualmente.

Parte B: Manipulación de celdas, filas y columnas

1. **Objetivo:** Aprender a modificar el tamaño, color y formato de las celdas, filas y columnas.
2. **Material necesario:**
 - Una computadora con Microsoft Excel.
3. **Instrucciones:**
 - **Seleccionar celdas:** Haz clic en una celda para seleccionarla. Para seleccionar varias celdas, haz clic y arrastra o utiliza las teclas Shift y las flechas de dirección.
 - **Cambiar tamaño de filas y columnas:** Coloca el cursor entre los encabezados de las filas o columnas y arrástralo para cambiar el tamaño.
 - **Formato de celdas:** Usa la pestaña **Inicio** para cambiar el formato de las celdas (negrita, cursiva, color de fondo, bordes, tipo de letra, etc.).
 - **Insertar y eliminar filas/columnas:** Haz clic derecho sobre el encabezado de una fila o columna y selecciona **Insertar** o **Eliminar**.

Parte C: Introducción a las fórmulas y funciones

1. **Objetivo:** Aprender a usar fórmulas y funciones básicas para realizar cálculos.
2. **Material necesario:**

- Una computadora con Microsoft Excel.

3. Instrucciones:

- **Realizar una suma:** En una celda, escribe `=A1+A2` (asumiendo que tienes datos en las celdas A1 y A2). Esto sumará los valores de las celdas A1 y A2.
- **Usar la función SUMA:** En una celda vacía, escribe `=SUMA(A1:A5)` para sumar los valores de las celdas A1 a A5.
- **Usar la función PROMEDIO:** En una celda vacía, escribe `=PROMEDIO(A1:A5)` para obtener el promedio de los valores en las celdas A1 a A5.
- **Fórmula condicional con la función SI:** En una celda vacía, escribe `=SI(A1>100, "Mayor a 100", "Menor o igual a 100")` para mostrar un mensaje dependiendo del valor en la celda A1.

Parte D: Creación de gráficos

1. **Objetivo:** Aprender a crear gráficos para representar visualmente los datos.

2. Material necesario:

- Una computadora con Microsoft Excel.

3. Instrucciones:

- Selecciona un rango de datos (por ejemplo, valores en las celdas A1:A5 y B1:B5).
- Ve a la pestaña **Insertar** y selecciona el tipo de gráfico que desees (por ejemplo, un gráfico de barras, líneas o circular).
- Personaliza el gráfico con el título y las etiquetas del eje si es necesario.
- Modifica los colores y el formato del gráfico desde la pestaña **Diseño**.

U.1 Practica 3 Introducción a la computación.

Parte A: Grabar una Macro Sencilla

1. **Objetivo:** Crear una macro básica para automatizar un proceso simple, como dar formato a un conjunto de celdas.
2. **Material necesario:**
 - Una computadora con Microsoft Excel.
3. **Instrucciones:**
 1. Abre una nueva hoja de cálculo de Excel.
 2. Ve a la pestaña **Vista** y haz clic en **Grabar Macro**.
 3. En el cuadro de diálogo, asigna un nombre a la macro, por ejemplo, "FormatoCeldas". Si deseas, puedes asignar una tecla de acceso rápido para ejecutar la macro más fácilmente. Luego haz clic en **Aceptar**.
 4. Realiza algunas acciones en la hoja de cálculo:
 - i. Selecciona algunas celdas (por ejemplo, A1:A10).
 - ii. Cambia el formato de las celdas seleccionadas (por ejemplo, cambia el color de fondo, la fuente, el tamaño de la letra, etc.).
 5. Una vez que hayas terminado, ve nuevamente a la pestaña **Vista** y haz clic en **Detener Grabación**.
1. Ahora, para ejecutar la macro, ve a la pestaña **Vista**, haz clic en **Macros** y selecciona **Ver Macros**. Luego selecciona "FormatoCeldas" y haz clic en **Ejecutar**. Verás cómo se aplica el formato automáticamente a las celdas seleccionadas.

Parte B: Editar una Macro con VBA

1. **Objetivo:** Modificar una macro grabada utilizando el Editor de VBA para cambiar su comportamiento.
2. **Material necesario:**
 - Una computadora con Microsoft Excel.

Instrucciones:

1. Ve a la pestaña **Vista** y haz clic en **Macros > Ver Macros**.
2. Selecciona la macro que grabaste (por ejemplo, "FormatoCeldas") y haz clic en **Modificar**. Esto abrirá el Editor de VBA.
3. En el editor, podrás ver el código generado por Excel para la macro. El código básico se verá algo así:

Sub FormatoCeldas() Range("A1:A10").Select With Selection .Interior.Color = RGB(255, 255, 0) ' Color de fondo amarillo .Font.Name = "Arial" .Font.Size = 12 End With End Sub

4. Modifica el código según lo desees. Por ejemplo, si quieres cambiar el color de fondo a verde, cambia esta línea:

`.Interior.Color = RGB(0, 255, 0) ' Color de fondo verde`

- a. Guarda los cambios y cierra el Editor de VBA.

5. Vuelve a ejecutar la macro como en el paso anterior, y ahora verás que se aplica el nuevo formato.

Parte C: Crear una Macro con VBA desde Cero

1. **Objetivo:** Crear una macro personalizada desde el principio, escribiendo código VBA para realizar una tarea más compleja.
2. **Material necesario:**
 - Una computadora con Microsoft Excel.

Instrucciones:

1. Abre una nueva hoja de cálculo de Excel.
2. Ve a la pestaña **Desarrollador** (si no aparece, habilítala desde las opciones de Excel) y haz clic en **Visual Basic** para abrir el Editor de VBA.
3. En el Editor de VBA, selecciona **Insertar > Módulo** para crear un nuevo módulo.
4. Escribe el siguiente código para crear una macro que copie los datos de una columna a otra:

```
Sub CopiarDatos() ' Copiar datos de la columna A a la columna B Range("A1:A10").Copy Destination:=Range("B1") End Sub
```

5. Cierra el Editor de VBA y guarda el archivo como un **Libro de Excel habilitado para macros (.xlsm)**.
6. Para ejecutar la macro, ve a la pestaña **Vista**, selecciona **Macros > Ver Macros**, elige "CopiarDatos" y haz clic en **Ejecutar**. Verás cómo se copian los datos de la columna A a la columna B automáticamente.

U.1 Practica 4 Aplicaciones

En esta práctica, vamos a construir una hoja de cálculo de Excel para calcular el salario neto de los trabajadores de una empresa. Utilizaremos conceptos básicos de contabilidad y cálculo de nómina, como salario base, deducciones, bonos y el cálculo final de la nómina a pagar.

Elementos necesarios:

- Salario base de cada trabajador.
- Deducciones por impuestos y seguridad social.
- Bonificaciones (si aplica).
- Cálculo de salario neto.

2. Estructura de la Hoja de Cálculo

Paso 1: Crear la tabla básica de la nómina

Comienza creando una tabla con los siguientes encabezados:

ID Trabajador	Nombre	Salario Base	Bonificación	Deducciones	Total a Pagar (Neto)
1	Juan Pérez	\$1,500	\$200	\$150	\$1,550
2	Ana Gómez	\$1,200	\$100	\$120	\$1,180
3	Luis Martínez	\$1,800	\$250	\$200	\$1,850

Paso 2: Agregar las fórmulas

1. **Salario Base:** Ingresar los valores manualmente en la columna correspondiente para cada trabajador.
2. **Bonificación:** Puedes ingresar los valores manualmente o crear una fórmula para calcular una bonificación fija o un porcentaje del salario base.

Ejemplo: Si quieres una bonificación del 10% del salario base, usa la fórmula:

$=C2*0.10$

(donde C2 es la celda con el salario base).

3. **Deducciones:** Las deducciones pueden incluir impuestos y contribuciones a la seguridad social. Supón que las deducciones son un porcentaje del salario base. Por ejemplo, si las deducciones son del 10%, la fórmula será:

$=C2*0.10$

4. **Total a Pagar (Neto):** El cálculo del salario neto se realiza sumando la bonificación y restando las deducciones al salario base. La fórmula sería:

$=C2+D2-E2$

(donde C2 es el salario base, D2 es la bonificación y E2 son las deducciones).

Paso 3: Copiar las fórmulas para los demás trabajadores

Una vez que hayas configurado la fórmula para un trabajador, puedes copiarla hacia abajo para todos los demás trabajadores de la lista.

3. Ejemplo Completo

Aquí tienes un ejemplo de cómo quedaría la tabla con fórmulas:

ID Trabajador	Nombre	Salario Base	Bonificación (10%)	Deducciones (10%)	Total a Pagar (Neto)
1	Juan Pérez	\$1,500	\$150	\$150	$\$1,500 + \$150 - \$150 = \$1,500$
2	Ana Gómez	\$1,200	\$120	\$120	$\$1,200 + \$120 - \$120 = \$1,200$
3	Luis Martínez	\$1,800	\$180	\$180	$\$1,800 + \$180 - \$180 = \$1,800$

4. Agregar Elementos Adicionales

Paso 4: Agregar una Columna de “Horas Extras”

Si se desea incluir horas extras, puedes agregar una nueva columna para registrar las horas extras trabajadas y su valor (por ejemplo, 1.5 veces el salario por hora).

1. **Horas Extras Trabajadas:** Agrega una columna para registrar las horas extras trabajadas por cada empleado.
2. **Pago de Horas Extras:** Supón que el valor de la hora extra es 1.5 veces el salario por hora. Si un trabajador tiene un salario base de \$1,500 mensuales y trabaja 10 horas extras, calcula el valor de las horas extras de la siguiente manera:

- Calcula el salario por hora:

$=C2/160$ ' Suponiendo 160 horas al mes

- Luego, calcula el pago por horas extras:

$=F2*1.5$

Donde F2 es la celda con el salario por hora.

3. **Ajuste del Total Para Pagar:** Agrega el pago por horas extras al total a pagar. La fórmula del total a pagar sería:

$=C2+D2-E2+G2$

Donde G2 es la celda con el valor de las horas extras.

5. Ejemplo Final con Horas Extras

ID Trabajador	Nombre	Salario Base	Bonificación (10%)	Deducciones (10%)	Horas Extras	Pago Horas Extras	Total a Pagar (Neto)
1	Juan Pérez	\$1,500	\$150	\$150	10	\$93.75	$\$1,500 + \$150 - \$150 + \$93.75 = \$1,593.75$
2	Ana Gómez	\$1,200	\$120	\$120	5	\$46.88	$\$1,200 + \$120 - \$120 + \$46.88 = \$1,246.88$
3	Luis Martínez	\$1,800	\$180	\$180	8	\$75.00	$\$1,800 + \$180 - \$180 + \$75.00 = \$1,875.00$

UNIDAD II

U.2 Practica 1 Metodología para el diseño de software

Descripción del Proyecto: Vamos a diseñar un **Sistema de Gestión de Tareas** para un equipo de trabajo. El sistema permitirá a los usuarios agregar, editar, eliminar y completar tareas, asignarlas a diferentes miembros del equipo y visualizar el progreso de las tareas.

Fase 1: Planeación y Requerimientos

1. Objetivo del Sistema:

- Proporcionar una herramienta para que los miembros del equipo puedan gestionar y hacer seguimiento de sus tareas.

2. Requerimientos Funcionales:

- El sistema debe permitir la creación de tareas con campos como título, descripción, fecha de vencimiento y prioridad.
- Debe permitir la asignación de tareas a miembros del equipo.
- Debe permitir marcar las tareas como completadas.
- El sistema debe mostrar un listado de tareas con la opción de filtrarlas por estado (pendientes, completadas, en progreso).

3. Requerimientos No Funcionales:

- El sistema debe ser fácil de usar y tener una interfaz clara.

U.2 Practica 2 Diagramas de Flujo

Ejercicio 1: Calcular el Área de un Círculo

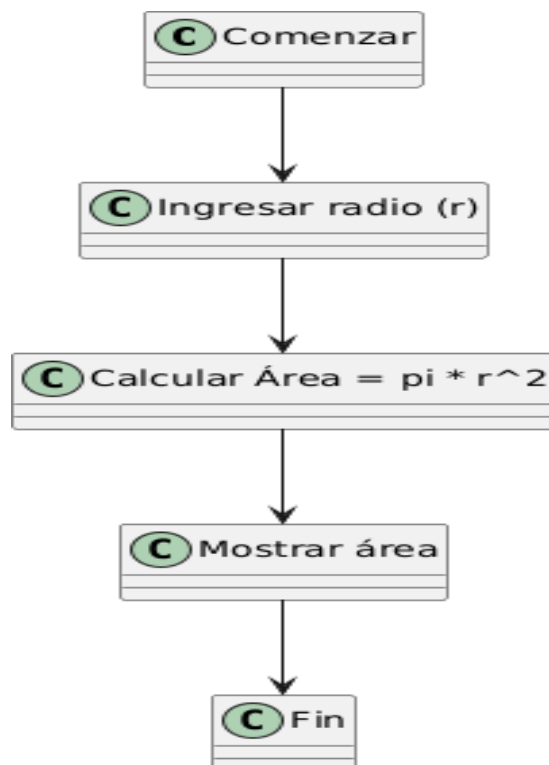
Descripción: Crear un diagrama de flujo para calcular el área de un círculo dado el valor del radio.

Fórmula del área del círculo:
 $A'rea = \pi \times r^2$

Pasos para el diagrama de flujo:

1. Comenzar.
2. Solicitar al usuario el valor del radio.
3. Calcular el área utilizando la fórmula mencionada.
4. Mostrar el resultado del área.
5. Fin.

Diagrama de Flujo:



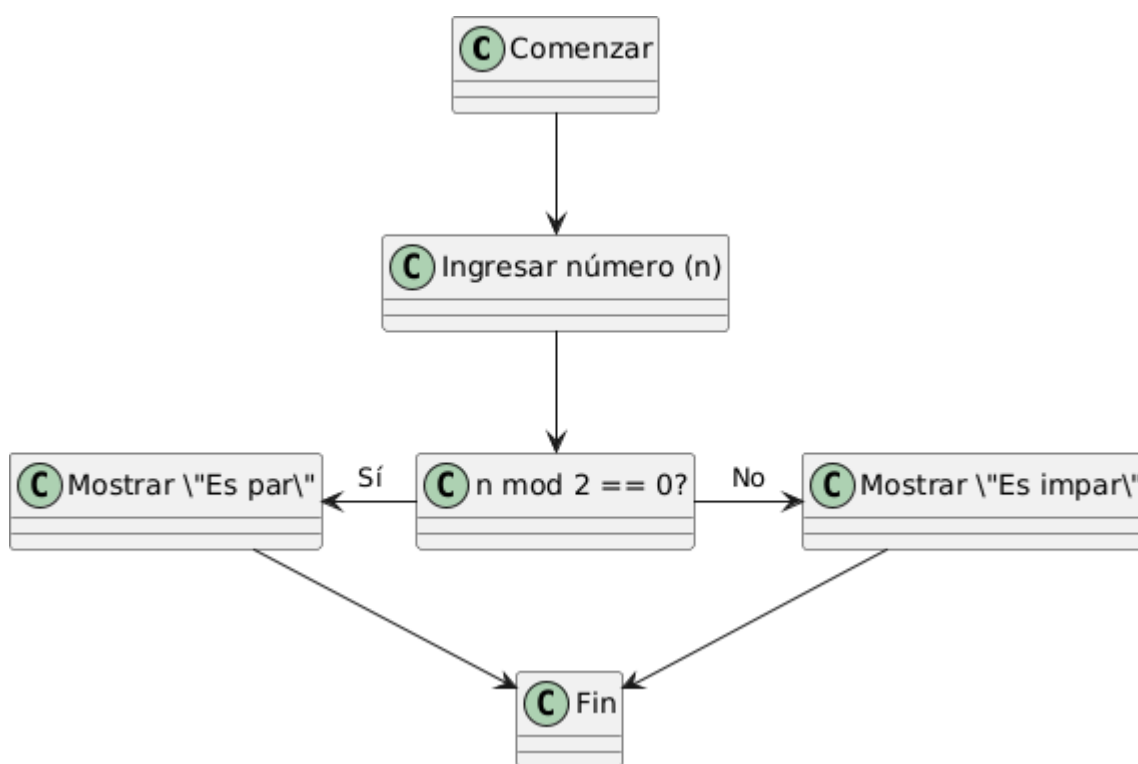
Ejercicio 2: Verificar si un Número es Par o Impar

Descripción: Crear un diagrama de flujo que verifique si un número es par o impar.

Pasos para el diagrama de flujo:

1. Comenzar.
2. Solicitar un número al usuario.
3. Comprobar si el número es divisible entre 2 (número mod 2).
4. Si es divisible entre 2, mostrar "Es par".
5. Si no es divisible entre 2, mostrar "Es impar".
6. Fin.

Diagrama de Flujo:



Ejercicio 3: Calcular el Promedio de Tres Números

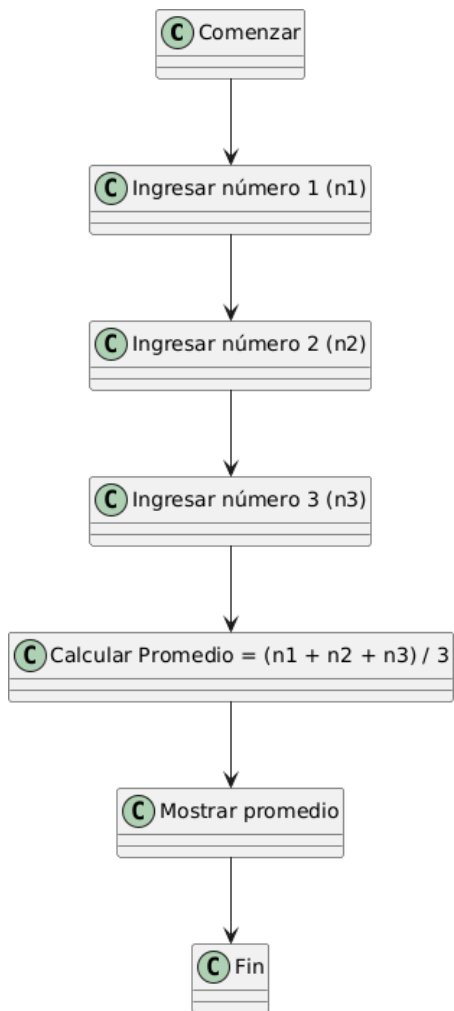
Descripción: Crear un diagrama de flujo para calcular el promedio de tres números introducidos por el usuario.

Fórmula del promedio: $\text{Promedio} = \frac{n1 + n2 + n3}{3}$

Pasos para el diagrama de flujo:

1. Comenzar.
2. Solicitar los tres números al usuario.
3. Calcular el promedio utilizando la fórmula.
4. Mostrar el resultado del promedio.
5. Fin.

Diagrama de Flujo:



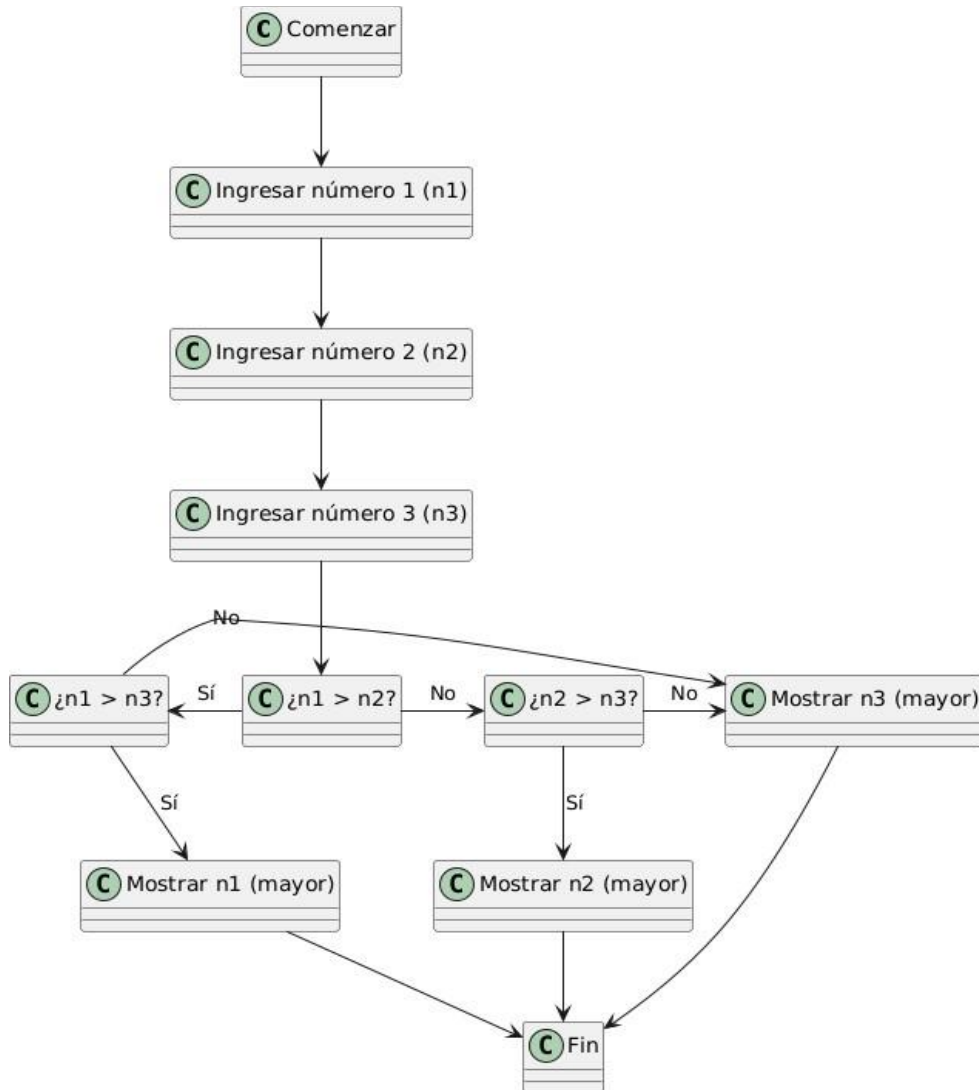
Ejercicio 4: Encontrar el Número Mayor de Tres Números

Descripción: Crear un diagrama de flujo que determine el número mayor entre tres números introducidos por el usuario.

Pasos para el diagrama de flujo:

1. Comenzar.
2. Solicitar tres números al usuario.
3. Comparar el primer número con el segundo.
4. Si el primer número es mayor, compararlo con el tercero.
5. Si el primer número es mayor que ambos, es el mayor.
6. Si el segundo número es mayor que el primero, compararlo con el tercero.
7. Si el segundo número es mayor que ambos, es el mayor.
8. Si el tercero es mayor que los otros dos, es el mayor.
9. Mostrar el número mayor.
10. Fin.

Diagrama de Flujo:



Ejercicio 5: Sumar los N Primeros Números Naturales

Descripción: Crear un diagrama de flujo que calcule la suma de los n primeros números naturales (del 1 al n), donde n es ingresado por el usuario.

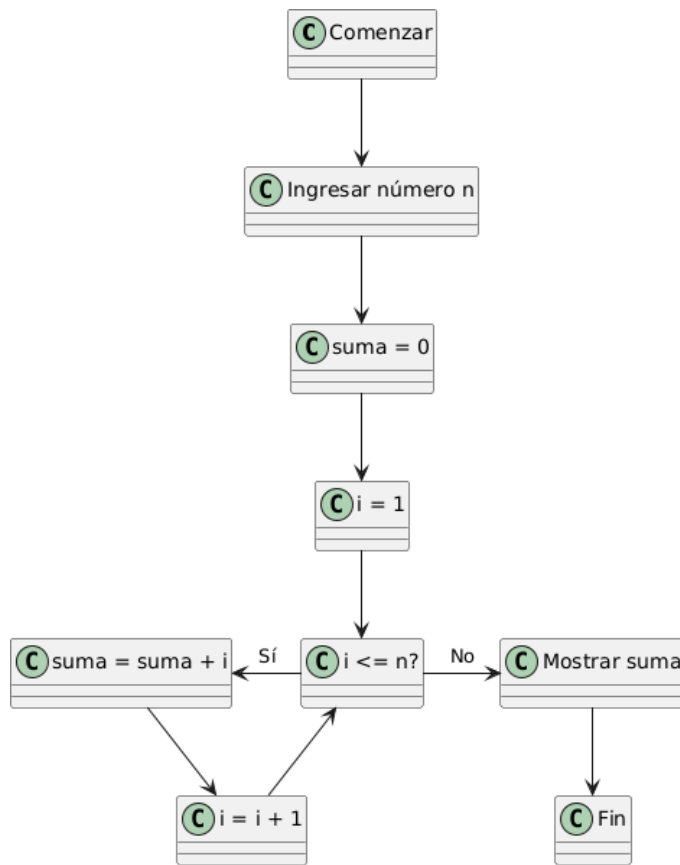
Fórmula de la suma: $\text{Suma} = 1 + 2 + 3 + \dots + n$

Pasos para el diagrama de flujo:

1. Comenzar.
2. Solicitar el número n al usuario.
3. Inicializar una variable suma a 0.
4. Crear un ciclo que vaya desde 1 hasta n , sumando cada número a suma.
5. Mostrar el resultado de la suma.

6. Fin.

Diagrama de Flujo:



U.2 Practica 3 Pseudocódigo

Ejercicio práctico de un pseudocódigo para resolver un problema típico: calcular el promedio de n números ingresados por el usuario.

Descripción del Problema:

El programa debe:

1. Pedir al usuario la cantidad de números a ingresar (n).
2. Solicitar cada uno de los n números.
3. Calcular la suma de los números.
4. Determinar el promedio dividiendo la suma entre n .

5. Mostrar el promedio al usuario.

Pseudocódigo:

```
1  INICIO
2
3      IMPRIMIR  " Ingrese la cantidad de números (n):"
4      LEER  n
5
6      SI  n <= 0  ENTONCES
7          IMPRIMIR  " La cantidad debe ser mayor a 0."
8          SALIR
9      FIN_SI
10
11     suma ← 0
12
13     PARA i ← 1 HASTA n HACER
14         IMPRIMIR  " Ingrese el número" , i , " : "
15         LEER  numero
16         suma ← suma + numero
17     FIN_PARA
18
19     promedio ← suma / n
20
21     IMPRIMIR  " El promedio de los" , n , " números es: " , promedio
22
```

U.2 Practica 4 Implementación de algoritmos

Crea un programa que calcule el factorial de un número n ingresado por el usuario.

Descripción del Problema

El **factorial** de un número n se define como:

$$n! = n \times (n-1) \times (n-2) \times \dots \times 1 \quad n! = n \times (n-1) \times (n-2) \times \dots \times 1$$

Por ejemplo:

- $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$ $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
- $0! = 10! = 1$ (por definición).

El programa debe cumplir los siguientes pasos:

1. Pedir al usuario que ingrese un número entero positivo.
2. Validar que el número ingresado sea positivo.
3. Calcular el factorial usando un bucle.
4. Mostrar el resultado del factorial.

Pseudocódigo

```
1  INICIO
2
3      IMPRIMIR  " Ingrese la cantidad INICIO
4
5  IMPRIMIR      " Ingrese un número entero positivo ( n ) : "
6      LEER n
7
8      SI n < 0 ENTONCES
9          IMPRIMIR " El número debe ser positivo " .
10         SALIR
11     FIN_SI
12
13     factorial ← 1
14
15     PARA i ← 1 HASTA n HACER
16         factorial ← factorial * i
17     FIN_PARA
18
19     IMPRIMIR " El factorial de " , n, " es : " , factorial
20
```

Implementación en Python

main.py



```
1 # Solicitar al usuario un número entero positivo
2 n = int(input("Ingrese un número entero positivo: "))
3
4 # Validar si el número es positivo
5 if n < 0:
6     print("El número debe ser positivo.")
7 else:
8     # Inicializar el factorial en 1
9     factorial = 1
10
11     # Calcular el factorial
12 for i in range(1, n + 1):
13     factorial *= i
14
15     # Mostrar el resultado
16 print(f"El factorial de {n} es: {factorial}")
17
```

Ln: 17, Col: 1



Run



Share

Command Line Arguments



Ingrese un número entero positivo:



26



El factorial de 26 es: 403291461126605635584000000



** Process exited - Return Code: 0 **



Press Enter to exit terminal

UNIDAD III

U.3 Practica 1: Conversión de temperatura

Descripción:

Escribe un programa que convierta una temperatura dada en grados Celsius a grados Fahrenheit.

Detalles del ejercicio:

1. **Entrada:** Pedir al usuario la temperatura en grados Celsius.
2. **Procesamiento:** Convertir la temperatura usando la fórmula: $F = C \times 9/5 + 32$
3. **Salida:** Mostrar la temperatura en grados Fahrenheit.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 celsius = float(input("Introduce la temperatura en grados Celsius: "))
3
4 # Procesamiento
5 fahrenheit = celsius * (9/5) + 32
6
7 # Salida
8 print(f"La temperatura en grados Fahrenheit es: {fahrenheit}")
9
```

Ln: 9, Col: 1

[Run](#) [Share](#) Command Line Arguments

```
Introduce la temperatura en grados Celsius:
20
La temperatura en grados Fahrenheit es: 68.0

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

U.3 Practica 2: Cálculo del promedio

Descripción:

Haz un programa que calcule el promedio de tres números introducidos por el usuario.

Detalles del ejercicio:

1. **Entrada:** Solicitar tres números.
2. **Procesamiento:** Calcular el promedio de los números:
$$\text{Promedio} = \frac{n1 + n2 + n3}{3}$$
3. **Salida:** Mostrar el promedio calculado.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 n1 = float(input("Introduce el primer número: "))
3 n2 = float(input("Introduce el segundo número: "))
4 n3 = float(input("Introduce el tercer número: "))
5
6 # Procesamiento
7 promedio = (n1 + n2 + n3) / 3
8
9 # Salida
10 print(f"El promedio de los tres números es: {promedio}")
11
```

Ln: 11, Col: 1

[Run](#) [Share](#) Command Line Arguments

```
Introduce el primer número:
1
Introduce el segundo número:
6
Introduce el tercer número:
10
El promedio de los tres números es: 5.666666666666667
```

Ejemplo 3: Verificación de números pares

Descripción:

Crea un programa que determine si un número introducido por el usuario es par o impar.

Detalles del ejercicio:

1. **Entrada:** Solicitar un número entero.
2. **Procesamiento:** Determinar si el número es divisible entre 2: $\text{par} \leftrightarrow (\text{número} \bmod 2 == 0)$ $\text{par} \leftrightarrow (\text{número} \bmod 2 \neq 0)$
3. **Salida:** Mostrar un mensaje indicando si el número es par o impar.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 numero = int(input("Introduce un número entero: "))
3
4 # Procesamiento
5 if numero % 2 == 0:
6     resultado = "par"
7 else:
8     resultado = "impar"
9
10 # Salida
11 print(f"El número {numero} es {resultado}.")
12
```

Ln: 12, Col: 1

 Run  Share Command Line Arguments

```
 Introduce un número entero:
 25
 El número 25 es impar.
>_
 ** Process exited - Return Code: 0 **
Press Enter to exit terminal
|
```

U.3 Practica 4: Calculadora de descuento

Descripción:

Haz un programa que calcule el precio final de un producto después de aplicar un descuento.

Detalles del ejercicio:

1. **Entrada:** Solicitar el precio original y el porcentaje de descuento.
2. **Procesamiento:** Calcular el precio final usando la fórmula: $\text{Precio final} = \text{Precio original} \times (1 - \text{Descuento} / 100)$
3. **Salida:** Mostrar el precio final.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 precio_original = float(input("Introduce el precio original del producto: "))
3 descuento = float(input("Introduce el porcentaje de descuento: "))
4
5 # Procesamiento
6 precio_final = precio_original * (1 - descuento / 100)
7
8 # Salida
9 print(f"El precio final después del descuento es: {precio_final}")
10
```

Ln: 10, Col: 1

 Run  Share Command Line Arguments

```
 Introduce el precio original del producto:
120
 Introduce el porcentaje de descuento:
20
 El precio final después del descuento es: 96.0





















































































































































































































































































































































































































```

U.3 Practica 5: Conversión de tiempo

Descripción:

Crea un programa que convierta una cantidad de minutos en horas y minutos.

Detalles del ejercicio:

1. **Entrada:** Solicitar el número total de minutos.
2. **Procesamiento:** Calcular cuántas horas completas hay y cuántos minutos sobran:
 - Horas: $\text{Horas} = \text{Minutos totales} // 60$
 - Minutos restantes: $\text{Minutos restantes} = \text{Minutos totales} \% 60$
3. **Salida:** Mostrar las horas y minutos resultantes.

Código de ejemplo (Python):

```
main.py +
1  # Entrada
2  minutos_totales = int(input("Introduce el número total de minutos: "))
3
4  # Procesamiento
5  horas = minutos_totales // 60
6  minutos_restantes = minutos_totales % 60
7
8  # Salida
9  print(f"{minutos_totales} minutos son {horas} horas y {minutos_restantes} minutos.")
10
```

Ln: 10, Col: 1

 Run  Share Command Line Arguments

```
 Introduce el número total de minutos:
258
258 minutos son 4 horas y 18 minutos.
>_
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

U.3 Practica 6: Comparación de números

Descripción:

Escribe un programa que compare dos números y determine cuál es mayor, menor o si son iguales.

Detalles del ejercicio:

1. **Entrada:** Pedir al usuario dos números.
2. **Procesamiento:** Comparar los dos números usando estructuras condicionales (if, elif, else).
3. **Salida:** Mostrar el resultado de la comparación.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 numero1 = float(input("Introduce el primer número: "))
3 numero2 = float(input("Introduce el segundo número: "))
4
5 # Procesamiento y salida
6 if numero1 > numero2:
7     print(f"El número {numero1} es mayor que {numero2}.")
8 elif numero1 < numero2:
9     print(f"El número {numero2} es mayor que {numero1}.")
10 else:
11     print("Ambos números son iguales.")
12
```

Ln: 12, Col: 1

 Run  Share Command Line Arguments

```
 Introduce el primer número:
1
Introduce el segundo número:
6
> El número 6.0 es mayor que 1.0.


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

UNIDAD IV

U.4 Practica 5 Estructuras selectivas y de repetición.

U.4 Practica 1 Búsqueda Lineal

Busca un elemento en una lista dada.

```
main.py +
1 def busqueda_lineal(lista, objetivo):
2     for i in range(len(lista)):
3         if lista[i] == objetivo:
4             return f"Elemento encontrado en la posición {i}"
5     return "Elemento no encontrado"
6
7 # Ejemplo
8 lista = [4, 2, 7, 1, 9, 3]
9 objetivo = 7
10 print(busqueda_lineal(lista, objetivo))
11
```

Ln: 9, Col: 13

 Run  Share Command Line Arguments

```

❏ Elemento encontrado en la posición 2
⬇
🏠 ** Process exited - Return Code: 0 **
>_ Press Enter to exit terminal
↕
```

U.4 Practica 2 Ordenamiento por Selección

Ordena una lista en orden ascendente.

```
main.py +
1 def ordenamiento_seleccion(lista):
2     for i in range(len(lista)):
3         min_index = i
4         for j in range(i + 1, len(lista)):
5             if lista[j] < lista[min_index]:
6                 min_index = j
7         lista[i], lista[min_index] = lista[min_index], lista[i]
8     return lista
9
10 # Ejemplo
11 lista = [29, 10, 14, 37, 13]
12 print("Lista ordenada:", ordenamiento_seleccion(lista))
13
```

Ln: 13, Col: 1

 Run  Share Command Line Arguments

```
 Lista ordenada: [10, 13, 14, 29, 37]

 ** Process exited - Return Code: 0 **
> Press Enter to exit terminal

```

U.4 Practica 3 Números Primos hasta un Límite

Encuentra todos los números primos menores o iguales a n .

```
main.py +
1 def numeros_primos(hasta):
2     primos = []
3     for num in range(2, hasta + 1):
4         es_primo = True
5         for divisor in range(2, int(num**0.5) + 1):
6             if num % divisor == 0:
7                 es_primo = False
8                 break
9         if es_primo:
10             primos.append(num)
11     return primos
12
13 # Ejemplo
14 limite = 100
15 print(f"Números primos hasta {limite}:", numeros_primos(limite))
16
```

Ln: 16, Col: 1

[Run](#) [Share](#) Command Line Arguments

Números primos hasta 100: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

** Process exited - Return Code: 0 **

Press Enter to exit terminal

U.4 Practica 4 Verificar un Palíndromo

Comprueba si una palabra o frase es un palíndromo.

```
main.py +
1 def es_palindromo(cadena):
2     cadena = cadena.replace(" ", "").lower() # Eliminar espacios y convertir a minúsculas
3     return cadena == cadena[::-1]
4
5 # Ejemplo
6 frase = "Anita lava la tina"
7 print(f"La frase '{frase}' es un palíndromo:", es_palindromo(frase))
8
```

Ln: 8, Col: 1

 Run  Share Command Line Arguments

```
La frase 'Anita lava la tina' es un palíndromo: True

** Process exited - Return Code: 0 **

>_ Press Enter to exit terminal
|
```

U.4 Practica 5 Validación de Contraseña Fuerte

Comprueba si una contraseña cumple con los requisitos de seguridad.

```
main.py +
1 def es_contraseña_fuerte(contraseña):
2     tiene_mayuscula = any(c.isupper() for c in contraseña)
3     tiene_minuscula = any(c.islower() for c in contraseña)
4     tiene_numero = any(c.isdigit() for c in contraseña)
5     tiene_longitud = len(contraseña) >= 8
6     return tiene_mayuscula and tiene_minuscula and tiene_numero and tiene_longitud
7
8 # Ejemplo
9 contraseña = "guillermog2024"
10 print(f"La contraseña '{contraseña}' es fuerte:", es_contraseña_fuerte(contraseña))
11
```

Ln: 11, Col: 1

 Run  Share Command Line Arguments

```
La contraseña 'guillermog2024' es fuerte: False
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

UNIDAD V

U.5 Practica 1: Promedio de números almacenados en un arreglo

Descripción:

Crea un programa que almacene varios números en un arreglo, calcule su promedio y lo muestre.

Detalles del ejercicio:

1. **Entrada:** Solicitar al usuario que ingrese 5 números.
2. **Procesamiento:** Guardar los números en una lista, calcular la suma y el promedio.
3. **Salida:** Mostrar el promedio.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 numeros = []
3 for i in range(5):
4     num = float(input(f"Introduce el número {i+1}: "))
5     numeros.append(num)
6
7 # Procesamiento
8 promedio = sum(numeros) / len(numeros)
9
10 # Salida
11 print(f"El promedio de los números es: {promedio}")
12
```

Ln: 12, Col: 1

 Run  Share Command Line Arguments

```
Introduce el número 1:
25
Introduce el número 2:
36
Introduce el número 3:
3
Introduce el número 4:
5
Introduce el número 5:
6
El promedio de los números es: 15.0
```

U.5 Practica 2: Buscar un valor en un arreglo

Descripción:

Haz un programa que permita al usuario buscar un valor específico dentro de un arreglo y diga si está o no.

Detalles del ejercicio:

1. **Entrada:** Solicitar al usuario 5 números y un valor a buscar.
2. **Procesamiento:** Verificar si el valor está en la lista usando el operador in.
3. **Salida:** Mostrar si el valor fue encontrado.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 numeros = []
3 for i in range(5):
4     num = int(input(f"Introduce el número {i+1}: "))
5     numeros.append(num)
6
7 buscar = int(input("Introduce el número que quieres buscar: "))
8
9 # Procesamiento y salida
10 if buscar in numeros:
11     print(f"El número {buscar} se encuentra en la lista.")
12 else:
13     print(f"El número {buscar} no está en la lista.")
14
```

Ln: 14, Col: 1

 Run  Share Command Line Arguments

```
-----
2
Introduce el número 4:
5
Introduce el número 5:
8
Introduce el número que quieres buscar:
5
El número 5 se encuentra en la lista.
```

U.5 Practica 3: Guardar nombres en un archivo

Descripción:

Escribe un programa que solicite 5 nombres al usuario, los guarde en un arreglo y luego los escriba en un archivo llamado nombres.txt.

Detalles del ejercicio:

1. **Entrada:** Solicitar 5 nombres.
2. **Procesamiento:** Guardar los nombres en una lista y escribirlos en un archivo.
3. **Salida:** Archivo nombres.txt con los nombres.

Código de ejemplo (Python):

```
main.py +
1 # Entrada
2 nombres = []
3 for i in range(5):
4     nombre = input(f"Introduce el nombre {i+1}: ")
5     nombres.append(nombre)
6
7 # Procesamiento: Guardar en un archivo
8 with open("nombres.txt", "w") as archivo:
9     for nombre in nombres:
10         archivo.write(nombre + "\n")
11
12 # Salida
13 print("Los nombres han sido guardados en el archivo 'nombres.txt'.")
14
```

Ln: 14, Col: 1

[Run](#) [Share](#) Command Line Arguments

```
Introduce el nombre 1:
2
Introduce el nombre 2:
3
Introduce el nombre 3:
5
Introduce el nombre 4:
7
Introduce el nombre 5:
6
Los nombres han sido guardados en el archivo 'nombres.txt'.
```

U.5 Practica 4: Leer nombres desde un archivo y ordenarlos

Descripción:

Crea un programa que lea una lista de nombres desde un archivo, los almacene en un arreglo, los ordene alfabéticamente y los muestre.

Contenido del archivo nombres.txt:

Carlos Ana Luis Beatriz David

Código de ejemplo (Python):

```
main.py +
1 # Apertura y lectura del archivo
2 with open("nombres.txt", "r") as archivo:
3     nombres = [linea.strip() for linea in archivo]
4
5 # Procesamiento: Ordenar alfabéticamente
6 nombres.sort()
7
8 # Salida
9 print("Nombres ordenados alfabéticamente:")
10 for nombre in nombres:
11     print(nombre)
12
```

```
nombres: Bloc de notas
Archivo Edición Formato Ver Ayuda
Carlos
Ana
Luis
Beatriz
David
```

U.5 Practica 5: Leer y analizar un archivo CSV

Descripción:

Lee un archivo CSV que contenga nombres y edades, almacénalos en listas separadas, y muestra el promedio de las edades.

Contenido del archivo personas.csv:

Nombre,Edad Carlos,30 Ana,25 Luis,35 Beatriz,28 David,40

Código de ejemplo (Python):

```
main.py +
1 # Apertura y lectura del archivo
2 with open("personas.csv", "r") as archivo:
3     lineas = archivo.readlines()
4
5 # Procesamiento
6 nombres = []
7 edades = []
8
9 for linea in lineas[1:]: # Omitir la cabecera
10     nombre, edad = linea.strip().split(",")
11     nombres.append(nombre)
12     edades.append(int(edad))
13
14 promedio_edad = sum(edades) / len(edades)
15
16 # Salida
17 print("Nombres:", nombres)
18 print("Promedio de edades:", promedio_edad)
19
```

A1	Nombre	
A	B	C
1	Nombre	Edad
2	Carlos	30
3	Ana	25
4	Luis	35
5	Beatriz	28
6	David	40
7		
8		

UNIDAD 6

U.6 Practica 1: Simulador de cajero automático

1. Funciones:

- mostrar_menu: Muestra las opciones disponibles.
- consultar_saldo: Permite consultar el saldo actual.
- retirar_dinero: Permite retirar una cantidad de dinero, validando fondos disponibles.
- depositar_dinero: Permite depositar dinero en la cuenta.

2. Programa principal:

- Ejecuta un bucle para que el usuario interactúe con el cajero hasta que decida salir.

Código completo

```
# Saldo inicial
saldo = 1000.0 # Saldo predeterminado

# Función para mostrar el menú
def mostrar_menu():
    print("\n--- Menú del Cajero Automático ---")
    print("1. Consultar saldo")
    print("2. Retirar dinero")
    print("3. Depositar dinero")
    print("4. Salir")

# Función para consultar el saldo
def consultar_saldo():
    print(f"\nTu saldo actual es: ${saldo:.2f}")

# Función para retirar dinero
def retirar_dinero():
    global saldo
    monto = float(input("¿Cuánto deseas retirar? $"))
    if monto > saldo:
        print("Fondos insuficientes. No se puede realizar la operación.")
    elif monto <= 0:
        print("Por favor, introduce una cantidad válida.")
    else:
        saldo -= monto
```

```

    print(f'Has retirado ${monto:.2f}. Tu nuevo saldo es: ${saldo:.2f}')

# Función para depositar dinero
def depositar_dinero():
    global saldo
    monto = float(input("¿Cuánto deseas depositar? $"))
    if monto <= 0:
        print("Por favor, introduce una cantidad válida.")
    else:
        saldo += monto
        print(f'Has depositado ${monto:.2f}. Tu nuevo saldo es: ${saldo:.2f}')

# Programa principal
while True:
    mostrar_menu()
    opcion = input("Selecciona una opción: ")

    if opcion == "1":
        consultar_saldo()
    elif opcion == "2":
        retirar_dinero()

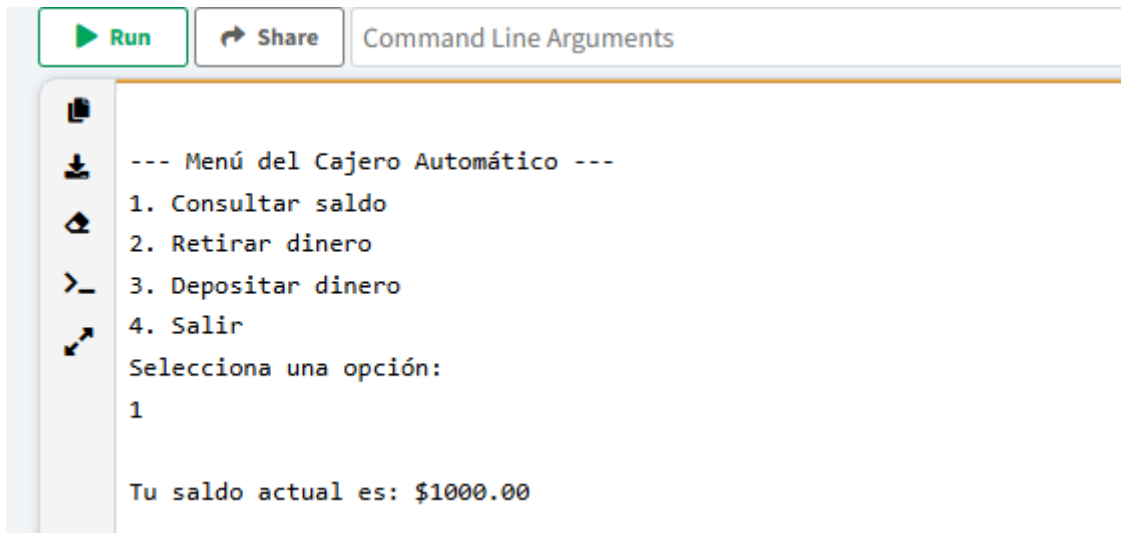
```

Explicación del funcionamiento

1. **Saldo inicial:** Se inicia el saldo en 1000.0.
2. **Funciones:**
 - Cada operación tiene su propia función. Esto facilita la modularidad y la reutilización del código.
3. **Menú interactivo:**
 - El menú se muestra en cada iteración del bucle.
 - El usuario elige una opción ingresando un número del 1 al 4.
4. **Validaciones:**
 - Se valida que el usuario no retire más dinero del disponible.
 - Se verifica que los montos de retiro y depósito sean positivos.
5. **Salir del programa:** Si el usuario elige la opción 4, el programa se detiene con un mensaje de despedida.

Ejecución y Ejemplo de Interacción

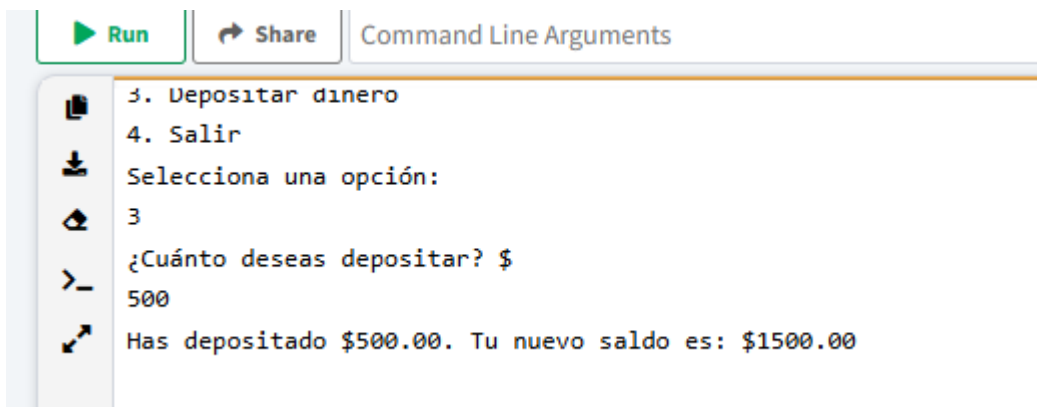
Entrada del usuario:



The screenshot shows a terminal window with a title bar containing 'Run', 'Share', and 'Command Line Arguments'. On the left is a sidebar with icons for file operations. The terminal text is as follows:

```
--- Menú del Cajero Automático ---  
1. Consultar saldo  
2. Retirar dinero  
3. Depositar dinero  
4. Salir  
Selecciona una opción:  
1  
  
Tu saldo actual es: $1000.00
```

Salida:



The screenshot shows a terminal window with a title bar containing 'Run', 'Share', and 'Command Line Arguments'. On the left is a sidebar with icons for file operations. The terminal text is as follows:

```
3. Depositar dinero  
4. Salir  
Selecciona una opción:  
3  
¿Cuánto deseas depositar? $  
500  
Has depositado $500.00. Tu nuevo saldo es: $1500.00
```

Flujo completo:

1. El usuario consulta el saldo: \$1000.00.
2. Deposita \$500:
 - Nuevo saldo: \$1500.00.
3. Retira \$200:
 - Nuevo saldo: \$1300.00.
4. Intenta retirar \$2000 (error por falta de fondos).

5. Sale del cajero.

U.6 Practica 2: Pago de nomina a empleados

1. Funciones:

- `calcular_pago_bruto`: Calcula el pago bruto con base en el salario por hora y las horas trabajadas.
- `calcular_deducciones`: Calcula las deducciones aplicadas al salario bruto.
- `calcular_bonificaciones`: Calcula las bonificaciones adicionales, si las hay.
- `calcular_pago_netto`: Calcula el pago neto (pago bruto - deducciones + bonificaciones).

2. Programa principal:

- Solicita información del empleado.
- Calcula y muestra el desglose de la nómina.

Código completo

Función para calcular el pago bruto

```
def calcular_pago_bruto(salario_hora, horas_trabajadas):
```

```
    return salario_hora * horas_trabajadas
```

Función para calcular deducciones

```
def calcular_deducciones(pago_bruto, porcentaje_impuestos):
```

```
    return pago_bruto * (porcentaje_impuestos / 100)
```

Función para calcular bonificaciones

```
def calcular_bonificaciones(bonificacion_fija):
```

```
    return bonificacion_fija
```

Función para calcular el pago neto

```
def calcular_pago_netto(pago_bruto, deducciones, bonificaciones):
```

```
    return pago_bruto - deducciones + bonificaciones

# Programa principal
print("Simulación de Pago de Nómina\n")

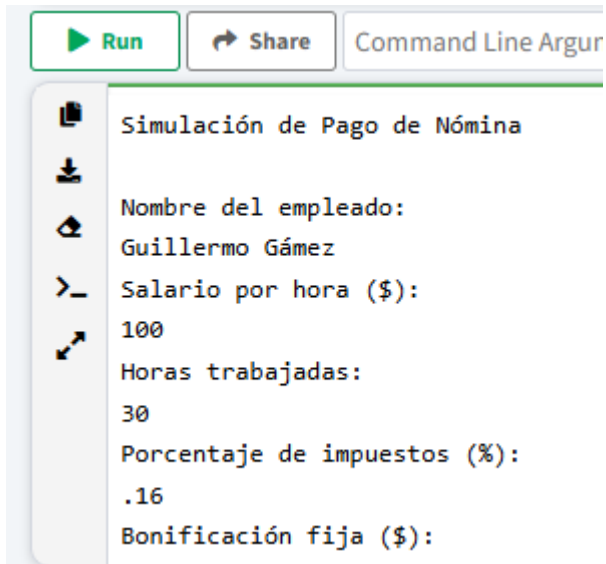
# Entrada de datos
nombre_empleado = input("Nombre del empleado: ")
salario_hora = float(input("Salario por hora ($): "))
horas_trabajadas = float(input("Horas trabajadas: "))
porcentaje_impuestos = float(input("Porcentaje de impuestos (%): "))
bonificacion_fija = float(input("Bonificación fija ($): "))

# Cálculos
pago_bruto = calcular_pago_bruto(salario_hora, horas_trabajadas)
deducciones = calcular_deducciones(pago_bruto, porcentaje_impuestos)
bonificaciones = calcular_bonificaciones(bonificacion_fija)
pago_netto = calcular_pago_netto(pago_bruto, deducciones, bonificaciones)

# Salida
print("\n--- Desglose de Nómina ---")
print(f"Empleado: {nombre_empleado}")
print(f"Pago Bruto: ${pago_bruto:.2f}")
print(f"Deducciones (impuestos): ${deducciones:.2f}")
print(f"Bonificaciones: ${bonificaciones:.2f}")
print(f"Pago Netto: ${pago_netto:.2f}")
```

Ejemplo de Ejecución

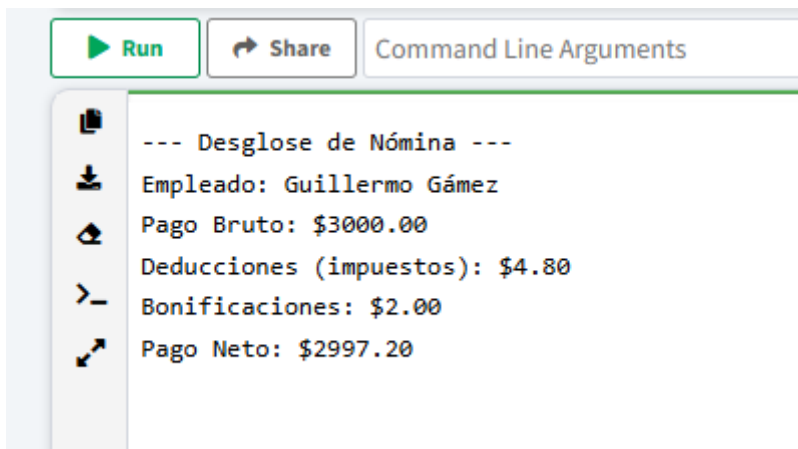
Entrada:



The screenshot shows a code editor interface with a top bar containing 'Run', 'Share', and 'Command Line Arguments' buttons. The main area displays a Python script for a payroll simulation. The script prompts for the employee's name, hourly salary, hours worked, tax percentage, and a fixed bonus. The user has entered the following values: 'Guillermo Gámez', '100', '30', '.16', and '\$2.00'.

```
Simulación de Pago de Nómina  
Nombre del empleado:  
Guillermo Gámez  
> Salario por hora ($):  
100  
Horas trabajadas:  
30  
Porcentaje de impuestos (%):  
.16  
Bonificación fija ($):  
$2.00
```

Salida:



The screenshot shows the same code editor interface as above, but now displaying the output of the payroll simulation. The output is a formatted text block showing the employee's name, gross pay, deductions, bonuses, and net pay.

```
--- Desglose de Nómina ---  
Empleado: Guillermo Gámez  
Pago Bruto: $3000.00  
Deducciones (impuestos): $4.80  
Bonificaciones: $2.00  
Pago Neto: $2997.20
```

Explicación del funcionamiento

1. Entradas del usuario:

- Salario por hora.
- Horas trabajadas.
- Porcentaje de impuestos.

- Bonificación fija.

2. Cálculo de la nómina:

- **Pago bruto:** Se calcula multiplicando el salario por hora por las horas trabajadas.
- **Deducciones:** Se calcula como un porcentaje del pago bruto.
- **Bonificaciones:** Una cantidad fija que se suma al pago neto.
- **Pago neto:** Resultado final después de aplicar las deducciones y bonificaciones.

3. Salida:

- Se presenta un desglose con el nombre del empleado, el pago bruto, las deducciones, las bonificaciones y el pago neto.

BIBLIOGRAFIA

<https://openwebinars.net/blog/que-es-cpp/>

<https://www.ecured.cu/C%2B%2B>

<http://pseint.sourceforge.net/>

<https://www.online-python.com/>

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 1 de 20

Tecnológico Nacional de México

Subdirección Académica

Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales

Nombre de la asignatura: ALGORITMOS Y LENGUAJES DE PROGRAMACIÓN

Plan de estudios: INC-1005

Clave de la asignatura: CAA4

Periodo: Agosto-diciembre/2023

Horas teoría - horas prácticas - créditos: 2 - 2 - 4

Atributos de Egreso: D.I. Diseño Ingenieril-, A.D. Análisis de datos-, CO. Comunicación-,

1. Caracterización de la asignatura

La asignatura de Algoritmos y Lenguajes de Programación aporta al perfil del egresado, los conocimientos, habilidades, metodologías, así como capacidades de análisis y síntesis, para resolver problemas numéricos en hojas de cálculo y lenguajes de programación pertinentes a las áreas de producción y toma de decisiones.

La asignatura es muy importante para los estudiantes ya que el uso de las tecnologías de información y comunicación son muy utilizadas en el ambiente laboral, la capacidad de resolver problemas por medio de las computadoras es muy demandado en la industria manufacturera que es el principal campo de acción de los egresados

La asignatura Algoritmos y Lenguajes de Programación ubicada en cuarto semestre se relacionan con la asignatura previa Álgebra Lineal, por lo que se requiere de los conocimientos previos de matrices y ecuaciones lineales. Con la asignatura posterior de Simulación de sexto semestre en donde se aborda la solución de problemas por medio de modelos matemáticos para lo cual se utilizan la metodología para el diseño de software.

2. Intención didáctica

Este programa de estudios se sugiere eminentemente práctico, es decir, el docente propone el planteamiento de un problema y el estudiante

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 2 de 20

deberá resolverlo mediante diversas técnicas, por mencionar algunas: diagramas de flujo, pseudocódigo y codificación con el monitoreo del docente.

En la unidad uno se aborda la introducción a la computación, el uso y aplicación de la hoja de cálculo.

En el tema dos se plantean el desarrollo de la lógica algorítmica.

El tema tres aborda la programación de un lenguaje estructurado.

En el tema cuatro se consideran las estructuras selectivas y de repetición.

En el tema cinco se abordan temas de aplicación de arreglos y archivos.

En el tema seis se manejan las funciones definidas por el usuario.

A través del conocimiento y uso de diferentes metodologías para la solución de problemas por medio de la computadora, el estudiante adquiere habilidades en el uso de las tecnologías de información y la comunicación a la vez que desarrolla la capacidad de actuar en nuevas situaciones y lleva los conocimientos a la práctica.

Las competencias genéricas se están desarrollando con el tratamiento de los contenidos de la asignatura.

Capacidad de abstracción, análisis y síntesis.

Capacidad de aplicar los conocimientos en la práctica.

Habilidades en el uso de las tecnologías de la información y de la comunicación.

Capacidad de investigación.

Capacidad creativa.

Capacidad para identificar, plantear y resolver problemas.

Capacidad de trabajo en equipo.

Compromiso ético y con la calidad.

El docente debe ser conocedor de la disciplina que está bajo su responsabilidad, conocer su origen y desarrollo histórico para considerar este conocimiento al abordar los temas.

Desarrollar la capacidad para coordinar y trabajar en equipo; orientar el trabajo del estudiante y desarrollar en él la autonomía, el trabajo cooperativo y la toma de decisiones.

Mostrar flexibilidad en el seguimiento del proceso formativo y propiciar la interacción entre los estudiantes.

Tomar en cuenta el conocimiento de los estudiantes como punto de partida y como obstáculo para la construcción de nuevos conocimientos.

3. Competencia de la asignatura

Resuelve problemas de programación mediante la aplicación de herramientas computacionales para el desarrollo de proyectos.

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 3 de 20

4. Análisis por competencias específicas

Competencia No. : 1 Introducción a la computación y hoja de cálculo.

Descripción:

Elabora hojas de cálculo para la solución de problemas en las áreas de ingeniería industrial.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
1.1. Introducción a la computación. 1.2. Sistemas operativos. 1.3. Elementos de Excel. 1.4. Fórmulas y funciones. 1.5. Macros. 1.6. Aplicaciones.	Investiga en Internet sobre las tecnologías de la información. Elabora mapa conceptual de los elementos que conforma un equipo computacional en software y hardware. Aplica las funciones estadísticas, matemáticas, trigonométricas y financieras; de la hoja de cálculo para la solución de problemas. Construye macros para resolver problemas de contexto con una hoja de cálculo.	Investiga en Internet sobre las tecnologías de la información. Elabora mapa conceptual de los elementos que conforma un equipo computacional en software y hardware. Aplica las funciones estadísticas, matemáticas, trigonométricas y financieras; de la hoja de cálculo para la solución de problemas. Construye macros para resolver problemas de contexto con una hoja de cálculo.	Competencias instrumentales Capacidad de análisis y síntesis. Capacidad de organizar y planificar. Conocimientos generales básicos. Conocimientos básicos de la carrera. Comunicación oral y escrita en su propia lengua. Conocimiento de una segunda lengua Habilidades básicas de manejo de la computadora. Habilidades de gestión de información (habilidad para buscar y analizar). Información proveniente de fuentes diversas. Solución de problemas. Toma de decisiones. Competencias interpersonales	15

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 5 de 20

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	40	40	Conocimiento del concepto	D.I.
2._ Examen	20	20	Conocimiento del concepto	A.D.
3._ Tarea	40	40	Conocimiento del concepto	CO.
Total:	100	100		

Competencia No. : 2 Desarrollo de lógica algorítmica.

Descripción:

Desarrolla la capacidad de pensamiento lógicoalgorítmico para la solución de problemas de contexto a través de la computadora.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 6 de 20

2.1. Metodología para la solución de problemas. 2.2. Metodología para el diseño de software: Top down, Bottom up, modular y programación estructurada. 2.3. Elementos y reglas de los lenguajes algorítmicos. 2.3.1. Diagramas de flujo. 2.3.2. Pseudocódigo. 2.4. Implementación de algoritmos. 2.4.1. Secuenciales. 2.4.2. Selectivos. 2.4.3. Repetitivos. 2.5. Pruebas y depuración.	Formula y plantea soluciones lógicas de problemas susceptibles de ser programados. Realiza algoritmos en diagramas de flujo para dar solución a problemas de contexto. Elabora pseudocódigo a partir de diagramas de flujo para la futura codificación de programas.	Formula y plantea soluciones lógicas de problemas susceptibles de ser programados. Realiza algoritmos en diagramas de flujo para dar solución a problemas de contexto. Elabora pseudocódigo a partir de diagramas de flujo para la futura codificación de programas.	Competencias instrumentales Capacidad de análisis y síntesis. Capacidad de organizar y planificar. Conocimientos generales básicos. Conocimientos básicos de la carrera. Comunicación oral y escrita en su propia lengua. Conocimiento de una segunda lengua. Habilidades básicas de manejo de la computadora. Habilidades de gestión de información (habilidad para buscar y analizar). Información proveniente de fuentes diversas Solución de problemas. Toma de decisiones. Competencias interpersonales Capacidad crítica y autocrítica. Trabajo en equipo. Habilidades interpersonales: tolerancia, respeto. Capacidad para trabajar en equipos interdisciplinarios. Respeto a la diversidad y multiculturalidad. Flexibilidad para trabajar en diferentes ambientes de trabajo. Tener compromiso con los valores y principios éticos. Competencias sistémicas Capacidad de aplicar los conocimientos en la práctica.	15
---	---	---	---	-----------

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 7 de 20

			Habilidades de investigación. Capacidad de aprender. Capacidad para diseñar y gestionar proyectos. Capacidad de generar nuevas ideas (creatividad). Habilidad para trabajar en forma autónoma. Poseer iniciativa al elaborar y resolver los problemas propios de los proyectos. Dar enfoques de calidad al realizar el trabajo Búsqueda del logro.	
--	--	--	---	--

Indicadores de alcance	Valor del indicador
A. Desarrolla la capacidad de pensamiento lógico algorítmico	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 8 de 20

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	40	40	Conocimiento del concepto	D.I.
2._ Examen	20	20	Conocimiento del concepto	A.D.
3._ Tarea	40	40	Conocimiento del concepto	CO.
Total:	100	100		

Competencia No. : 3 Introducción a la programación de un lenguaje estructurado.

Descripción:

Aplica los conceptos básicos de la programación para el diseño de programas básicos.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
3.1 Introducción y estructura del entorno de un lenguaje de programación. 3.2 Estructura básica de un programa. 3.3 Tipos de datos. 3.4 Identificadores. 3.5 Almacenamiento, direccionamiento y representación en memoria. 3.6 Proposición de asignación. 3.7 Operadores, operandos y expresiones. 3.8 Prioridad de operadores, evaluación de expresiones. 3.9 Elaboración de programas.	Realiza una investigación sobre los diferentes lenguajes de programación estructurados. Desarrolla la codificación de un programa a partir del diagrama de flujo y pseudocódigo del algoritmo. Resuelve problemas utilizando diferentes tipos de algoritmos de programación básicos.	Realiza una investigación sobre los diferentes lenguajes de programación estructurados. Desarrolla la codificación de un programa a partir del diagrama de flujo y pseudocódigo del algoritmo. Resuelve problemas	Competencias instrumentales Capacidad de análisis y síntesis. Capacidad de organizar y planificar. Conocimientos generales básicos. Conocimientos básicos de la carrera. Comunicación oral y escrita en su propia lengua. Conocimiento de una segunda lengua. Habilidades básicas de manejo de la computadora.	10

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 9 de 20

3.10 Pruebas y depuración		utilizando diferentes tipos de algoritmos de programación básicos.	Habilidades de gestión de información (habilidad para buscar y analizar). Información proveniente de fuentes diversas. Solución de problemas. Toma de decisiones. Competencias interpersonales Capacidad crítica y autocrítica. Trabajo en equipo. Habilidades interpersonales: tolerancia, respeto. Capacidad para trabajar en equipos interdisciplinarios. Respeto a la diversidad y multiculturalidad. Flexibilidad para trabajar en diferentes ambientes de trabajo. Tener compromiso con los valores y principios éticos. Competencias sistémicas Capacidad de aplicar los conocimientos en la práctica. Habilidades de investigación. Capacidad de aprender. Capacidad para diseñar y gestionar proyectos. Capacidad de generar nuevas ideas (creatividad). Habilidad para trabajar en forma autónoma. Poseer iniciativa al elaborar y resolver los problemas propios de los proyectos. Dar enfoques de calidad al realizar el trabajo. Búsqueda del logro	
---------------------------	--	--	--	--

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 10 de 20

Indicadores de alcance	Valor del indicador
A. Aplica los conceptos básicos de la programación	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	40	40	Conocimiento del concepto	D.I.
2._ Examen	20	20	Conocimiento del concepto	A.D.
3._ Tarea	40	40	Conocimiento del concepto	CO.
Total:	100	100		

Competencia No. : 4 Estructuras selectivas y de repetición

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 11 de 20

Descripción:

Aplica las estructuras de un lenguaje de programación para dar solución a problemas cuantitativos.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
4.1 Selectiva simple. 4.2 Selectiva doble. 4.3 Selectiva anidada. 4.4 Selectiva múltiple. 4.5 Repetir mientras. 4.6 Repetir hasta. 4.7 Repetir desde. 4.8 Elaboración de programas.	Realiza programas utilizando estructuras selectivas y de iteración para procesos condicionales o repetitivos. Elabora programas orientados al área de la ingeniería industrial.	Realiza programas utilizando estructuras selectivas y de iteración para procesos condicionales o repetitivos. Elabora programas orientados al área de la ingeniería industrial.	Competencias instrumentales Capacidad de análisis y síntesis. Capacidad de organizar y planificar. Conocimientos generales básicos. Conocimientos básicos de la carrera. Comunicación oral y escrita en su propia lengua. Conocimiento de una segunda lengua. Habilidades básicas de manejo de la computadora. Habilidades de gestión de información (habilidad para buscar y analizar). Información proveniente de fuentes diversas. Solución de problemas. Toma de decisiones. Competencias interpersonales Capacidad crítica y autocrítica Trabajo en equipo Habilidades interpersonales: tolerancia, respeto Capacidad para trabajar en equipos interdisciplinarios Respeto a la diversidad y	10

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 12 de 20

			multiculturalidad Flexibilidad para trabajar en diferentes ambientes de trabajo Tener compromiso con los valores y principios éticos Competencias sistémicas Capacidad de aplicar los conocimientos en la práctica Habilidades de investigación Capacidad de aprender Capacidad para diseñar y gestionar proyectos Capacidad de generar nuevas ideas (creatividad) Habilidad para trabajar en forma autónoma Poseer iniciativa al elaborar y re-solver los problemas propios de los proyectos Dar enfoques de calidad al realizar el trabajo Búsqueda del logro	
--	--	--	--	--

Indicadores de alcance	Valor del indicador
A. Aplica las estructuras de un lenguaje de programación	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 13 de 20

	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	40	40	Conocimiento del concepto	D.I.
2._ Examen	20	20	Conocimiento del concepto	A.D.
3._ Tarea	40	40	Conocimiento del concepto	CO.
Total:	100	100		

Competencia No. : 5 Arreglos y archivos.

Descripción:

Aplica los arreglos y archivos en la solución de problemas que requieran el almacenamiento y manipulación de datos.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
5.1 Arreglo unidimensionales. 5.2 Arreglo bidimensionales y multidimensionales.	Desarrolla programas utilizando arreglos unidimensionales, bidimensionales y multidimensionales.	Desarrolla programas utilizando arreglos unidimensionales, bidimensionales y multidimensionales.	Competencias instrumentales Capacidad de análisis y síntesis. Capacidad de organizar y planificar. Conocimientos generales	20

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 14 de 20

5.3 Apertura, entrada-salida de datos, y cierre de archivos. 5.4 Elaboración de programas.	Diseña programas que resuelvan problemas de aplicación del álgebra matricial. Elabora programas para almacenar la información de manera permanente.	Diseña programas que resuelvan problemas de aplicación del álgebra matricial. Elabora programas para almacenar la información de manera permanente.	básicos. Conocimientos básicos de la carrera. Comunicación oral y escrita en su propia lengua. Conocimiento de una segunda lengua. Habilidades básicas de manejo de la computadora. Habilidades de gestión de información (habilidad para buscar y analizar). Información proveniente de fuentes diversas. Solución de problemas. Toma de decisiones. Competencias interpersonales Capacidad crítica y autocrítica Trabajo en equipo Habilidades interpersonales: tolerancia, respeto Capacidad para trabajar en equipos interdisciplinarios Respeto a la diversidad y multiculturalidad Flexibilidad para trabajar en diferentes ambientes de trabajo Tener compromiso con los valores y principios éticos Competencias sistémicas Capacidad de aplicar los conocimientos en la práctica. Habilidades de investigación. Capacidad de aprender. Capacidad para diseñar y gestionar proyectos. Capacidad de generar nuevas ideas (creatividad).	
--	---	---	---	--

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 15 de 20

Indicadores de alcance	Valor del indicador
A. Aplica los arreglos y archivos en la solución de problemas	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	40	40	Conocimiento del concepto	D.I.
2._ Examen	20	20	Conocimiento del concepto	A.D.
3._ Tarea	40	40	Conocimiento del concepto	CO.
Total:	100	100		

Competencia No. : 6 Funciones.

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 16 de 20

Descripción:

Utiliza las funciones definidas por el usuario para optimizar el código de programación.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje (estudiante)	Actividades de enseñanza (profesor)	Desarrollo de competencias genéricas	Horas teórico-práctica
6.1 Introducción. 6.2 Funciones estándar. 6.3 Entrada y salida de datos. 6.4 Funciones definidas por el usuario. 6.5 Pase por valor. 6.6 Pase por referencia. 6.7 Elaboración de programas.	Realiza programas que requieran funciones de paso por valor. Elabora programas que utilicen funciones de pase por referencia. Realiza programas aplicados a la ingeniería industrial utilizando funciones. Desarrolla el proyecto final por equipo que resuelva un problema de la ingeniería industrial utilizando los conocimientos adquiridos en el curso.	Realiza programas que requieran funciones de paso por valor. Elabora programas que utilicen funciones de pase por referencia. Realiza programas aplicados a la ingeniería industrial utilizando funciones. Desarrolla el proyecto final por equipo que resuelva un problema de la ingeniería industrial utilizando los conocimientos adquiridos en el curso.	Competencias instrumentales Capacidad de análisis y síntesis. Capacidad de organizar y planificar. Conocimientos generales básicos. Conocimientos básicos de la carrera. Comunicación oral y escrita en su propia lengua. Conocimiento de una segunda lengua. Habilidades básicas de manejo de la computadora. Habilidades de gestión de información (habilidad para buscar y analizar). Información proveniente de fuentes diversas. Solución de problemas. Toma de decisiones. Competencias interpersonales Capacidad crítica y autocrítica. Trabajo en equipo. Habilidades interpersonales: tolerancia, respeto. Capacidad para trabajar en equipos interdisciplinarios. Respeto a la diversidad y	10

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 17 de 20

			multiculturalidad. Flexibilidad para trabajar en diferentes ambientes de trabajo. Tener compromiso con los valores y principios éticos. Competencias sistémicas Capacidad de aplicar los conocimientos en la práctica. Habilidades de investigación. Capacidad de aprender. Capacidad para diseñar y gestionar proyectos.	
--	--	--	---	--

Indicadores de alcance	Valor del indicador
A. Utiliza las funciones definidas por el usuario para optimizar el código de programación	100

Niveles de desempeño

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple en su totalidad con los indicadores de alcance.	95 - 100
	Notable	Cumple en su totalidad con los indicadores de alcance de manera parcial.	85 - 94
	Bueno	Cumple con al menos 60% de los indicadores con desempeño excelente.	75 - 84
	Suficiente	Cumple con al menos 60% de los indicadores con desempeño parcial.	70 - 74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores definidos en el desempeño excelente.	NA (No alcanzada)

Matriz de evaluación

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 18 de 20

Evidencia de aprendizaje	%	A	Evaluación formativa de la competencia	Atributos de Egreso
1._ Tarea	40	40	Conocimiento del concepto	D.I.
2._ Examen	20	20	Conocimiento del concepto	D.I.
3._ Tarea	40	40	Conocimiento del concepto	CO.
Total:	100	100		

5. Fuentes de información y Apoyos didácticos

Fuentes de información: 1. Ceballos, Javier(1998). Programación Orientada a Objetos con C++. Rama. datos. Mc Graw Hill. 2. DeitelP y Deitel H.(1999). Programación Orientada a Objetos C++, Prentice Hall. 3. Joyanes Luis,(2003) Fundamentos de programación, algoritmos y estructura de 4. Joyanes Luis,(2010) Algoritmos, Programación y Estructuras de Datos,Mc Graw Hill 5. Joyanes Luis. (1996). Fundamentos de programación 2ª edición. Mc Graw Hill 6. JoyanesLuis(1990), Problemas de metodología de la programación.McGraw Hill. 7. JoyanesLuis(1998), Metodología de la programación, Mc Graw Hill. 8. LevineGuillermo(1993), Introducción a la computación y a la programación estructurada.Mc Graw Hill. 9. Lippman Stanley yLajoleJasse(1998), C++ Primer Mc Graw Hill. 10. Peñaloza Romero (2004).Fundamentos de programación 4ª. Edición. Alfaomega 11. Cid. José (2001).Computación básica.1ª. edición UPAEP	Apoyos didácticos: Proporcionado por el docente.
--	--

6. Calendarización de evaluación en semanas (6):

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.												Código: ITSLP-SAC-PG-001-02					
													Revisión: 4					
													Página: 19 de 20					

Semana:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
TP	ED		EF1			EF2		EF3		EF4				EF5		EF6	ES	
TR	ED		EF1			EF2		EF3		EF4				EF5		EF6	ES	
SD					SD1				SD2					SD3				

TP=tiempo planeado

TR=tiempo real

SD=seguimiento departamental

ED=evaluación diagnóstica

EFn=evaluación formativa(competencia específica n)

ES=evaluación sumativa

Fechas de evaluación por unidad

Tema	Evaluación diagnóstica	Evaluación formativa	Inicio Tema	Fin Tema	Sumativa
1	21/08/2023	08/09/2023	21/08/2023	08/09/2023	
2		29/09/2023	11/09/2023	29/09/2023	
3		13/10/2023	02/10/2023	13/10/2023	
4		27/10/2023	16/10/2023	27/10/2023	
5		24/11/2023	30/10/2023	24/11/2023	
6		08/12/2023	27/11/2023	08/12/2023	11/12/2023

Fecha de elaboración:09/01/2025

	Instrumentación Didáctica para la Formación y Desarrollo de Competencias Profesionales.	Código: ITSLP-SAC-PG-001-02
		Revisión: 4
		Página: 20 de 20

GUILLERMO GÁMEZ GALEANA

Nombre del(de la) profesor(a)

MC. ANTONIO MAYER SÁNCHEZ

Nombre del(de la) Jefe(a) de Departamento Académico