

Taller del proyecto del semestre

a) El ciclo de vida de construcción de un programa sigue una serie de fases estructuradas para garantizar que el software cumpla con los requisitos del cliente y funcione correctamente en el entorno en el que será desplegado. Estas fases incluyen:

1. Toma de requerimientos: Se recopila información detallada sobre lo que el cliente necesita. Se identifican los objetivos del software, los requisitos funcionales (qué debe hacer el programa) y los requisitos no funcionales (rendimiento, seguridad, escalabilidad).
2. Análisis del problema: En esta fase, se definen y examinan en profundidad la información obtenida en la toma de requerimientos. Se definen restricciones, posibles riesgos y el impacto del sistema en el entorno donde se desplegará. También se elaboran casos de uso para entender cómo interactuarán los usuarios con el software.
3. Diseño de software: Se estructura la solución mediante la definición de la arquitectura del software. Esto incluye la selección de patrones arquitectónicos, tecnologías, bases de datos y protocolos de comunicación. También se diseñan diagramas UML (como diagramas de clases, de secuencia y de despliegue) para visualizar la estructura y comportamiento del sistema.
4. Implementación (desarrollo): Se escribe el código del software basándose en el diseño. Se aplican buenas prácticas de programación, principios de desarrollo como SOLID y patrones de diseño para garantizar un código limpio, modular y evitable.

5. Pruebas y Validación: Se ejecutan pruebas unitarias, de integración y funcionales para garantizar que cada módulo del sistema funcione correctamente y que el software en su conjunto cumpla con los requisitos. También se realizan pruebas de rendimiento, seguridad y usabilidad.

6. Despliegue: Una vez aprobadas las pruebas, el software se instala en el entorno de producción. Puede implicar la configuración de servidores, bases de datos y otros componentes.

7. Mantenimiento y evolución: Después del despliegue, el software puede requerir ajustes, corrección de errores, optimización del rendimiento y nuevas funcionalidades en función de la retroalimentación del usuario y cambios en los requisitos del negocio.

b) El análisis de un problema en el desarrollo de software es una fase crítica que ayuda a determinar la mejor solución posible. Los aspectos más relevantes que deben considerarse incluyen:

1. Comprendiendo del dominio: Es necesario entender el contexto en el que se aplicará la solución. Por ejemplo, el desarrollo de software debe cumplir, para un hospital tiene requisitos y normativa diferentes a los de una tienda en línea.

2. Identificación de los requisitos funcionales: Se definen todos los funciones y características que el software debe cumplir. Por ejemplo, en un sistema bancario, un requisito funcional puede ser que los usuarios pueden realizar transferencias en línea.

3. Definición de los requisitos no funcionales: Son características que afectan el desempeño del software, como la capacidad de soportar un alto número de usuarios concurrentes, la seguridad contra ataques informáticos o la facilidad de mantenimiento.

4. Restricciones tecnológicas: Se evalúan los límites y condiciones impuestos por el hardware, software o regulaciones del negocio. Por ejemplo, si un cliente ya usa hardware específico, la solución debe ser adaptarse a esa infraestructura.

5. Casos de uso y escenarios: Se crean descripciones detalladas de cómo los usuarios interactuarán con el software en diferentes situaciones. Esto ayuda a prever problemas y mejorar la experiencia de usuario.

6. Identificación de riesgos: Se analizan posibles problemas que podrían surgir durante el desarrollo o implementación del software, como cambios en los requisitos, fallas en la infraestructura o incompatibilidades con otros sistemas existentes.

c) El proceso de solución de problemas en el desarrollo de software sigue una serie de pasos estructurados para garantizar que la solución propuesta sea viable y efectiva:

1. Definición del problema: Se establece con claridad cuál es el problema a resolver. Se identifican las necesidades del negocio y se delimitan los objetivos del sistema.

2. Análisis del problema: Se recopila información detallada sobre el problema, sus causas y sus posibles soluciones. En esta etapa, se pueden hacer entrevistas con los usuarios, revisar sistemas existentes y analizar datos históricos para comprender mejor la situación.

3. Diseño de la solución: Se define la arquitectura del Sistema y los componentes que lo conformarán. Se eligen tecnologías, patrones de diseño y estrategias de integración para asegurar que la solución sea eficiente y escalable.

4. Implementación: Se traduce el diseño en código funcional. Se adoptan metodologías como el desarrollo ágil, integración continua y pruebas automatizadas para mejorar la calidad del software.

5. Pruebas y Validación: Se realizan diversas pruebas para garantizar que el sistema cumple con los requerimientos. Estas incluyen pruebas unitarias (para verificar el correcto funcionamiento de cada módulo), pruebas de integración (para comprobar la comunicación entre módulos) y pruebas de aceptación (para validar que el software cumple con lo que el cliente necesita).

6. Mantenimiento y mejora continua: Una vez implementado, el Software entra en una fase de mantenimiento en la que se corrigen errores, se optimiza el rendimiento y se agregan nuevas funcionalidades según sea necesario.

d) Cuando un software está listo para su entrega, es importante proporcionar una serie de elementos que permitan su correcto uso, mantenimiento y evolución. Estos incluyen:

1. Código fuente: Se entrega el código del programa debidamente documentado para que otros desarrolladores puedan entenderlo y modificarlo si es necesario. En algunos casos, se incluyen archivos de configuración, scripts de base de datos y biblioteca utilizada.
2. Manual de usuario: Documento que explica el uso del software para los usuarios finales. Debe incluir capturas de pantalla, ejemplos de uso y pasos detallados para ejecutar las funciones principales.
3. Resultados de pruebas: Extracción informe de pruebas realizadas durante el desarrollo, incluyendo pruebas unitarias, de integración y de aceptación. Esto demuestra que el software ha sido validado y cumple con los requerimientos establecidos.
4. Documentación técnica: Describe la arquitectura del software, las tecnologías utilizadas, la estructura del código, la base de datos y la interfaz de comunicación con otros sistemas. Es fundamental para futuros mantenimientos o ampliaciones del software.
5. Archivos ejecutables o instaladores: Si el software requiere instalación, se proporcionan los archivos necesarios junto con un manual de instalación, que detalla los pasos a seguir para su correcta implementación en el entorno del cliente.

6. Plan de soporte y mantenimiento: Se establece un acuerdo

con el cliente sobre cómo se gestionarán futuras actualizaciones, corrección de errores y mejoras del software. Puede incluir un periodo de garantía o un contrato de soporte técnico.

7. Acceso y credenciales: En caso de que el software esté alojado en servidores en la nube, se deben proporcionar credenciales de acceso, configuraciones de seguridad y protocolos de respaldo.

e) Identificar los aspectos que forman parte de un problema.

Problema: Un banco quiere crear un programa para manejar sus cajeros automáticos. Dicho programa sólo debe permitir retirar dinero y consultar el saldo de una cuenta.

Solución:

Cliente	La entidad bancaria que desea implementar el programa. Es quien contrata el desarrollo y define las necesidades.
Usuario	Los clientes del banco que utilizan los cajeros. Son las personas que interactuarán con el programa para realizar transacciones.
Requerimientos funcionales	El programa debe permitir retirar dinero de una cuenta. Debe mostrar el saldo disponible en la cuenta bancaria.
	Se requiere acceso a la base de datos de la cuenta. Cada cuenta tiene asociado un saldo y un N°. de Id. Los usuarios deben autenticarse antes de realizar operaciones. Existe límite de retiros diarios que deben repetirse.

**Requerimiento
No Funcional**

Seguridad: El programa debe garantizar que solo los clientes autorizados tengan acceso a sus cuentas.

Disponibilidad: El software debe funcionar sin interrupciones 24/7.

Velocidad: Las operaciones deben ejecutarse en corto tiempo.

Escalabilidad: Debe de existir la posibilidad de manejar múltiples transacciones simultáneamente sin disminuir el rendimiento.

- F) Crear habilidad en la identificación y especificación de requerimientos funcionales. Identifique y especifique tres requerimientos.

Requerimiento funcional 1

Nombre	Consulta de Saldo
Retorno	Permite a los usuarios consultar el saldo de su cuenta bancaria.
Entrada	Número de cuenta, Clave de acceso
Resultado	Se muestra el saldo actual disponible de la cuenta del usuario.

Requerimiento funcional 2

Nombre	Transferencia entre cuentas
Retorno	Permite a los usuarios realizar transferencias de dinero entre cuentas del mismo banco.
Entrada	Número de cuenta origen, Número de cuenta destino, Monto a transferir, Clave de seguridad.



Resultado	Se hace la transferencia del monto elegido. se muestra un mensaje de confirmación con el monto pagado.
-----------	---

Requerimiento funcional 3

Nombre	Pago de servicios.
Resumen	Permite a los usuarios pagar facturas de servicios como agua, luz, internet, etc.
Entrada(s)	Número de cuenta, servicio a pagar, cantidad a pagar.
Resultado	Se realiza el pago y genera un comprobante de la transacción.

g) Crear Mabiliad en la identificación y especificación de requerimientos funcionales. Un programa para manejar un triángulo, identifique y especifique tales requerimientos funcionales.

Requerimiento funcional 1

Nombre	Determinar el tipo de triángulo
Resumen	El sistema debe identificar si un triángulo es equilátero, isósceles o escaleno según sus lados.
Entrada(s)	Tres valores numéricos que representan las longitudes de los lados del triángulo



Resultado	El tipo de triángulo identificado
-----------	-----------------------------------

Requerimiento funcional 2

Nombre	Calcular el perímetro del triángulo
Resumen	El sistema debe calcular el perímetro sumando las longitudes de los tres lados.
Entrada(s)	Tres valores numéricos que representan las longitudes de los lados del triángulo
Resultado	Un número que indica el perímetro del triángulo

Requerimiento funcional 3

Nombre	Calcular el área del triángulo
Resumen	El sistema debe calcular el área utilizando la fórmula de Herón ($A = \sqrt{s(s-a)(s-b)(s-c)}$)
Entrada(s)	Tres valores numéricos que representan las longitudes de los lados del triángulo
Resultado	Un número que indica el área del triángulo

h) Identificar las entidades del mundo: Un programa que maneje un triángulo.

Nombre de Entidad	Descripción
Triángulo	Representa la figura geométrica principal. Tiene atributos como lado, tipo, área, perimetro.
Objeto	Persona que maneja los datos del mundo y recibe las resultados.
Sistema	Sistema que procesa la información y ejecuta las funciones necesarias para determinar características del triángulo.

Punto de reflexión: ¿Qué pasa si no identificamos bien las entidades del mundo?

Si no identificamos correctamente las entidades, el diseño del sistema será confuso o incompleto, lo que podría llevar a errores en la implementación. Además, podríamos omitir información clave para el funcionamiento del programa.

Punto de reflexión: ¿Cómo decidir si se trata de una entidad y no solo de una característica de una entidad y identificada?

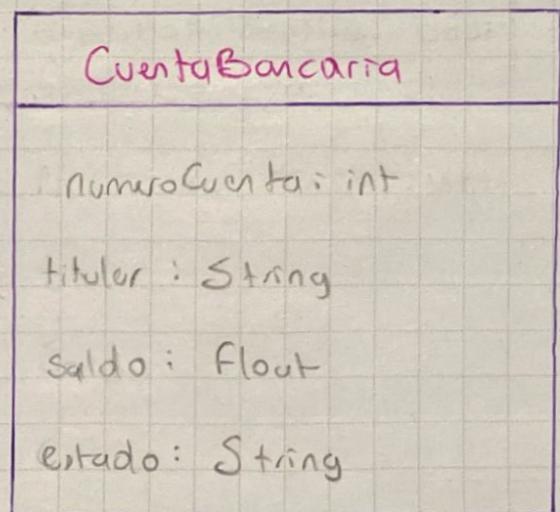
Para diferenciar una entidad de una característica, podemos preguntarnos si el elemento en cuestión puede existir por sí mismo y si tiene atributos propios. Si solo complementa o describe otra entidad, entonces es una característica.

- ii) Para cada uno de los cinco entidades, identifique los atributos, sus valores posibles, y escriba la clase en UML.

Clase: CuentaBancaria

Atributos	Valores posibles
Número obj. Cuenta	Número único (ej. 123456789)
Titular	Nombre del cliente (ej. "Juan Pérez")
Saldo	Monto en la cuenta (ej. \$0 - \$1,000,000)
Estado	Activa, inactiva, Bloqueada

Diagrama UML



Clase: Cuenta Corriente

Atributo	Valores posibles
Límite de Sobregiro	Monto permitido en negativo (ej. \$0 - \$5000)
Comisión por manejo	Tarifa manual (ej. \$5 - \$20)
Chequera	Sí / No

Diagrama UML

Cuenta Corriente
l límiteSobregiro : float
comisionManejo : float
chequera : bool

Clase: CuentaAhorro

Atributo	Valores Posibles
Tasa de interés	Porcentaje anual (ej: 0,5% - 5%)
Saldo Mínimo	Monto mínimo requerido (ej \$0 - \$1000)
Depósito automático	Sí / No

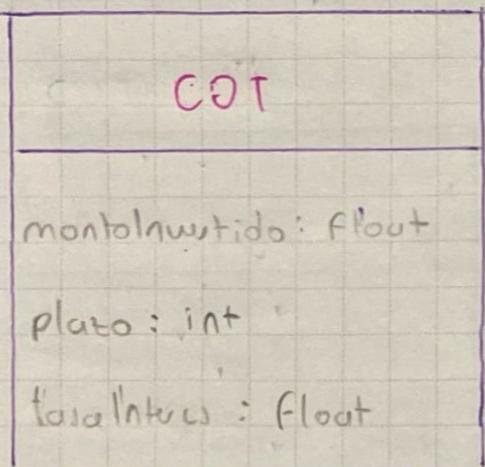
Diagrama UML

CuentaAhorro
tasaInteres: float
SaldoMinimo: float
depositoAutomatico: bool

Clase: COT

Atributos	Valores posibles
Monto invertido	Cantidad de dinero (ej. £500 - £10,000)
Plazo	Tiempo en meses (ej. 3, 6, 12, 14)
Tasa de interés	Porcentaje anual (ej. 3% - 10%)

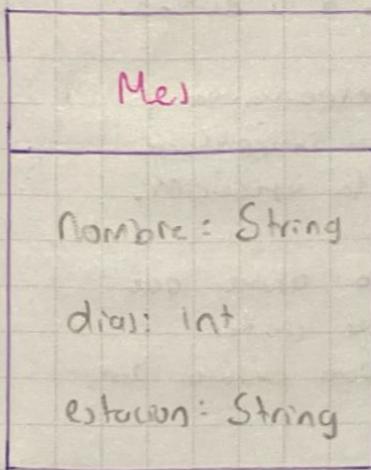
Diagrama UML



Clase: Mes

Atributo	Valores posibles
Nombre	"Enero", "Febrero", ..., "Diciembre"
Dia	1, 2, 3, 4, ..., 30, 31
Estación	"Invierno", "primavera", "verano", "otoño"

Diagrama UML



j) Reflexionar sobre el nivel de precisión que debe tener un algoritmo para evitar ambigüedades

Solución:

Al analizar el nivel de precisión que debe tener un algoritmo para moverse en el metro de París, se pueden identificar varios problemas si las instrucciones no son claras o completas.

Ambigüedad en las instrucciones: Si el algoritmo usa términos poco específicos como "tome la línea correcta", el usuario puede no saber cuál es la línea correcta sin información adicional, como el número o color, de la línea.

Falta de detalles esenciales: No especificar la dirección del tren podría llevar al usuario en sentido contrario a su destino.

Interpretaciones múltiples: Dependiendo de la experiencia y contexto del usuario, algunos pasos pueden interpretarse de manera diferente, lo que genera errores en la ejecución.

Dependencia del sentido común: Si el algoritmo asume que el usuario utilizará su "sentido común" para resolver cierta ambigüedad, esto puede ser problemático para personas sin experiencia en ese sistema de transporte.

Estandarización y claridad: Un algoritmo bien diseñado debe usar instrucciones precisas y detalladas para que cualquier usuario, sin importar su nivel de conocimiento, pueda seguirlo sin errores.

Gestión de requisitos de software

Trabajo preliminar: Caso de estudio

El empleado:

Se requiere una aplicación que permita manejar la información de un empleado. Se maneja la siguiente información:

- Nombre, Apellido, Género, fecha de nacimiento, foto, fecha de ingreso a la empresa y salario básico

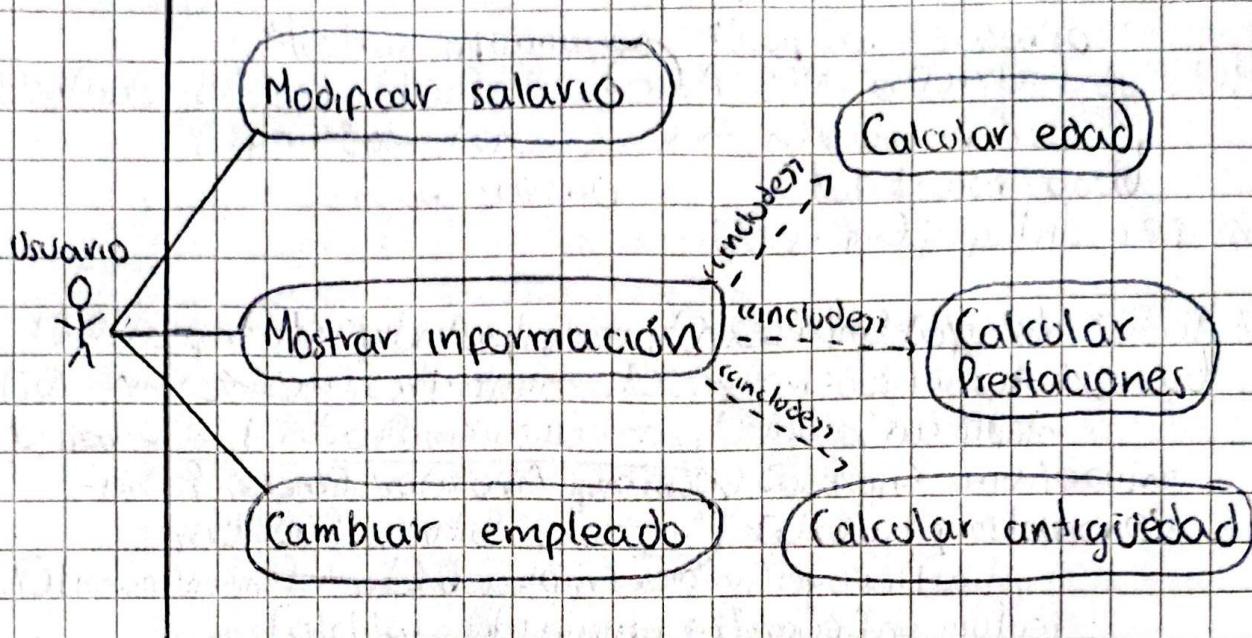
La aplicación permite visualizar la información del empleado, y hacer los siguientes cálculos:

- Edad del empleado, usando la fecha de nacimiento
- Antigüedad del empleado, usando la fecha de ingreso
- Prestaciones del empleado, usando $\text{prestaciones} = (\text{antiguedad}) \times \text{salario} / 12$.

El programa debe permitir (Requerimientos):

- Visualizar la información del empleado.
- Modificar el salario del empleado.
- Calcular la edad del empleado.
- Calcular la antigüedad del empleado en la empresa.
- Calcular las prestaciones del empleado.
- Cambiar el empleado.

Casos de uso



Clases, atributos y métodos

Clase: Empleado

Atributos: nombre, apellido, genero, imagen, salario

Métodos: Empleado(), darNombre(), darGenero(),
darFechaNacimiento(), darFechaIngreso(),
darImagen(), darSalario(), darApellido(),
calcularEdad(), calcularAntiguedad(),
calcularPrestaciones(), cambiarEmpleado(Nombre,
apellido, genero, fechaNacimiento, fecha
ingreso, salario, imagen, cambiarSalario(Salario),
difFechaActual()), metodo1(), metodo2()

Claase: Fecha

Atributos: dia, mes, año

Métodos: fecha(Dia, Mes, Ano), darDia(), darMes(), darAno(),
darDiferenciaEnMeses(Fecha), toString()

Interfaz: InterfazEmpleado

Métodos: InterfazEmpleado(), calcularAntiguedadEmpleado(),
calcularEdadEmpleado(), calcularPrestacionesEmpleado(),
modificarSalario(), actualizarEmpleado(),
cambiarEmpleado(Nombre, Apellido, Genero, fecha-
nacimiento, fechaIngreso, Salario, Imagen),
mostrarDialogoCambiarEmpleado(), refuncionOpcion1(),
refuncionOpcion2(), main()

Interfaz: PanelImagen

Métodos: PanelImagen()

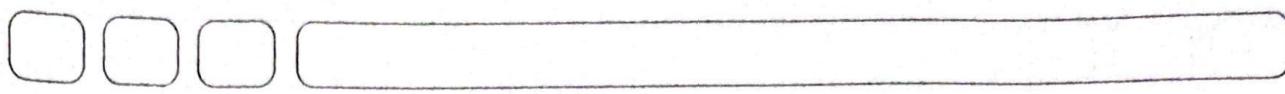
Interfaz: PanelOpciones

Métodos: PanelOpciones(), actionPerformed()

Atributos: CambiarEmpleado, Opcion1, Opcion2, btnCambiar-
Empleado, btnOpcion1, btnOpcion2

Interfaz: PanelConsultas

Atributos: CalcularEdad, CalcularAntiguedad, CalcularPrestaciones,
txtEdad, txtAntiguedad, txtPrestaciones, btnEdad,
btnAntiguedad, btnPrestaciones



Interfaz 2 : DialogoCambiarEmpleado

Atributos: SeleccionarImagen, Aceptar, cancelar, txtNombre, txtApellido, cbGenero, dFechaNacimiento, dFechaIngreso, txtSalario, txImage, btnSeleccionImagen, btnAceptar, btnCancelar

Métodos: DialogoCambiarEmpleado(), actionPerformed()

Interfaz: PanelDatos

Atributos: lblNombre, lblApellido, lblIngreso, lblGenero, lblSalario, lblImagen, txtNombre, txtApellido, txtIngreso, txtFNacimiento, txtGenero, txtSalario, btnModificarSalario, ModificarSalario

Métodos: PanelDatos(), actualizarCompos(), actualizarSalario(), actionPerformed()

Taller Gestión de requisitos de software

DD | MM | AA

Punto n:

Idea 1: Sistema de recomendación de canciones

Requerimiento
funcional
1

Nombre: R1: Crear usuario

Resumen: Permite la creación de un usuario

Entradas: Nombres, apellidos, correo electrónico, número tel

Resultado: El usuario queda creado y con acceso a la aplicación

Requerimiento
funcional
2

Nombre: R2: Agregar canción

Resumen: Permite a un administrador agregar canciones

Entradas: Nombre, Artista, álbum, fecha lanzamiento, duración

Resultado: Se actualizan las canciones disponibles en la aplicación

Requerimiento
funcional
3

Nombre: R3: Recomendar canción

Resumen: El usuario podrá marcar canciones como favoritas o géneros, basado en su selección se recomendarán canciones similares

Entradas: Género Favorito, estado de ánimo, bandas o artistas preferidos, canciones escuchadas recientemente, Puntuación

Resultado: Se recomienda al usuario una canción basada en la similitud y puntuación

Requerimiento
funcional
4

Nombre: R4: Puntuar Canción

Resumen: El usuario podrá calificar una canción

Entradas: Canción, Puntuación, Reseña

Resultado: La canción se agrega la reseña a la canción

Idea 2: Sistema de rutas

Nombre: Crear ruta (R1)

Requerimiento
Funcional
1

Resumen: agrega el recorrido de un bw al mapa

Entradas: Nombre, id, Destino inicial, Destino final, Fondo

Resultado: se agrega una nueva ruta al mapa

Requerimiento
Funcional
2

Nombre: Crear Usuario (R2)

Resumen: Permite la creación de un usuario para uso de la aplicación

Entradas: Nombre, usuario, contraseña, correo

Resultado: se agrega un usuario para uso del sistema

Requerimiento
Funcional
3

Nombre: R3: Buscar ruta

Resumen: Permite Buscar una ruta

Entradas: Destino inicio, Destino final

Resultado: regresa la mejor ruta para el usuario

Requerimiento
Funcional
4

Nombre: R4: Calcular tiempo de llegada

Resumen: Permite aproximar el tiempo de llegada a la ubicación final

Entradas: Destino actual, Destino final, tiempo trayecto total

Resultado: regresa la aproximación de el tiempo de llegada