



Principios de Desarrollo de Software



Toma de nota diseñada por **Alejandro Ríos** para la clase de Principios de Desarrollo de Software 2022-1

Unidad 1: Contexto inicial de la Ingeniería de Software

Motivación

Fases del Desarrollo de Software

Teoría General de Sistemas

Elementos Básicos de un Sistema

Tipos de Sistemas

Innovación y Creatividad en Ingeniería de Sistemas

Innovación

Creatividad

Unidad 1: Pensamiento Computacional

Pensamiento Computacional

Pilares

Historia de la Computación

Unidad 1: PSP y TSP

Actividades para producir Software

PSP (Proceso de Software Personal)

TSP (Proceso de Software en Equipo)

Niveles de PSP

Herramientas de Planificación de Proyectos

Roles

Gerente de Proyectos

Analista

Diseñador

Desarrollador

Administrador

Probador / Tester

Unidad 1: Análisis de problemas y pasos para la solución de problemas

Estructura

Bien Estructurado

Mal Estructurado

Complejidad

Simple

Complejo

Complicado

Dinamicidad

Estable

Dinámico

Dominio

Componentes de un problema

Pasos para solucionar un problema

Métodos Científicos

Investigación en Ing. de Sistemas

Tipos de Razonamiento

Conocimiento

Unidad 1: Problemas en la Ingeniería de Sistemas

Tipos de problemas Ing. de Sistemas

Problemas Informáticos

Problemas de Software

Problemas Computacionales

Modelo Cinefyn

Unidad 2: Esquemas o representaciones gráficas

Pensamiento visual

Tipos de esquemas

Representación libre *

Mapa mental *

Mapa Conceptual *

Red semántica

Diagrama de Pert *

Gráfico Estadístico

Unidad 3: El papel de los requisitos para entender una necesidad

Requisito

Dimensiones de los Requisitos

Ámbito

Características

Audiencia

Tipos de Requisitos

Requisitos como Sentencias de Lenguaje Natural

Historias de Usuario

Unidad 3: Creación del Prototipo

Tipos de Prototipos

Sketches (Bocetos)

Wireframe

Mockup

Prototipado

Características

Tipos de Prototipos - I

Tipos de Prototipos - II

Rápido o desechable

Evolutivos

Tipos de Prototipos - III

Recursos

Unidad 3: Modelado del sistema

UML (Unified Modeling Language)

¿Para qué?

Perspectivas de un Modelo

Unidad 3: Diagrama de casos de uso

Escenarios

Características de un escenario

Casos de Uso

Notación I

Plantilla descripción Caso de Uso

Notación II

Construir un Diagrama de Casos de Uso

1. Diagrama de Contexto

2. Diagrama de Casos de Uso Inicial

3. Diagrama de Casos de Uso refinado

4. Plantillas de descripción

Unidad 3: Principios básicos del modelo de objetos

Orientación a Objetos (OO)

Principios Básicos

Clase

Unidad 3: Diagrama de Clases

Tipos de Relaciones

- Dependencia
 - Asociación
 - Agregación
 - Composición
 - Herencia
- Visibilidad de Atributos
 - Visibilidad Pública
 - Visibilidad Privada
 - Visibilidad Protegida
- Unidad 3: Diagrama de clases a código
- Unidad 3: Diagrama de secuencias
 - Diagramas UML
 - Diagrama de secuencias
 - Partes Diagrama de secuencias
 - Objeto y línea de vida
 - Activación
 - Mensaje
 - Creación de un objeto en la secuencia
 - 3 Principios básicos
- Unidad 4: Mantenimiento
 - Componentes claves del Mantenimiento
 - Tipos de Mantenimiento
 - Preventivo
 - Correctivo
 - Adaptativo / Continuo
 - Beneficios
 - Ética Profesional Creación de Software
- Unidad 4: Procesos de Desarrollo de Software
 - Codificar y corregir - Code and Fix
 - Cascada (Tradicional)
 - Scrum (Ágil)

Unidad 1: Contexto inicial de la Ingeniería de Software

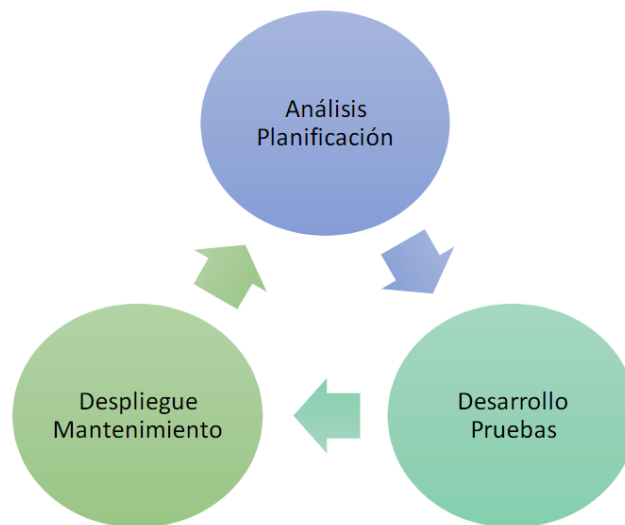
Motivación

1. Planear
2. Analizar
3. Diseñar
4. Codificar
5. Validar y verificar

Ingeniería: Invención y utilización de recursos para aprovechar recursos naturales o industriales

Software: Programas / instrucciones y reglas para ejecutar una tarea. Así mismo, es el desarrollo, operación y mantenimiento de software de forma sistemática y disciplinada

Fases del Desarrollo de Software



- Evolucionar para adaptarse a las necesidades del cliente (Mantenibilidad)
- Garantizar la seguridad (Confiabilidad)
- No malgastar los recursos (Eficiencia)
- Facilidad de uso (Usabilidad)

Teoría General de Sistemas

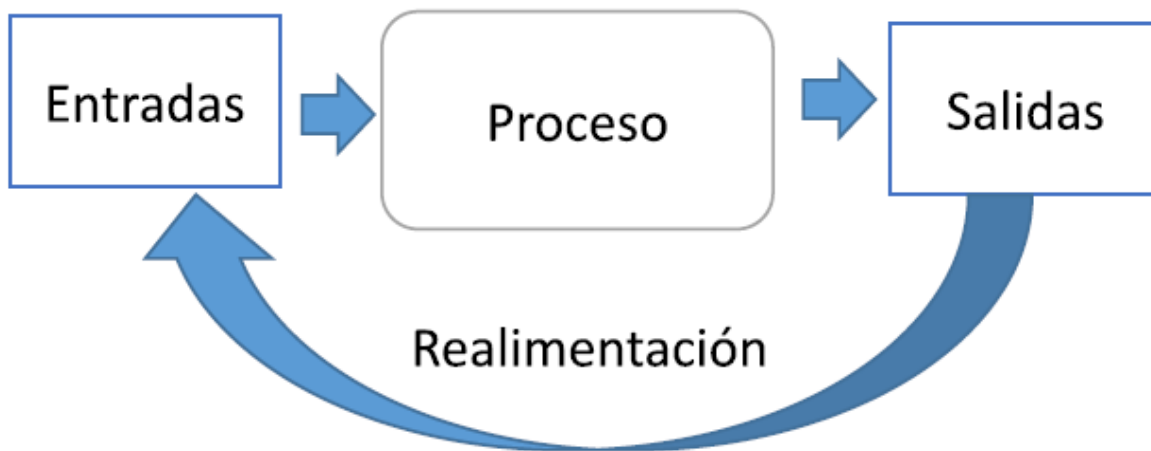


Un sistema es un conjunto de elementos que interactúan para lograr un objetivo específico

Elementos Básicos de un Sistema

- Meta / Objetivo: Beneficiar al usuario con calidad
- Entorno o Ambiente: Sistema Operativo, otros sistemas
- Componentes
- Entrada: Energía e información
- Proceso: Interacción entre componentes (Vista, lógica, datos)
- Salida: Información para el medio y otros sistemas
- Retroalimentación: Cambios del entorno
- Controlador

La función de un sistema es convertir información, energía, o materiales en algo planificado o producto para su uso dentro del sistema, fuera del sistema o ambos



Tipos de Sistemas

- Sistemas Complejos o Complicados

- Sistemas Automáticos o Inteligentes
 - **Automáticos:** Conjunto de componentes que pueden regular su propia conducta para lograr un funcionamiento predeterminado
 - **Inteligentes:** Sistema que aprende. Modifica su estructura para enfrentar cambios en el entorno

Innovación y Creatividad en Ingeniería de Sistemas

Innovación

- Introducción de novedades. Modificando elementos o generando unos desde cero

Creatividad

- Es lo relacionado o que involucra imaginación o ideas originales

Proceso para ser creativos

1. Preparación
2. Incuración
3. Inspiración
4. Validación

¿Cómo ser más creativo?

- Trabajar en equipo maximiza el proceso de creatividad debido a la colectividad
- Aplicar el Design Thinking

Design Thinking

- Empatía (Entender los problemas del usuario y reflexionar en soluciones que piensa en TODOS)
- Definir (Requisitos, necesidades, metas, opciones, posibles soluciones, orden a seguir)
- Idear (¿Cómo solucionar la problemática?)

- Prototipar
- Probar - Testing / Debugging



Unidad 1: Pensamiento Computacional

Pensamiento Computacional

- Los modelos y métodos computacionales nos ayudan a tener la iniciativa de resolver problemas y diseñar sistemas que ningún de nosotros por si solo seria capaz de superar
- El pensamiento computacional es esencial para cualquier persona. Nos ayuda a ver e intepretar el mundo de una forma distintas. De esta forma, obtener obtener herramientas para los problemas de la vida
- Saber como resolverlo y cual es la mejor forma de lograr ello
- Tener en cuenta cada uno de los factores que componen este problema
- Reducir la complejidad de lo que se nos dificulta a algo mas entendible y mas razonable para nuestro desarrollo cognitivo
- El pensamiento computacional nos lleva a pensar en muchas cosas, en pro de encontrar lo mejor, lo mas eficiente, lo mas practico, lo mas seguro y lo mas preparado posible para la resolución de problemas
- Hoy en día, este pensamiento, esta presente desde la situación mas cotidiana de la vida hasta en las mas complejas. Lo cual, ha hecho, que el pensamiento

computacional se haga presente en cada aspecto de la vida para volvernos mas eficientes

- No solo las areas involucradas en la tecnologia, la computación o la programación requieren de esta forma de pensar. Muchas otras disciplinas, necesitan tener un razonamiento logico similar para lograr respuestas y resultados a los problemas
- No es solo programar, es pensar mas alla y ser capaz de trabajar con la complejidad del problema
- Es la forma de pensar de los humanos, no de los computadores. Los computadores no son los que crean la magia, nosotros somos quienes logramos esto

-
1. Reducir
 2. Embeber
 3. Transformar
 4. Simular

Pilares

- Descomponer el problema en algo mas manejable
- Como se han resuelto problemas similares
- Detalles importantes
- Diseñar pasos simples

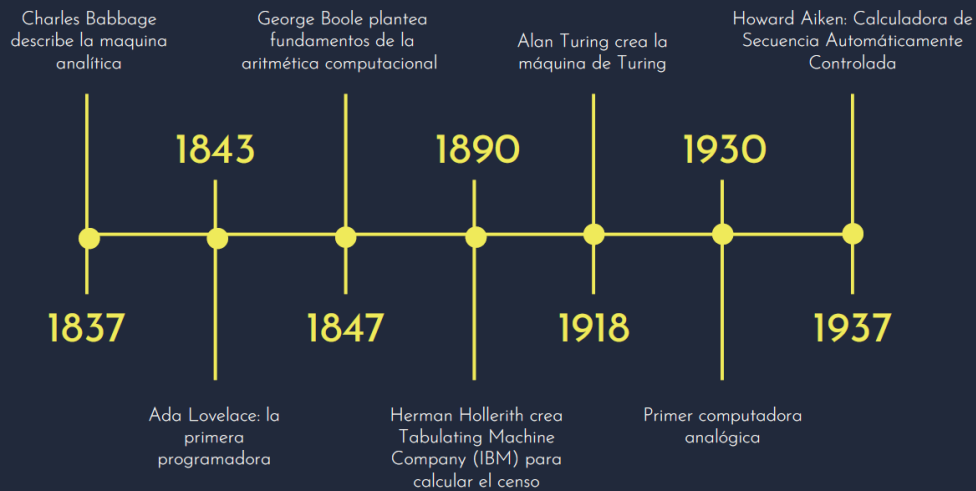
Análisis: De lo concreto a lo abstracto. Desarticular cada parte

Síntesis: De lo abstracto a lo concreto. Aprender lo que se estudia

Algoritmo: Diseñar una solución de forma secuencial y ordenada para lograr un resultado a través de pasos

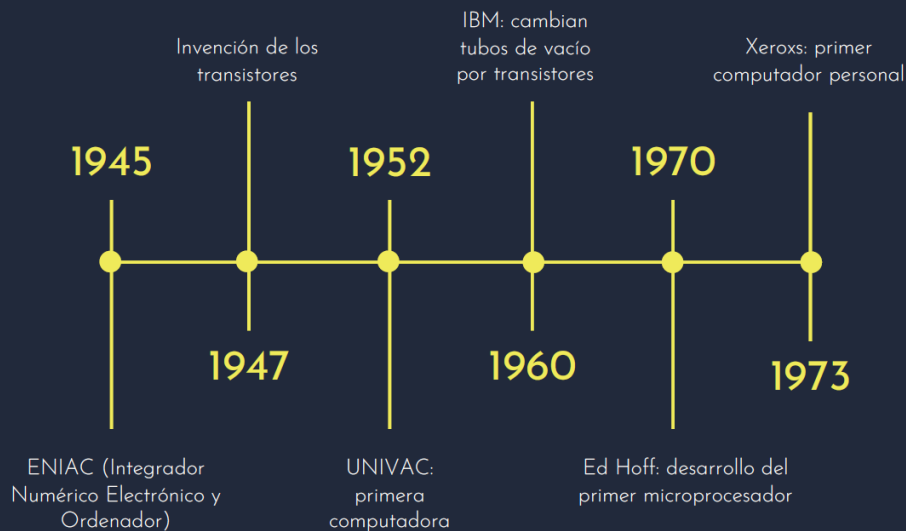
Historia de la Computación

De Charles Babbage al internet

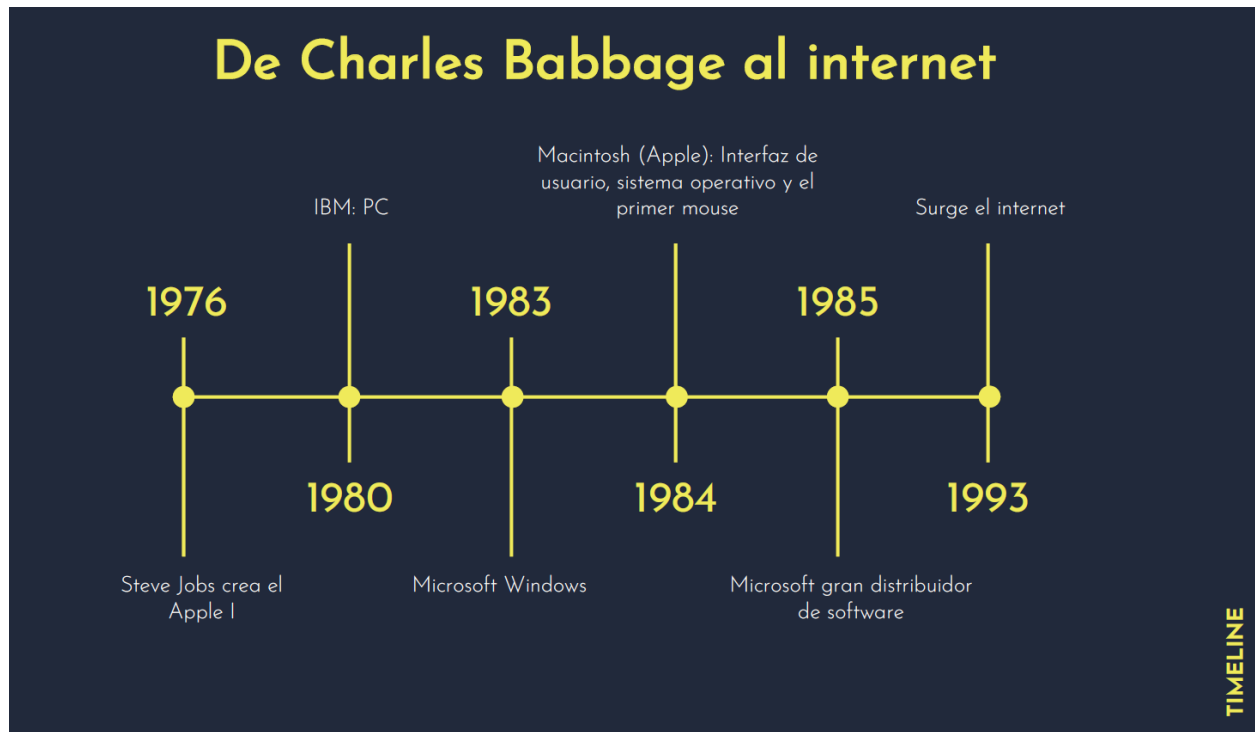


TIMELINE

De Charles Babbage al internet



TIMELINE



Unidad 1: PSP y TSP

Actividades para producir Software

1. Análisis
2. Diseño
3. Implementación
4. Validación
5. Mantenimiento

PSP (Proceso de Software Personal)

- Procesos centrados en el individuo que trabaja
- Calidad de entrega, diseño y código
- Gestiona, califica y aprueba sus productos
 - Planificar su trabajo

- Seguir el plan de trabajo
- Esforzarse por la calidad
- Gestionar el tiempo
- Mejorar

TSP (Proceso de Software en Equipo)

- Procesos que sincronizan el trabajo de un equipo
- Deben de realizar entregas balanceadas

Niveles de PSP

- **PSP 0:** Proceso actual, registro de tiempo y de defectos
 - **PSP 0.1:** *Estándares de codificación - Propuestas de Mejora*
- **PSP 1:** Estimación de tamaño y reporte de pruebas
 - **PSP 1.1:** *Planificar tareas y calendario*
- **PSP 2:** Revisión de diseño y código
 - **PSP 2.1:** *Planificación diseño*
- **PSP 3 o TSP:** Gestión de riesgos / Seguimiento proyecto

Herramientas de Planificación de Proyectos

- Microsoft Planner
- Process Dashboard (<https://www.processdash.com/>)
- Float (<https://www.float.com/>)
- Trello (<https://trello.com/>)
- Workbook (<http://workbook.net/>)

Roles

- Definir funciones claras, definir responsabilidades para llevar a cabo las actividades

- Los roles son realizados por un individuo o por un grupo de individuos que trabajan juntos como equipo
- Los roles describen cómo se comportarán los individuos en el contexto del proyecto

Gerente de Proyectos

- Planeación
- Buen desarrollo del proyecto
- Enlace entre grupo de desarrollo - cliente - gerencia de la empresa

Analista

- Elicitación e investigación de requisitos
 - Modelo de negocio
 - Procesos del negocio
 - Revisor requisitos
 - Analista sistemas
 - Diseñador UX

Diseñador

- UX - UI de la aplicación y la comunicación
 - Interfaz
 - Base de Datos

Desarrollador

- Diseño del código del software
 - Programador
 - Revisar código
 - Integrador

Administrador

- Administrador y configuración del proceso de ingeniería de software
 - Administrador de control de cambios
 - Administrador de configuraciones
 - Administrador de entregas
 - Administrador del proyecto
 - Revisor del proyecto

Probador / Tester

- Habilidades específicas para realizar pruebas



Unidad 1: Análisis de problemas y pasos para la solución de problemas



Un problema es una situación cuya respuesta es desconocida y se debe obtener a través de métodos científicos

Los problemas varían en al menos 4 cosas:

- Estructura
- Complejidad
- Dinamicidad
- Especificidad de dominio o abstracción

Estructura

Bien Estructurado

- Aplicación de un número limitado y conocido de conceptos, reglas y principios de un dominio restringido
- Estado inicial definido y estado final claro

Mal Estructurado

- No son predecibles ni convergentes
- Interdisciplinario y con múltiples dominios

Complejidad

Simple

- Pocas variables con poca relación
- Bien estructurado

Complejo

- Mayor número de variables que se relacionan de forma impredecible
- No conocemos todas las variables involucradas
- **Mal estructurado**

Complicado

- Necesidad de conocimiento científico

- Es retador
- Se requiere de innovación

Dinamicidad

Estable

- Son estables sus condiciones o factores en el tiempo
- **Bien estructurado**

Dinámico

- Las condiciones y factores van cambiando durante el tiempo
- Reformular la solución a medida que cambia
- Es **complejo**, por lo cual, se encuentra **mal estructurado**

Dominio

- El dominio depende al área en la que se encuentra el problema. Las soluciones varían según el contexto
-

Componentes de un problema

- **Estado inicial:** Motivación o causa
- **Estado final:** Efecto o beneficio
- **Métodos:** Pasar del estado inicial al final

Pasos para solucionar un problema

- Determinar el estado inicial
- Determinar el estado final
- Hacer una representación mental o modelo del problema, comprendiendo las relaciones entre las entidades del problema
- Definir el método para resolver el problema

- Aplicar el método definido
- Verificar el resultado
- Documentar el proceso de resolución del problema

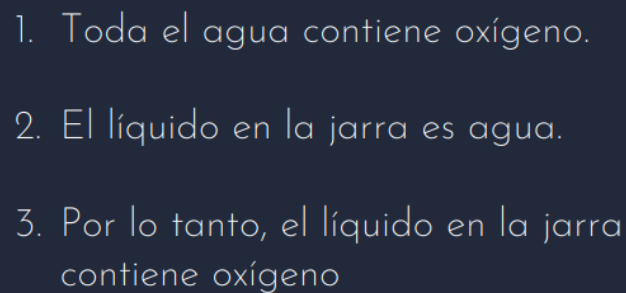
Métodos Científicos

- Métodos científicos simples:
 - Método deductivo
 - Método inductivo
 - Método analítico
 - Método sintético
- Métodos científicos mixtos:
 - Método deductivo – inductivo
 - Método analítico – sintético
 - Método hipotético – deductivo
 - Método histórico – comparativo

Investigación en Ing. de Sistemas

Tipos de Razonamiento

Deductivo: Es una conclusión que se obtiene lógicamente a partir de ciertas premisas

- 
1. Toda el agua contiene oxígeno.
 2. El líquido en la jarra es agua.
 3. Por lo tanto, el líquido en la jarra contiene oxígeno

Inductivo: Se analiza de lo particular a lo general. Sigue una secuencia temporal

1. Los clavos de hierro se oxidan en agua.
2. Las arandelas de hierro se oxidan en agua
3. Los cuchillos de hierro se oxidan en agua
4. Todos los objetos de hierro se oxidan en agua

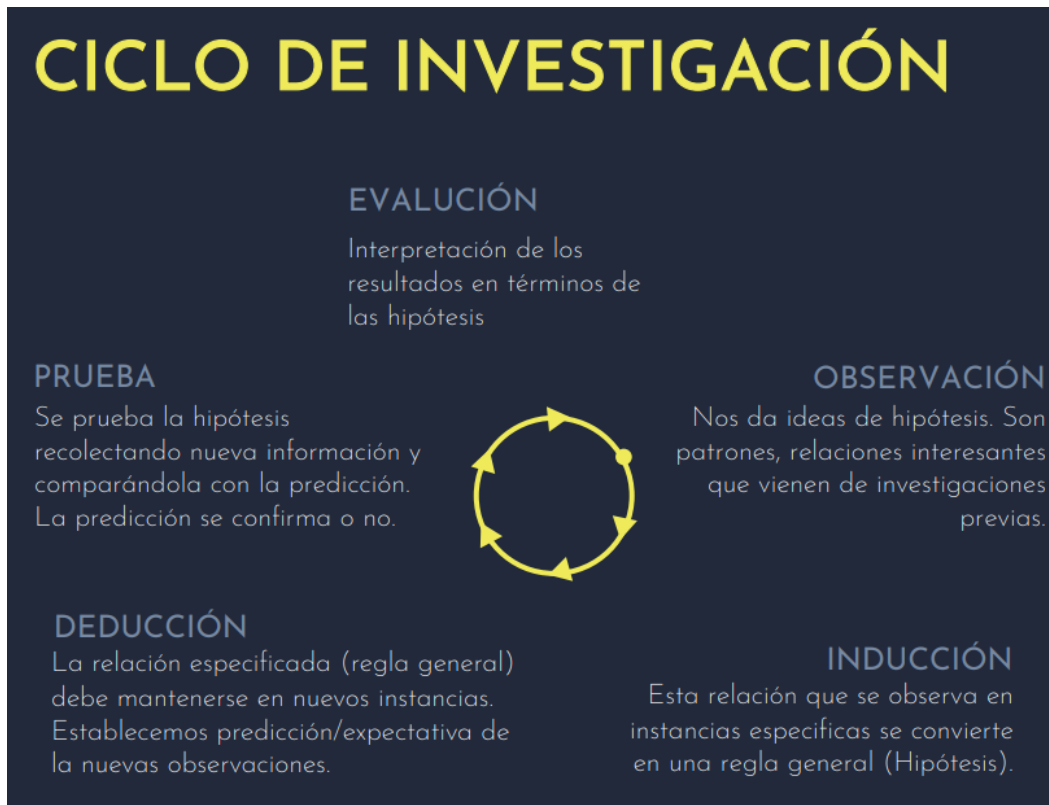
- El conocimiento científico está basado en evidencia empírica: información que se puede verificar con datos

Conocimiento

Observación: Representación precisa o imprecisa del mundo

Hipótesis: Declaración que describe un patrón o una relación general entre propiedades

Teoría: Una amplia explicación general de muchos fenómenos relacionados. Una teoría se construye a partir de hipótesis que están fuertemente apoyadas por evidencia empírica



Unidad 1: Problemas en la Ingeniería de Sistemas

Tipos de problemas Ing. de Sistemas

Problemas Informáticos

Analizar, identificar y proponer soluciones en el manejo de los sistemas de información, en pro de lograr el desarrollo de un proyecto

Problemas de Software

Desarrollar, mantener, evolucionar y operar un producto de software dependiendo de los factores del contexto

Problemas Computacionales

- Problema abstracto, permite establecer formalmente la entrada de un algoritmo y su salida

- Solución algorítmica
- Un problema abstracto pasa a ser concreto cuando se representa y califica en un lenguaje formal

Tipo de problema computacional

- Problemas de decisión: Respuesta puede ser si o no
- Problema de ordenamiento y búsqueda: Buscar un elemento que satisfice una propiedad. Establecer relación de orden entre los elementos
- Problema de optimización: Buscar una mejor solución

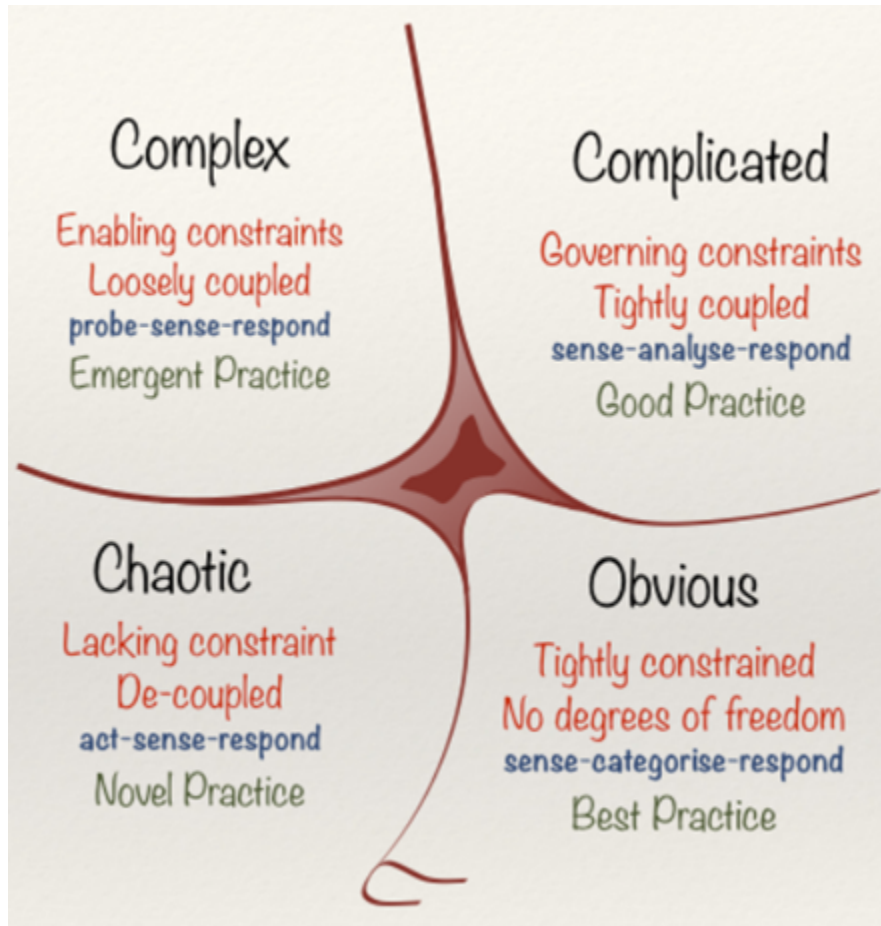
Tipo de terminación del algoritmo

- No computables: No existe algoritmo que lo haga
- No tratable: Requiere de muchos recursos, por lo cual, no es práctico. Se utilizan en cambio, heurísticas (aproximaciones o generalizaciones)
- Tratables: Algoritmo que busca la mejor solución

Tipo de modelo algorítmico

- Estructuras de datos básicas (listas unidimensionales, listas multidimensionales, pilas, colas, etc.).
- Algoritmos para manejo de cadenas de caracteres.
- Algoritmos de ordenamiento.
- Aritméticos y álgebra computacional.
- Algoritmos combinatorios.
- Teoría de números.
- Algoritmos de grafos.
- Etc.

Modelo Cinefyn



Unidad 2: Esquemas o representaciones gráficas



Un esquema es la representación gráfica o simbólica de cosas materiales o inmateriales. Que permite mostrar de forma visual lo que se interpreta de un problema o cómo se quiere solucionar éste

1. Observar
2. Tener una idea mental del problema
3. Representar el problema
4. Comunicar y analizar

Pensamiento visual

Es una forma de representar gráficamente lo que hemos logrado transformar en la mente al mirar

- Mirar
- Ver
- Imaginar
- Mostrar

Tipos de esquemas

Representación libre *

Representación general para entender los problemas, dando respuesta a:

- ¿Quién y qué?
 - ¿Qué ocurre a mi alrededor y dónde encajo?
 - ¿Quién está a cargo y quién más está involucrado?
 - ¿De quién es la responsabilidad?
- ¿Cuánto?
 - ¿Tenemos suficiente de X para que nos dure el tiempo necesario?
 - ¿Cuánto de X necesitamos para no parar la producción?
 - ¿Podemos disminuir el valor del parámetro X?
- ¿Cuándo?
 - ¿Cuándo se presentó el problema por primera vez?
 - ¿Cuándo presentamos una solución al problema?
- ¿Cómo?
- ¿Por qué?

Mapa mental *

- Extraer información

- Memorizar información
- Representar lógica y creativamente un tema
- Representar natural y coloridamente
- Presentar idea central y sus ideas relacionadas
- Estructurar contenidos

Mapa Conceptual *

- Representación del conocimiento
- Relaciones significativas entre conceptos en forma de proposiciones
- Dos o más términos conceptuales unidos por palabras para formar una unidad semántica

Componentes

- Concepto: Palabra expresa idea o hecho
- Palabra enlace: Verbo, preposición, conjunción o palabra que expresa relación
- Frases o proposiciones: Dos o más conceptos unidos por palabras enlace en una unidad semántica

Pasos para realizar uno

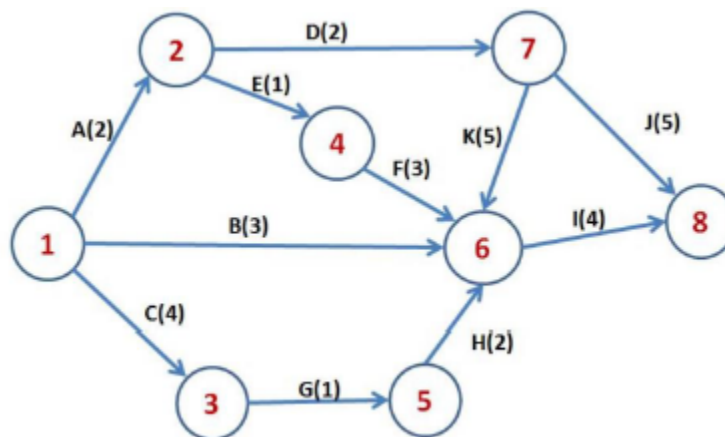
- Definir explícitamente la pregunta de enfoque
- Buscar información y listar conceptos más relevantes
- Identificar el concepto más general o nuclear
- Clasificar los conceptos por nivel de abstracción. Respetar jerarquía
- Hacer frases que incluyan los conceptos

Red semántica

- Representar conceptos y sus relaciones
- Presentar vínculos semánticos
- Formar agrupaciones de conceptos

- Establecer relaciones con conceptos previos
- Identificar los conceptos más relevantes
- Sintetizar ideas

Diagrama de Pert *



- Planear actividades
- Definir duración de las actividades
- Calcular rutas de trabajo optimizadas
- Realizar planificaciones más efectivas
- Identificar “cuellos de botella” en la ruta de trabajo
- Identificar dependencias entre las tareas

Material de apoyo

Gráfico Estadístico

Unidad 3: El papel de los requisitos para entender una necesidad

Requerimiento (*Request*): Acción y efecto de requerir. Necesidad o solicitud.

Requisito (*Requirement*): Circunstancia o condición necesaria para algo

Requisito de Software: Una condición o capacidad con la cual un software debe ser conforme (RUP)

Requisito Funcional: Como interactúa la solución con el mundo exterior

Requisito No Funcional: Los atributos de calidad de la solución

Requisito

- Un atributo necesario en un sistema, sentencia que identifica una capacidad, característica o factor de calidad de un sistema para otorgar un valor o utilidad para un cliente o usuario
- Esta bien definido
- Más específico que una necesidad, lo cual, satisface en mayor medida a un usuario o cliente

Dimensiones de los Requisitos

Ámbito

- Requisitos de software
- Requisitos de hardware
- Requisitos del sistema

Características

- Requisitos funcionales
- Atributos de calidad
- Restricciones de documentación, de interfaces, seguridad, estandarización, tecnológicas, regionalización

Audiencia

- Usuarios
- Análistas
- Desarrolladores
- Probadores

Tipos de Requisitos

- **Requisitos de Negocio:** Declaración de alto nivel de **metas, objetivos o necesidades**
- **Requisitos de Stakeholder:** Describe **necesidades** que el stakeholder tiene y interactúa con la solución
- **Requisitos de la solución:** Describe **características** necesarias para cumplir con los requisitos de negocio y de stakeholder
- **Requisitos de implementación:** **Capacidades** de la solución para pasar del estado actual de la organización al estado futuro deseado, pero que no serán necesarios cuando se de la transición

Requisitos como Sentencias de Lenguaje Natural

- Definir estos requisitos da paso a un modelo que será la base del diseño
- Representación en lenguaje natural para el entendimiento de los diferentes actores

Cada requisito de usuario debe tener

- Tipo de usuario beneficiado
- Estado deseable por el usuario (Objeto con un cualificador)
- Mecanismo de prueba del requisito

Los componentes de un requisito en el estilo tradicional

- Tipo de usuario
- Tipo de resultado (verbo)

- Objeto
- Cualificador (frase adverbiada)

El operador de un call center deberá visualizar el detalle de un cliente dentro de los dos siguientes segundos de haber hecho la consulta

Cómo escribirlos bien

- Frases simples y directas. Una sentencia activa, tan corta como sea posible sin exagerar
- Vocabulario limitado, evitar confundir a los lectores que no son técnicos o que son extranjeros
- Existen 3 clases de sentencias: **Entrada, Salida y Cambio de Estado**

Requisitos No Funcionales

- Restricciones como el tiempo de respuesta, precisión, seguridad, etc...
- Expresar de forma cuantitativa utilizando métricas objetivas

Propiedad	Medida
Rapidez	Transacciones por segundo
Tamaño	KB
Fiabilidad	Tiempo promedio entre fallas
Facilidad de uso	Tiempo de capacitación

Historias de Usuario

- Forma ágil de definir funcionalidades de la aplicación resaltando el punto de vista funcional y el usuario final
- Instrumento para levantar requisitos en el desarrollo de software, que emerge de la aparición de los marcos de trabajo de desarrollo ágil

- Yo como un [Rol (Quién)], necesito [Descripción de la funcionalidad (Qué)], con la finalidad de [Descripción de la consecuencia (Cómo)]

Yo como cliente registrado necesito realizar búsquedas de productos por categorías para identificar fácil y rápidamente lo que necesito comprar

- Criterios de calidad de una historia de usuario (INVEST)
 - Independiente
 - Negociable
 - Valuable
 - Estimable
 - Pequeña
 - Testeable



Buena contextualización del problema, para un entendimiento adecuado.
Evitar utilizar el sentido común y hacer supuestos

Unidad 3: Creación del Prototipo

- No hay alternativa de no diseñar. Hay un buen o mal diseño y ya
- Diseño de UX y UI
- Evolución de los diseños
- Representación navegable del producto final
- Calidad entre media y baja

Tipos de Prototipos

Sketches (Bocetos)

- Dibujo rápido sin muchos detalles
- Estático, baja calidad, a mano
- Entregable en imagen, en papel o digital

Wireframe

- Estático
- Columna vertebral del diseño
- Entregable: imagen

Mockup

- Representación estática
- Calidad media o alta
- Entregable: imagen

Prototipado

- Versión inicial de un sistema
 - Demostrar conceptos
 - Tratar opciones de diseño
 - Encontrar más sobre el problema y sus soluciones
- Controlar costos y los interesados en el sistema experimenten por anticipado

Características

- Funcionalidad limitada
- Poca fiabilidad
- Características de funcionalidad pobres
- Alto grado de participación del usuario para proponer
- Alto grado de participación del analista de sistemas, para indicar los requisitos
- Da mayor conocimiento al usuario y analistas

Tipos de Prototipos - I

- **Interfaz de usuario:** Modelos de pantallas
- **Funcional (operacional):** Implementa algunas funciones
- **Modelos de rendimiento:** Rendimiento de una aplicación crítica

Tipos de Prototipos - II

Rápido o desechable

- Sirve al análisis y validación de requisitos
- Especificación del sistema y se desecha el prototipo
- Se desarrolla con un paradigma diferentes la app
- Problema: No se desecha el prototipo y termina siendo el sistema final

Evolutivos

- Sistema relativamente simple, con requisitos importantes y mejor conocidos
- Cambia cuando se descubren nuevos requisitos
- Finalmente, se convierte en el sistema requerido
- Usado en el desarrollo de sitios web y apps de comercio electrónico

Tipos de Prototipos - III

- Vertical: Desarrolla completamente alguna de las funciones
- Horizontal: Desarrolla parcialmente todas las funciones

Recursos

- [Cacoo](#)
- [Balsamiq](#)
- [wireframe.cc](#)
- Herramientas CASE

- wireframe sketcher
- Enterprise Architect

Unidad 3: Modelado del sistema



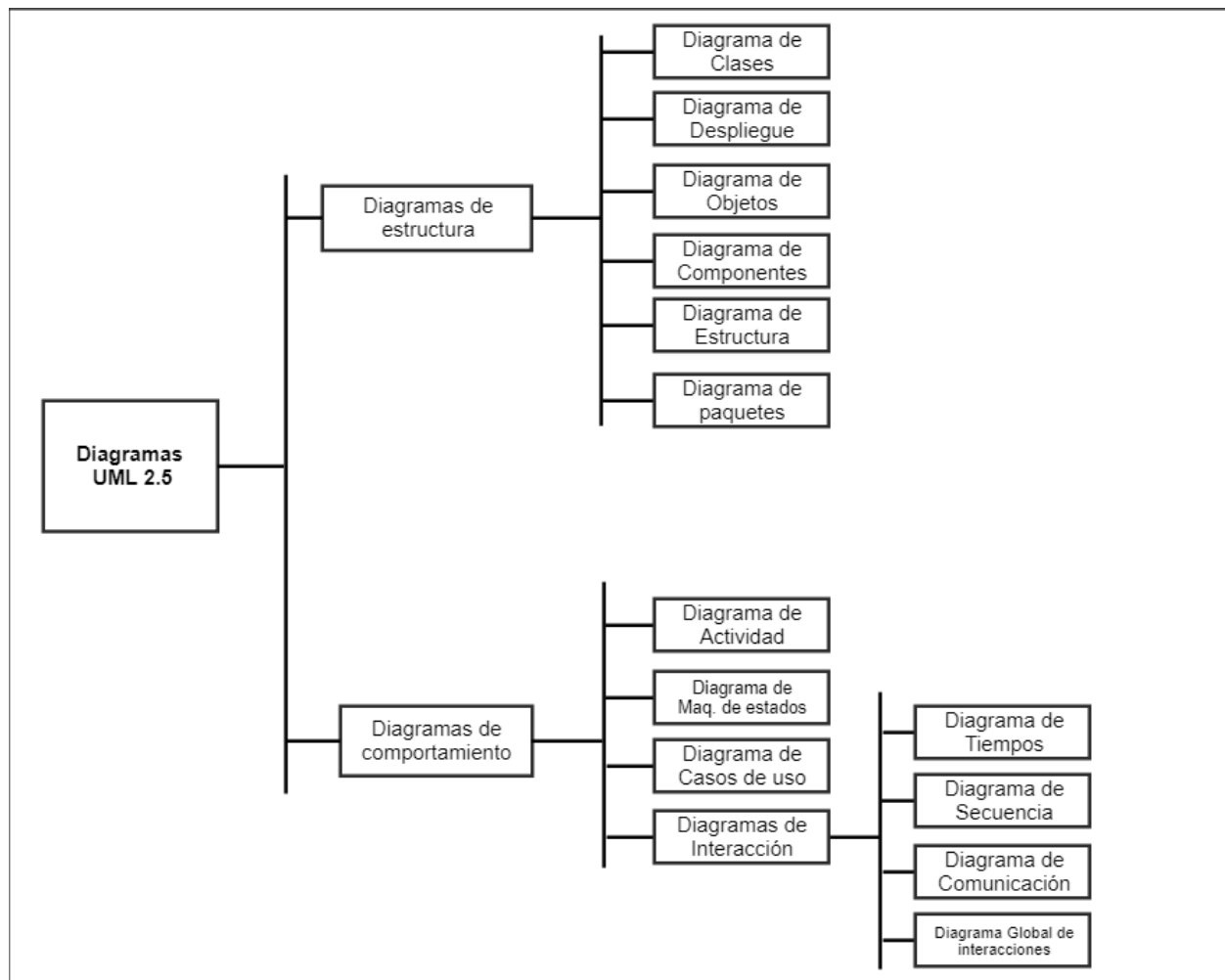
Modelar es **representar de manera abstracta, gráfica, escrita, conceptual, visual, o física un problema. Simplifica un objeto del mundo real** y que permite usarlo como **base del desarrollo**

UML (Unified Modeling Language)

- Representación gráfica para representar sistemas
- 13 diferentes tipos de diagramas para modelar sistemas de software
- Representar la solución de software que se desea implementar

¿Para qué?

- Discusión del sistema propuesto
- Documentar un sistema existente
- Descripción detallada
- Representar lo que hace
- Explicar requisitos



Perspectivas de un Modelo

- **Perspectiva externa:** Modelar contexto o entorno del sistema
 - Diagrama de Casos de uso
- **Perspectiva de interacción:** Modelar interacción del sistema, su entorno y componentes
 - Diagrama de Secuencia
- **Perspectiva estructural:** Modelar la organización del sistema, estructura de datos
 - Diagrama de Clases

Unidad 3: Diagrama de casos de uso

Escenarios

- Describe **situaciones** y sus relaciones con el sistema a ser construido
- Utilizado para describir interacciones entre componentes del sistema
- Lenguaje natural semi-estructurado

Características de un escenario

- **Título:** Indicativo de lo que sucede
- **Objetivo:** Trata de la satisfacción de un objetivo
- **Contexto:** Entorno de la situación
- **Actores:** Aparatos u organizaciones implicados
- **Recursos:** Elementos necesarios
- **Restricciones:** Limitantes para ejecutar la situación
- **Episodios:** Acciones concretas
- **Tiempo:** Momento específico
- **Lugar:** Contexto geográfico
- **Independiente a manera individual:** Necesita ser entendido por si solo
- **Inter-relacionada a manera global:** Se relaciona con otras situaciones

Título: La tienda confirma ficha de registro.

Objetivo: Verificar si la información de la ficha de registro está correcta.

Contexto: Cliente entrega ficha de registro y presenta cédula de ciudadanía para la tienda.

Actores: Tienda, Cliente.

Recursos: Cédula de ciudadanía, ficha de registro.

Restricción: El Cliente debe poseer una Cédula física válida.

Episodios:

- La tienda verifica el número de cédula presente en la ficha de registro con la cédula física del cliente.
- La tienda verifica la dirección y el teléfono de contacto presentes en la ficha de registro, llamando al teléfono de contacto.
- La tienda llena los campos de la ficha de registro no informados por el cliente con la sigla 'NE' (No Existente).

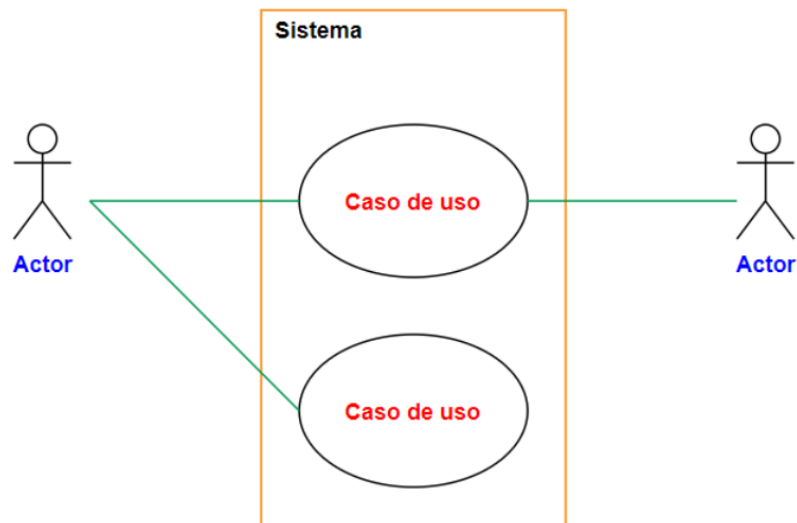
Casos de Uso

- Se crean a partir de los escenarios
- Modelan comportamiento del sistema (Requisitos funcionales)
- Interacciones entre sistema y entorno
- Narran historia de como interactua usuario final con el sistema en ciertas situaciones, usando lenguaje semi-estructurado
- Ilustran el software o sistema desde el punto de vista del usuario final

Notación I

Elementos:

- Caso de uso
- Actor
- Comunicación
- Entorno del sistema



Plantilla descripción Caso de Uso

- Cada caso se debe describir en lenguaje natural utilizando la plantilla

Nombre	
Autor	
Fecha	
Descripción	
Actores	
Incluye	
Excluye	
Hereda de	
Precondiciones	
Poscondiciones	
Flujo normal de eventos	
Actor	Sistema
Flujo alternativo de eventos	
Actor	Sistema

- Ejemplo de la plantilla

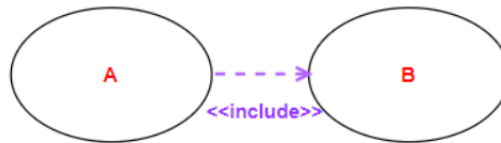
Nombre	Reservar libro
Autor	Juanito Pérez
Fecha	13/01/2020
Descripción	El socio puede solicitar la reserva de un libro para posteriormente solicitarlo prestado
Actores	Socio
Incluye	--
Excluye	--
Hereda de	--
Precondiciones	El socio no tiene ninguna reserva a su nombre
Poscondiciones	El socio tiene una reserva de un libro a su nombre. El libro tienen una nueva reserva
Flujo normal de eventos	
Actor	Sistema
1. El socio solicita la reserva (código libro, fecha) 5. El socio confirma la reserva	2. El sistema comprueba que el socio no tiene reserva 3. El sistema comprueba que el libro está libre para la fecha solicitada 4. El sistema solicita confirmación de la reserva 6. El sistema realiza la reserva
Flujo alternativo de eventos	
Actor	Sistema
--	--

Notación II

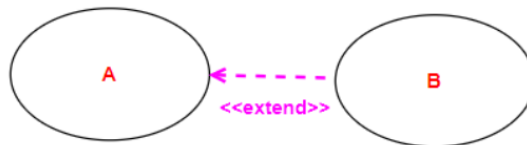
- Hay otras situaciones que se describen a través de este diagrama de casos que son particulares y también poseen su determinada notación

Relaciones entre casos de uso

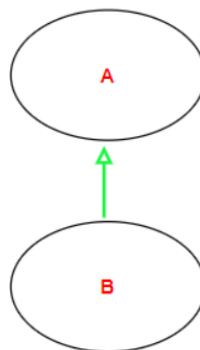
- Inclusión: `<<include>>` Un evento que debe ser obligatorio para que se de otro. Evitando describir el mismo flujo varias veces



- Extensión: `<<extend>>` Un evento que se puede dar pero que no es del todo obligatorio. Representa subflujos, que ocurren bajo ciertas condiciones

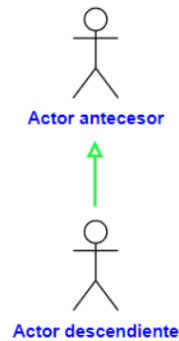


- Herencia: Sobreescibir una acción en otra para realizar una modificación en el flujo



Relaciones entre actores

- Herencia: Un actor descendiente puede jugar todos los roles del actor antecesor



Elementos:

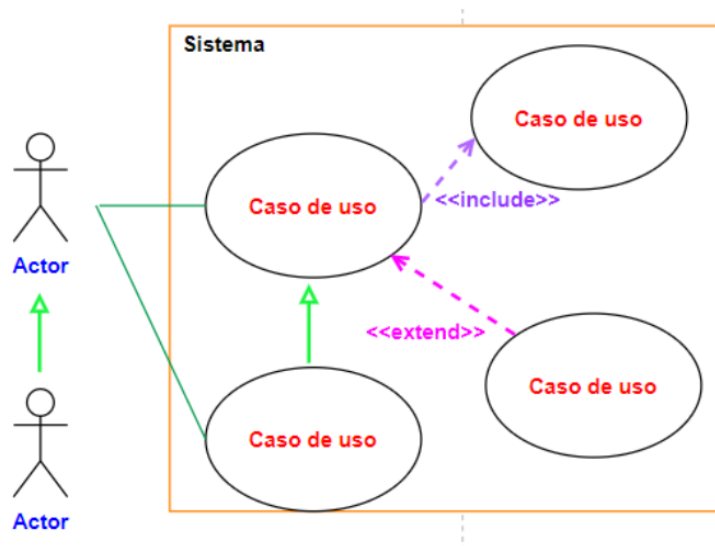
- Caso de uso
- Actor
- Comunicación
- Entorno del sistema

Relaciones entre casos de uso:

- Inclusión
- Extensión
- Herencia

Relaciones entre actores:

- Herencia

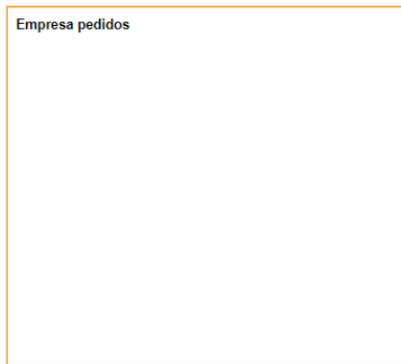


Construir un Diagrama de Casos de Uso

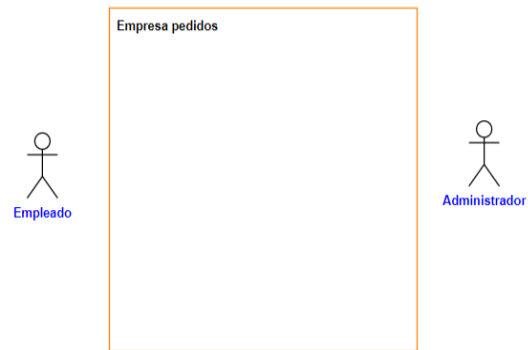
1. Diagrama de Contexto

- 1.1 Identificar entorno o limites del sistema a desarrollar
- 1.2 Identificar actores externos que interactúan con el sistema

1.1.

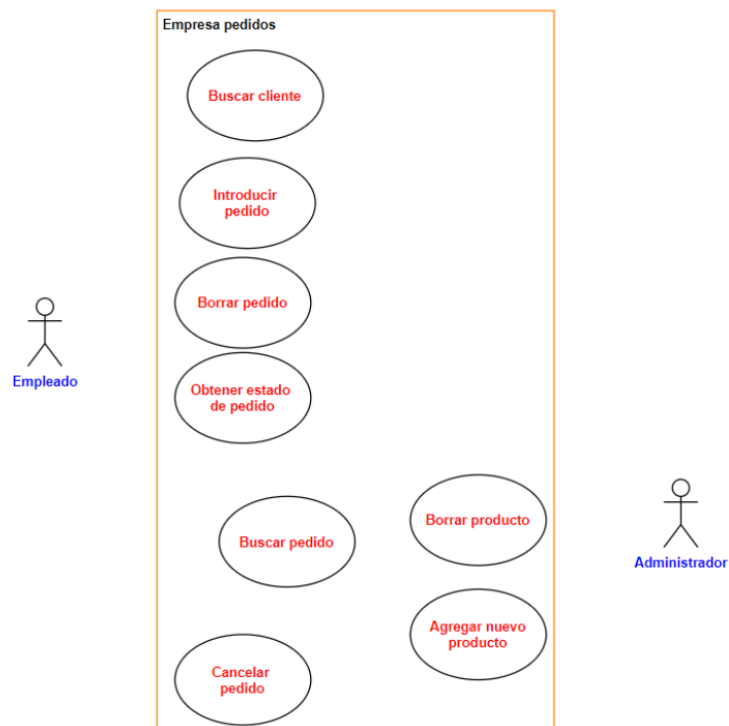


1.2.

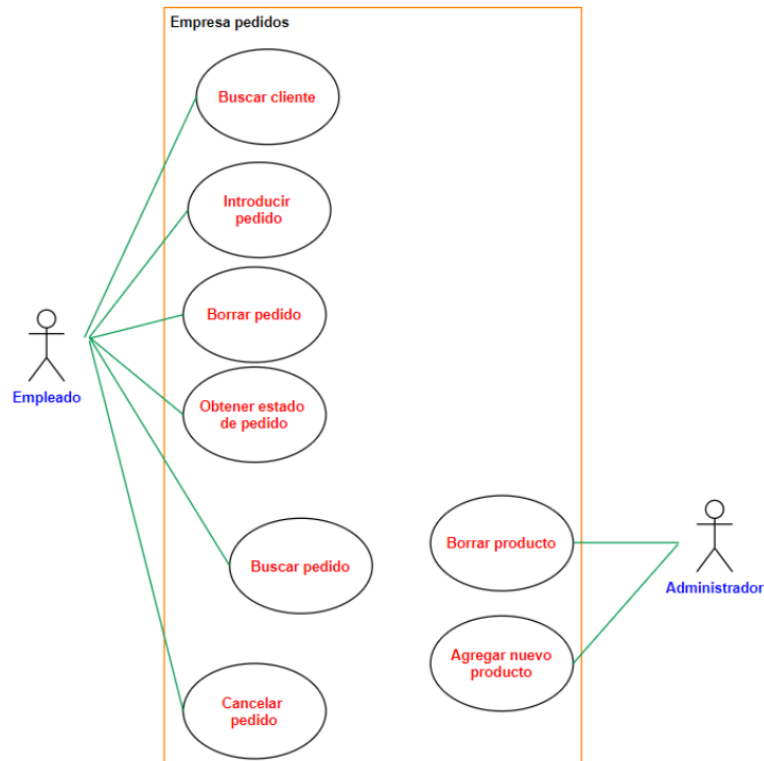


2. Diagrama de Casos de Uso Inicial

2.1 Identificar los casos de uso funcionales



2.2 Establecer relaciones entre actores y casos de uso

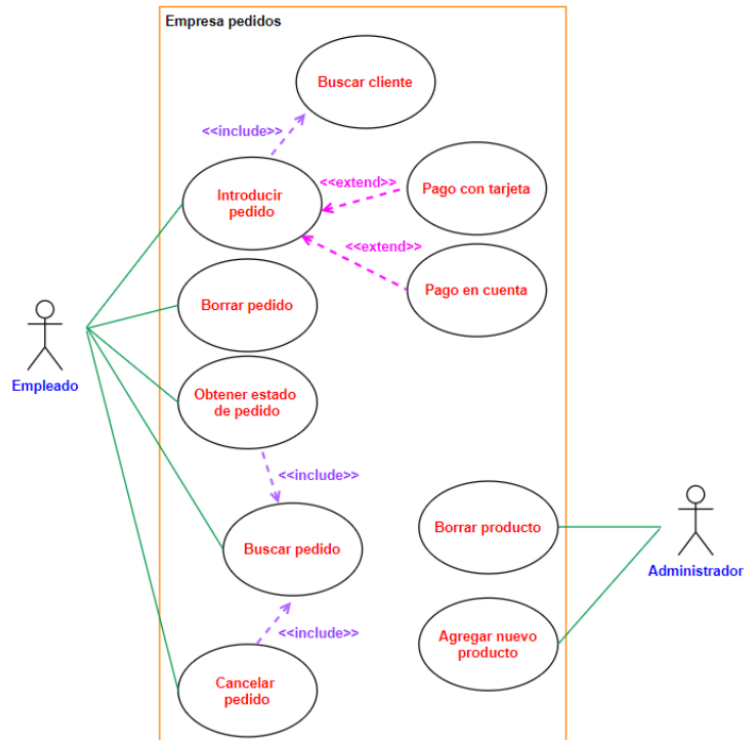


3. Diagrama de Casos de Uso refinado

3.1 Agregar las relaciones entre casos de uso (inclusión, extensión y herencia) si las hay

3.2 Agregar relaciones entre actores, si las hay

3.3 Realizar mejoras y complementar



4. Plantillas de descripción

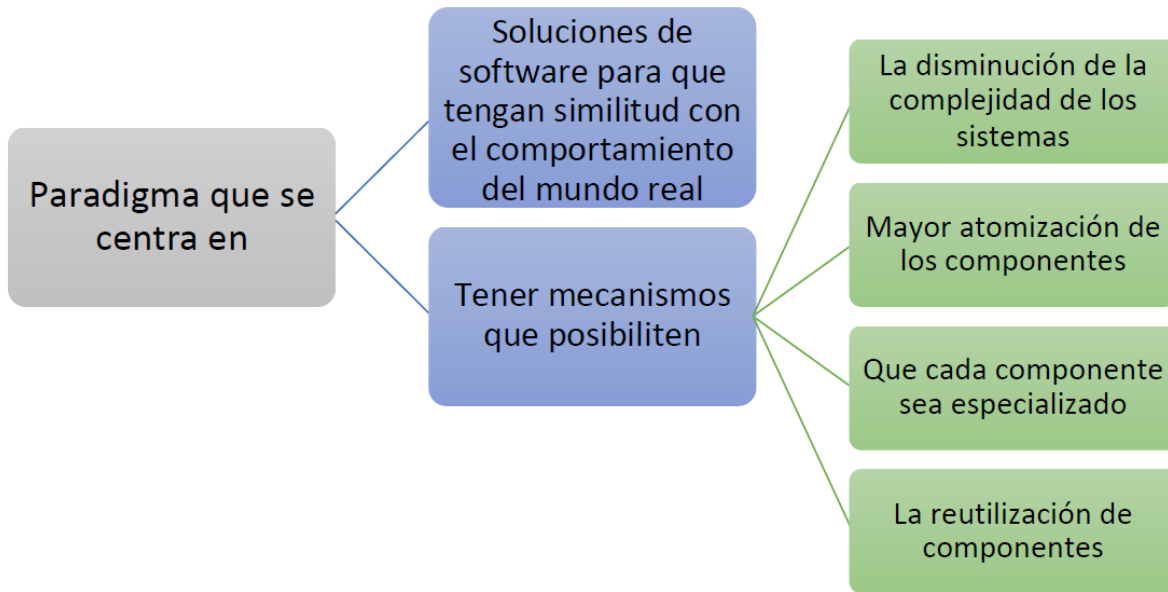
4.1 Completar plantilla para cada caso de uso

Unidad 3: Principios básicos del modelo de objetos

Orientación a Objetos (OO)

Es la agrupación de un conjunto de técnicas que nos permiten diseñar, desarrollar y mantener más fácilmente programas complejos

El propósito de la OO es identificar conceptos relevantes presentes en un problema para encapsularlos



Principios Básicos

1. Es más simple resolver un problema si lo divido en partes más simples
2. Estas partes simples pueden servir para resolver otro problema
3. Las partes que divido pueden ser reales o abstractas
4. Las partes en las que he dividido poseen atributos, acciones propias y relaciones con otras partes



Un objeto es cualquier cosa real o abstracta que puedes describir

Clase

- Un concepto que engloba varios objetos o los representa en uno solo, es una **clase**.
- Definición de las propiedades y comportamiento de un tipo de objeto concreto
- La instanciación es la lectura de esas definiciones y la creación de un objeto a partir de ella



Una clase posee atributos y acciones, que son las que definen como serán cada uno de los objetos en particular

Clase Persona

Atributos

- Nombre
- Edad
- Género
- Profesión

Acciones

- Estudiar
- Trabajar
- Hacer Deporte

Objeto Juan

Atributos

- Nombre: Juan Quintero
- Edad: 28 años
- Género: Masculino
- Profesión: Médico

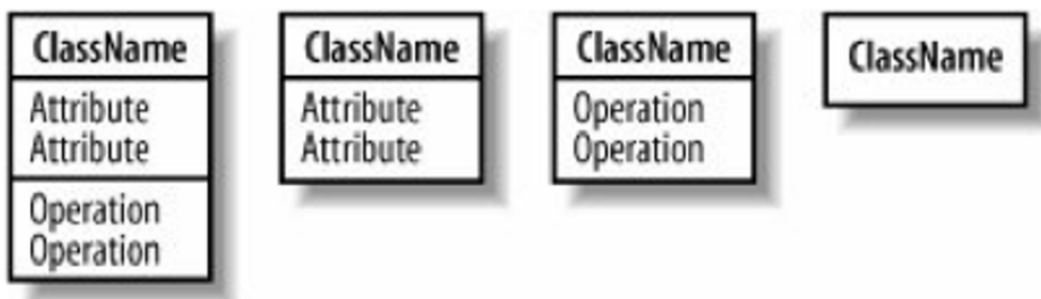
Acciones

- Estudia (Neurología)
- Trabaja (IPS Y)
- Hace Deporte (Tenis)

Una clase se describe por:

- Nombre
- Atributos (Características)
- Métodos (Acciones)

4 diferentes formas de mostrar una clase*



Para mas detalles sobre la modelación de la orientación a objetos en la programación

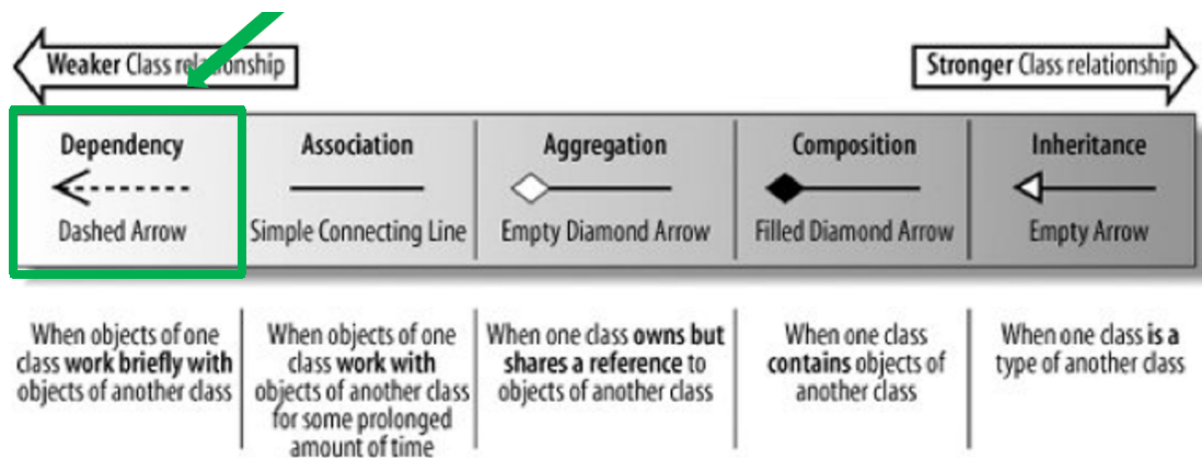


La diferencia entre una clase y un objeto:

- La clase es la plantilla que define los atributos y métodos de los objetos
- El objeto es un elemento creado a partir de esa plantilla de la clase

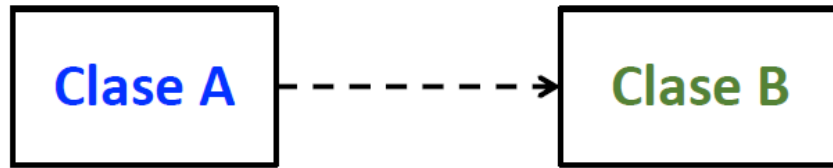
Unidad 3: Diagrama de Clases

Tipos de Relaciones



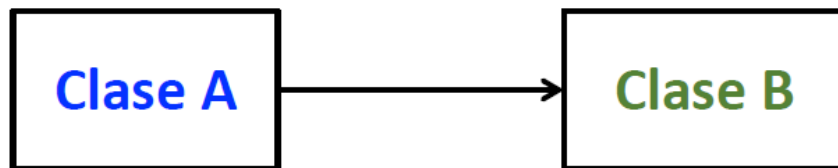
Dependencia

- Es una relación de uso: un objeto de una clase A usa objetos de una clase B o invocar métodos de la clase B
- Si se producen cambios en B, se pueden producir en A
- Relación entre un cliente y un proveedor de un servicio
- Se da cuando le pasamos la referencia de otro objeto a un método
- Un ejemplo, es cuando importamos en Java clases que nosotros no desarrollamos, pero podemos usar, como `Scanner`



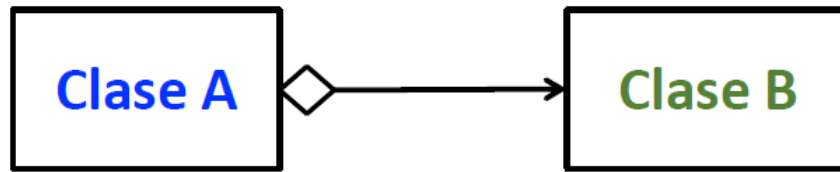
Asociación

- La asociación representa una relación **permanente** entre clases
- Puede tener dirección uni-direccional y bi-direccional
- Tiene cardinalidad
 - Uno a uno
 - Uno a muchos
 - Muchos a muchos
- Una clase A referencia a uno o varios objetos de la clase B
- La clase A tiene un atributo de la clase B
- Relación del tipo “tiene un”
- Si una clase trabaja con un objeto de otra clase, entonces la relación entre esas clases es un gran candidato para la asociación



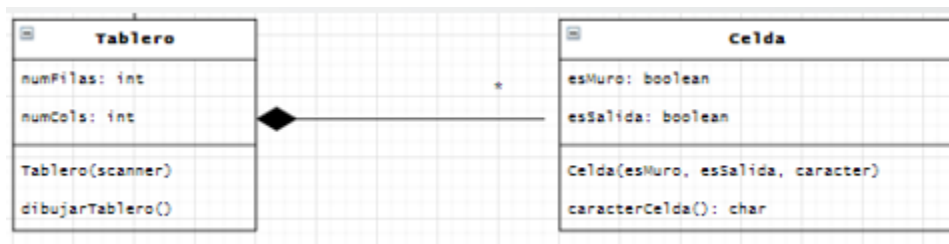
Agregación

- Los objetos poseen diferentes partes. Una clase A (el todo) se compone de otros objetos de la clase B (las partes)
- Es una versión más fuerte de la asociación y se utiliza para indicar que una clase posee y puede compartir los objetos de otra clase



Composición

- Es una forma de agregación
- Representa una relación muy fuerte: la clase compuesta no se concibe sin las que la compone
- Las partes **comparten el tiempo de vida** del todo
- Algunos autores dicen que no es necesario diferencia de la agregación
- En UML hayo forma de expresarla



Herencia

- Realizar enunciados generales que se aplican a todos los miembros de la clase
- La clase B hereda todos los atributos y métodos de la clase A
- Es una buena práctica de diseño, permite minimizar los costos asociados a las actualizaciones o modificaciones. Además de reutilizar código
- En Java se utiliza la palabra `extends` para referirse a esto, vamos a heredar de una clase padre sus atributos y métodos
- La herencia es transitiva

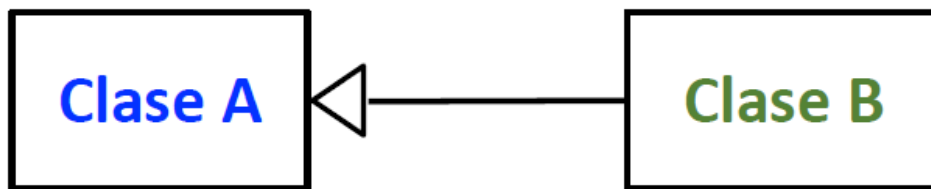
Beneficios

- Reutilización de código
- Compartir el código

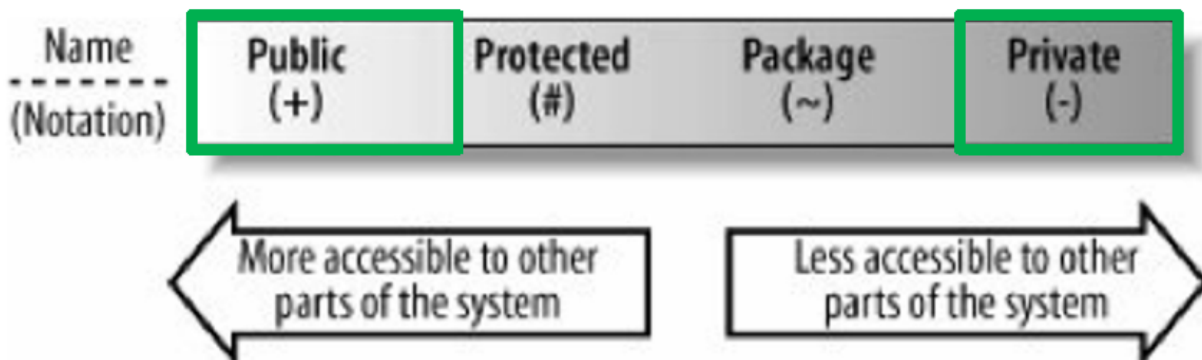
- Poliformismo
- Ocultamiento de la información

Costos

- Velocidad de ejecución (minimo)
- Tiempo para procesar invocaciones de los métodos (mínimo)
- Complejidad de los programas



Visibilidad de Atributos



Visibilidad Pública

- Se puede acceder al miembro de la clase desde cualquier lugar

Visibilidad Privada

- Solo se puede acceder al miembro de la clase desde la propia clase


Visibilidad Protegida

- Utilizada normalmente en herencia. Esta visibilidad permite que clases que poseen herencia de otra clase, puedan acceder directamente a los atributos de esta clase.

Esto será siempre y cuando ocurra herencia, ya que, se sigue protegiendo la visibilidad del atributo por fuera de la clase originaria

Unidad 3: Diagrama de clases a código

- Recurso para comprender mejor como implementar un diagrama de clases a nivel de código en un lenguaje de programación, en este caso Java
 - Java Programming - Class Diagram

 Java

Unidad 3: Diagrama de secuencias

Diagramas UML

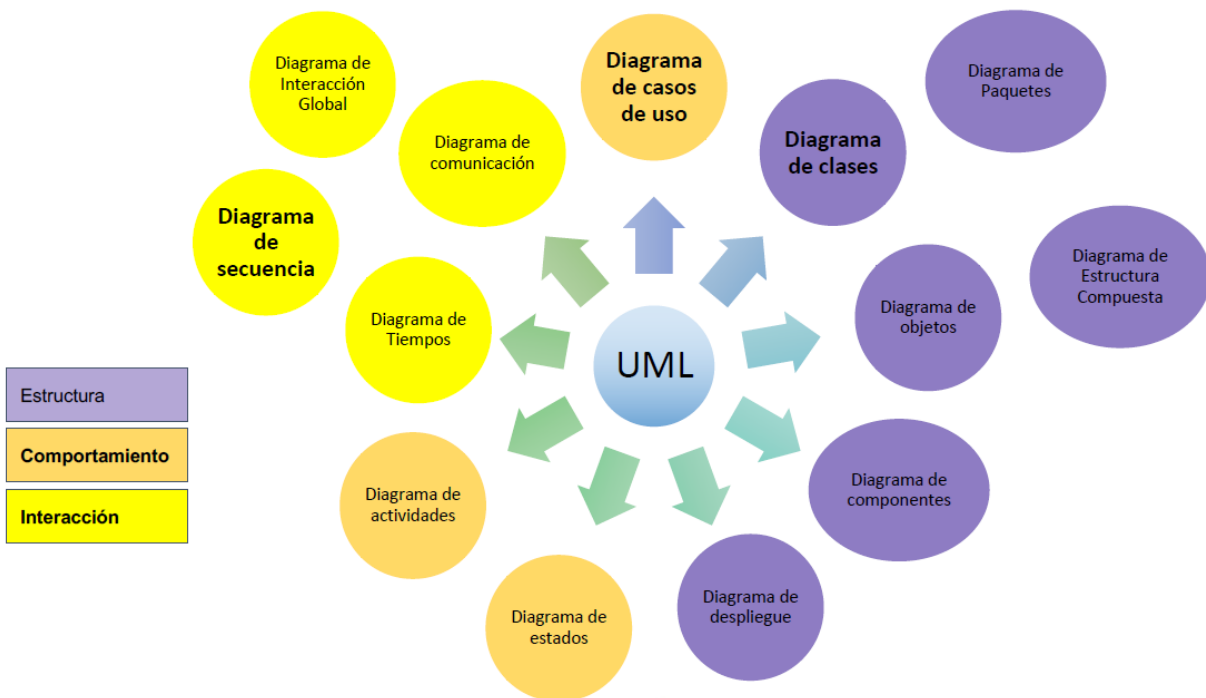


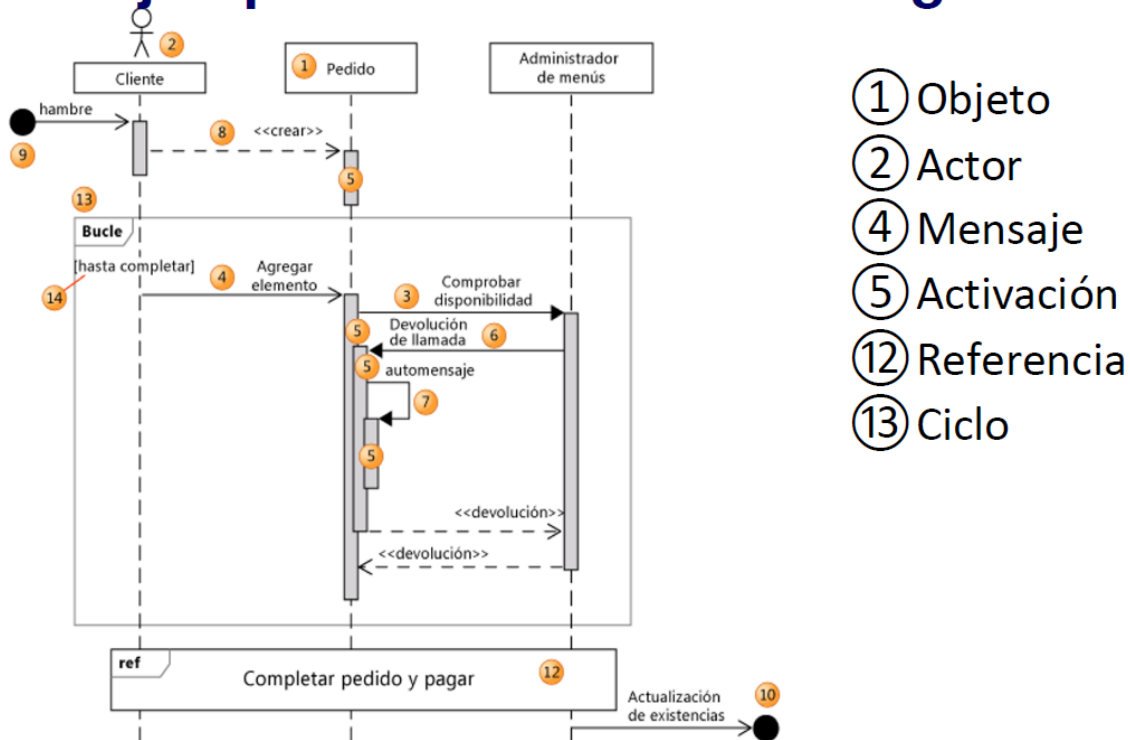
Diagrama de secuencias

- Modela la secuencia de interacciones entre distintos objetos del sistema para lograr alguna tarea ya sea un escenario de un caso de uso, la lógica de un método (a

nivel de programación en funcionamiento) o la lógica de un servicio

- Detalla cómo y cuándo se realizan las operaciones mediante el envío de mensajes
- Se organizan en función del tiempo
- Los objetos que se involucran se enumeran de izquierda a derecha, secuencialmente estos vayan interviniendo en el proceso
- Se hace al menos un diagrama de secuencias por cada caso de uso, ya que, este diagrama se enfoca en un solo escenario

Ejemplo 1 – Partes de un diagrama



Partes Diagrama de secuencias

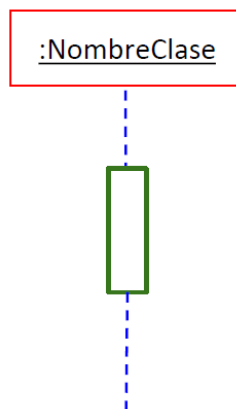
Objeto y línea de vida

- El objeto se representa con una caja rectangular, que lleva el nombre de la clase, y una línea discontinua que sale de la caja (**conocida como línea de vida**)



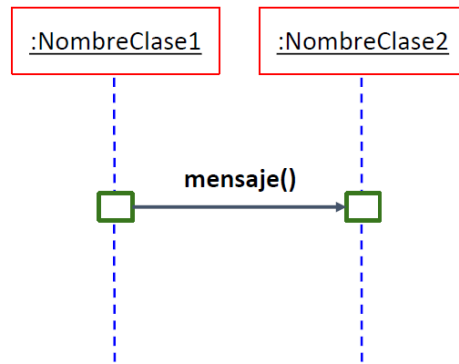
Activación

- Junto con la línea de vida del objeto, se encuentra un rectángulo que representa la activación o ejecución de una operación que realiza este objeto
- La longitud de este, representa la duración de la activación o ejecución



Mensaje

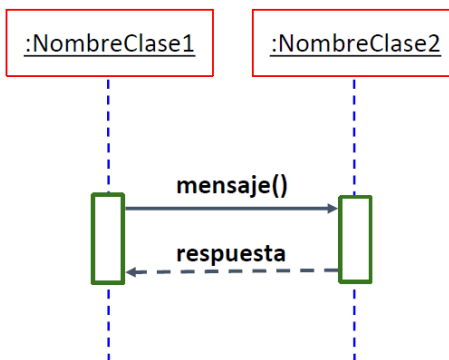
- Un mensaje que va de un objeto a otro, se representa con una flecha que va de la línea de vida de un objeto a la de otro



- Un objeto puede enviarse un mensaje a sí mismo, desde su línea de vida hacia esta misma

Tipos de Mensaje

- **Mensaje simple:** representa la invocación de un método
- **Mensaje de respuesta:** es la respuesta de un objeto luego de ejecutar un método (que no es void)



Creación de un objeto en la secuencia

- Cuando una secuencia da por resultado la creación de un objeto, tal objeto se representará de la forma usual: como un rectángulo con nombre
- Este objeto debe ubicarse inmediatamente se coloque el mensaje que creará el objeto, es decir, `Crear()`, una operación constructor genera un objeto. Se puede usar `<<crear>>`

3 Principios básicos

1. Identificar a los objetos involucrados
2. Definir la responsabilidad de cada objeto
3. Mostrar el ciclo de vida de cada objeto



Los diagramas de secuencia salen de los casos de uso y las clases

Unidad 4: Mantenimiento

- **Modificación de un producto de software después de la entrega**
- Se realiza para corregir errores, **mejorar el rendimiento**, entre otros atributos

Componentes claves del Mantenimiento

1. Identificar estrategias de mantenimiento
2. Determinar actividades para el mantenimiento y sus responsables
3. Trazar el plan
4. Ejecutar el plan

Tipos de Mantenimiento

Preventivo

- Preparar el producto para nuevas demandas y capacidad
- Mejorar la fiabilidad para evitar problemas o errores en el futuro
 - Uso eficiente del espacio de almacenamiento
 - Cambiar código para mayor velocidad en ejecución

Correctivo

- Diagnosticar y corrección de errores que normalmente los usuarios encuentran

Adaptativo / Continuo

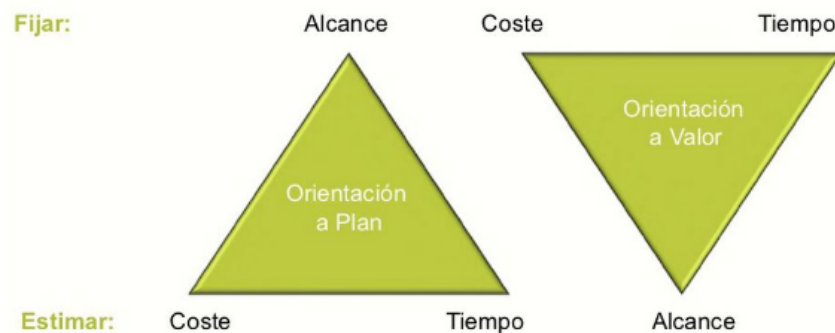
- Adaptaciones requeridas por la evolución del entorno
- Cambios por mejoras a la funcionalidad del software
- Implementar nuevos requisitos

Beneficios

- Ahorra tiempo cuando se anticipan situaciones futuras que pueden desencadenar cambios en el sistema
- Incrementa ingresos al entregar sistemas estables

Ética Profesional Creación de Software

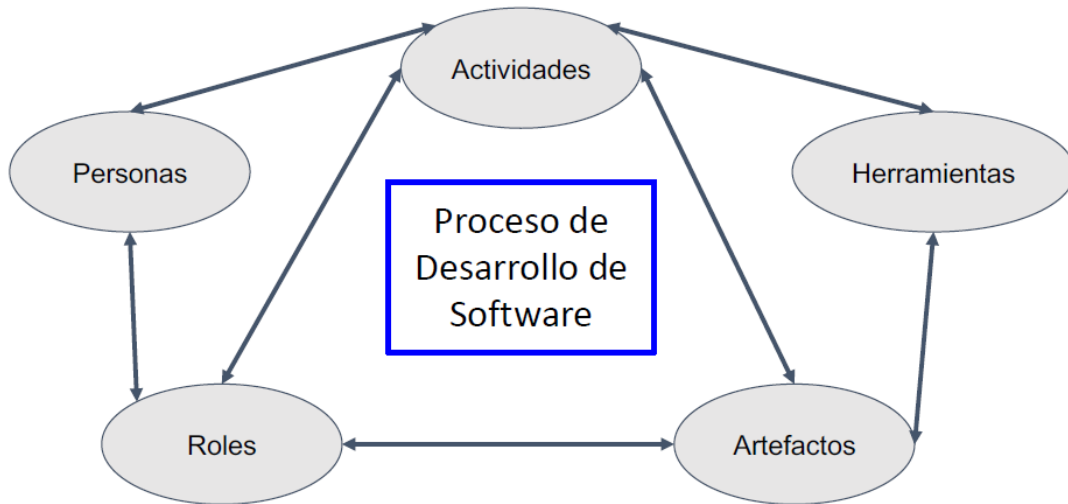
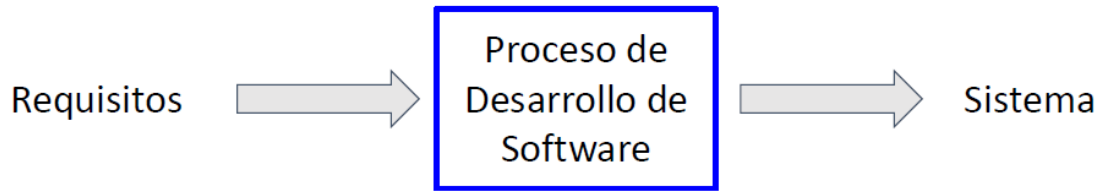
- Se espera que un ingeniero de software tenga un compromiso con la calidad del software como parte de su cultura
- Principio básico de la creación de cualquier producto



- [Software Engineering Code of Ethics](#)

Unidad 4: Procesos de Desarrollo de Software

- El proceso de Desarrollo de Software permite pasar de los requisitos al sistema implementado



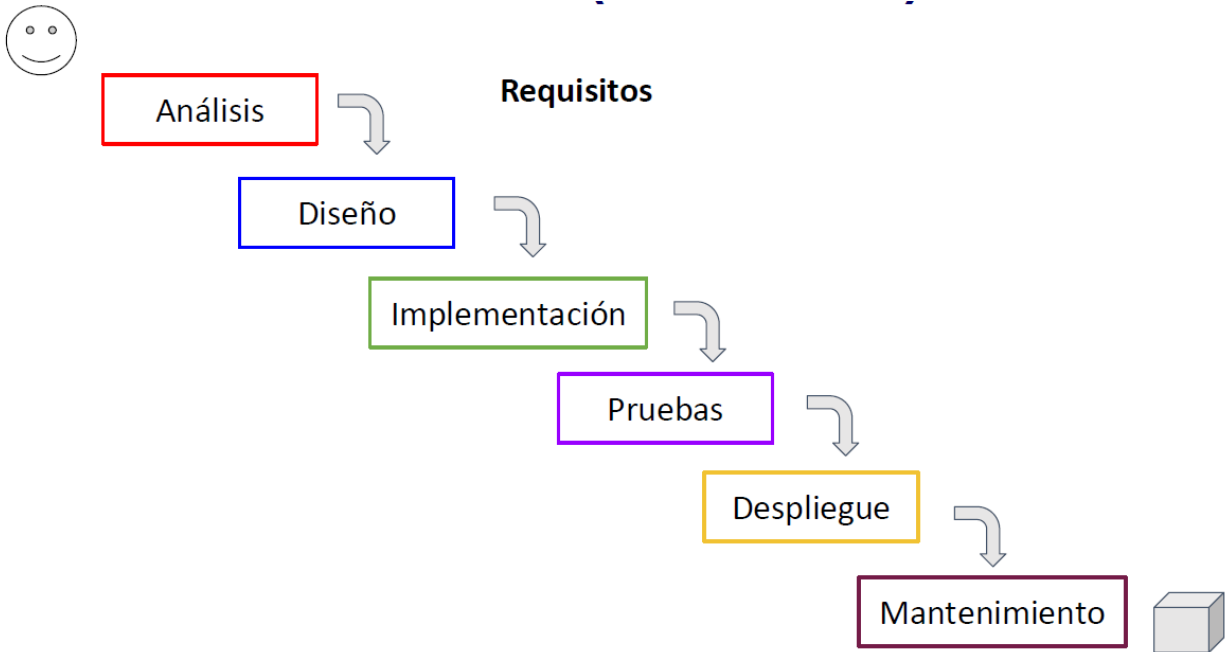
Codificar y corregir - Code and Fix

- Escribir código
- Probar el código
- Corregir problemas en el código
- Volver a probar



Lo anterior, es una mala práctica. Es mejor planear y diseñar antes de ir a codificar

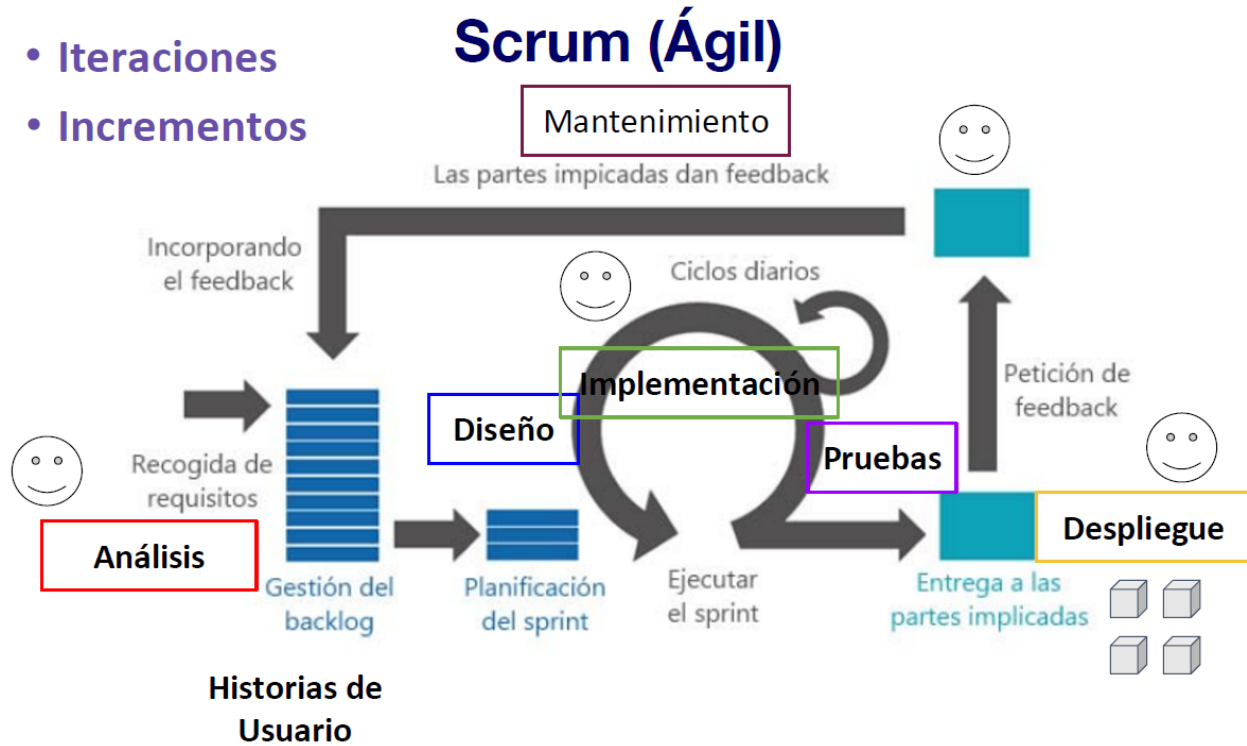
Cascada (Tradicional)



Utilizado normalmente, en Sistemas de salud en hospitales, Sistemas bancarios, Sistrmas militares y Sistemas de control

Scrum (Ágil)

- Iteraciones
- Incrementos



- Debemos de analizar la **historia de usuario**
- A partir de esta historia, analizar que **criterios de aceptación** sacamos de esto para sacar un tipo de lista de items con estos criterios

Cascada	Ágil
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Grupos grandes y posiblemente distribuidos	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
Más artefactos	Pocos artefactos
Más roles	Pocos roles
Documentación exhaustiva	Poca documentación (la necesaria)

El equipo de Scrum consiste en tres diferentes roles:

- El **Product Owner/Dueño del producto** es la “voz del cliente” y el responsable de desarrollar, mantener y priorizar las tareas en el *backlog*.
- El **Scrum Master** es responsable de asegurarse que el trabajo del equipo vaya bien siguiendo las bases de Scrum. Además, se encarga de remover cualquier obstáculo que pueda encontrar el equipo de desarrollo.
- Los **Development Team Members/Miembros del Equipo de desarrollo** son los encargados de escribir y probar el código.

Más detalles a profundidad sobre esta metodología de desarrollo en [Platzi](#)