*Introducción a la ingeniería del software y el modelado.

- Software: Conjunto de los programas de computo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.
- Características del software:
  - Se desarrolla o modifica con intelecto.
  - No se desgasta, pero sí se deteriora.
  - Se construye para un uso individualizado.
- Dominios de aplicación del software:
  - Software de sistemas: Conjunto de programas escritos para dar servicios a otros programas.
  - Software de aplicación: Programas que resuelven una necesidad específica de negocios.
  - Software de ingeniería y ciencias: Tambien llamados "devoradores de numeros", porque usa procesos de alto nivel de computación.
  - Software incrustado: Es el que reside dentro de un producto o sistema y se usa para el control de caracteristicas muy limitadas. (La placa de un microondas, los testigos del vehiculo...)
  - Software de línea de productos: Diserado para proporcionar una capacidad especifica para uso de muchos usuarios diferentes. (Hojas de calculo, procesadores de textos).
  - Aplicaciones web: Son aplicaciones que estan en la red, trabajan en concurrencia, el rendimiento y la disponibilidad son críticos, estan orientadas a datos, estan en constante evolución, el contenido y la estética son muy importantes y es fundamental agarantizar su seguridad.
  - Software de inteligencia Artificial: Hace uso de algoritmos no numericos para resolver problemas complejos.

-Software heredado o legacy. Es el sistema, tecnología o aplicación antiguo o desactualizado que sigue en uso dentro de una organización porque sigue desempeñando las funciones para las que fue diseñado.
Si un programa funciona, no lo toques.

¿Qué es la ingeniería del software?
Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.
- Se debe entender el problema antes de desarrollar una aplicación.
- El diseño es una actividad crucial de la ingeniería del software.
- Tanto la calidad como la facilidad de recibir mantenimiento son el resultado de un buen diseño.

La ingeniería del software se compone de 3 capas:
1. Proceso: Es donde se une la tecnología y permite el desarrollo inteligente del software.
Forma la base para el control de la administración de proyectos de software y establece donde se aplican los métodos técnicos.
2. Métodos: Proporciona la experiencia técnica para elaborar el software.
Las actividades técnicas fundamentales son:
   - Análisis
   - Diseño
   - Codificación
   - Pruebas
3. Herramientas: Proporciona soporte a las capas de proceso y métodos.

- Elementos de un proceso de desarrollo de software:
  - Comunicación: Entender los objetivos de cada uno de los participantes respecto de lo que se requiere.
  - Planificación: Crear el plan del proyecto y la planificación de las actividades.
  - Modelado: Crear un modelo para entender los requerimientos del software y el diseño que los cumplirá.
  - Construcción: Generar el código y las pruebas necesarias.
  - Despliegue: Entregar el software al usuario para su evaluación y nos de retroalimentación.

*Introducción al Lenguaje de Modelado Unificado (UML):

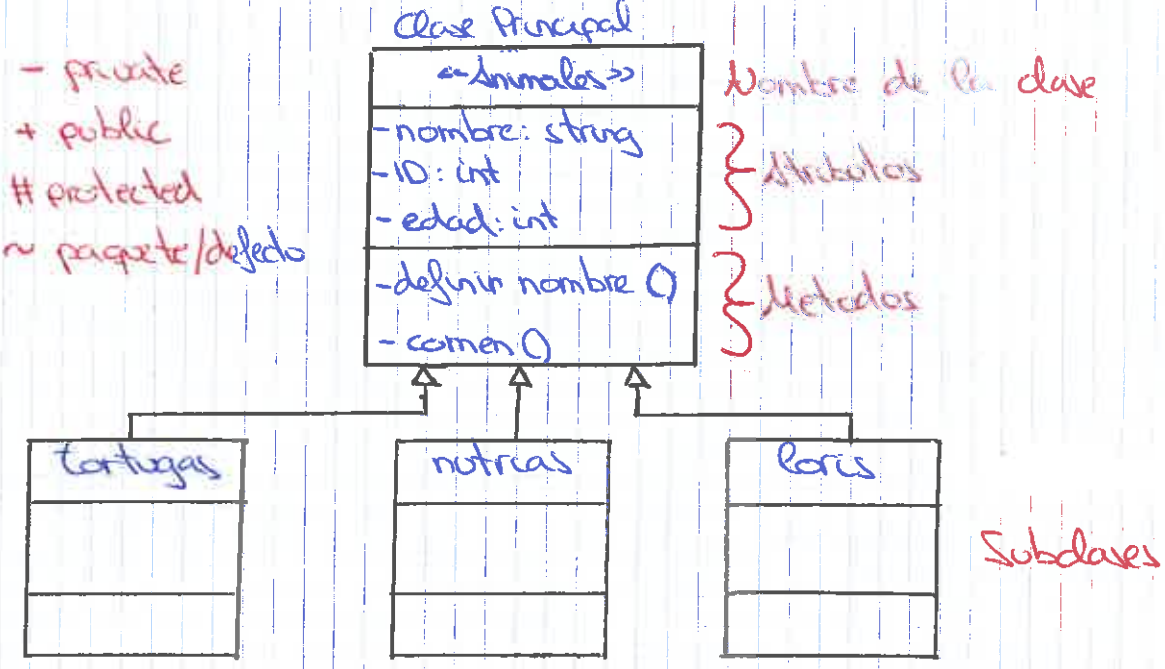
El UML es un lenguaje estandar para escribir diseños de software.

El UML puede usarse para visualizar, especificar, construir y documentar los artefactos de un sistema de software.

UML 2.0 proporciona 13 diagramas diferentes, aunque solo veremos 7.

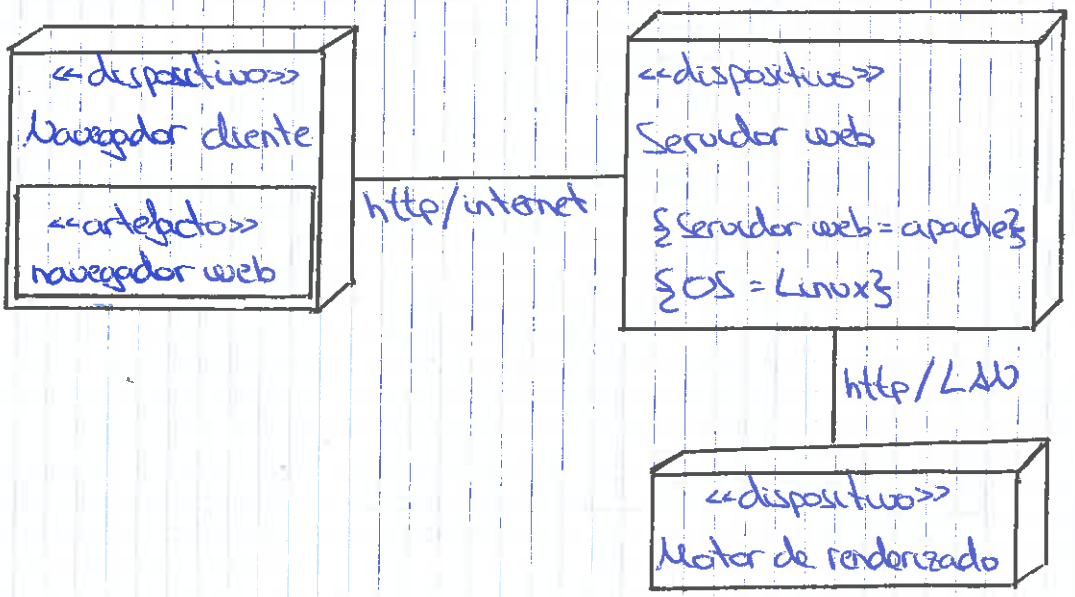
-Diagrama de clases:

- Permite modelar las clases (atributos y metodos).
- Modela las relaciones entre las clases.



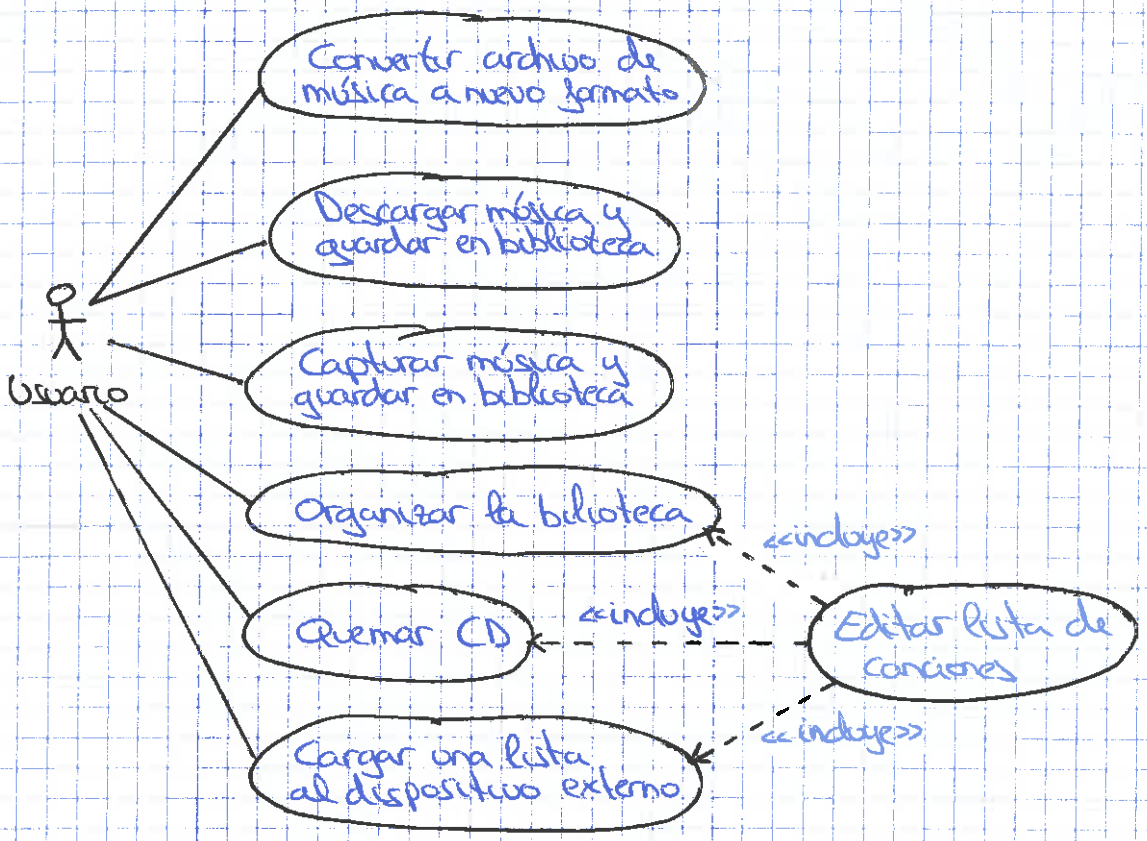
-Diagrama de implementación:

- Muestra la distribución física de un sistema de software entre plataformas de hardware y entornos de ejecución.



-Diagramas de casos de uso:

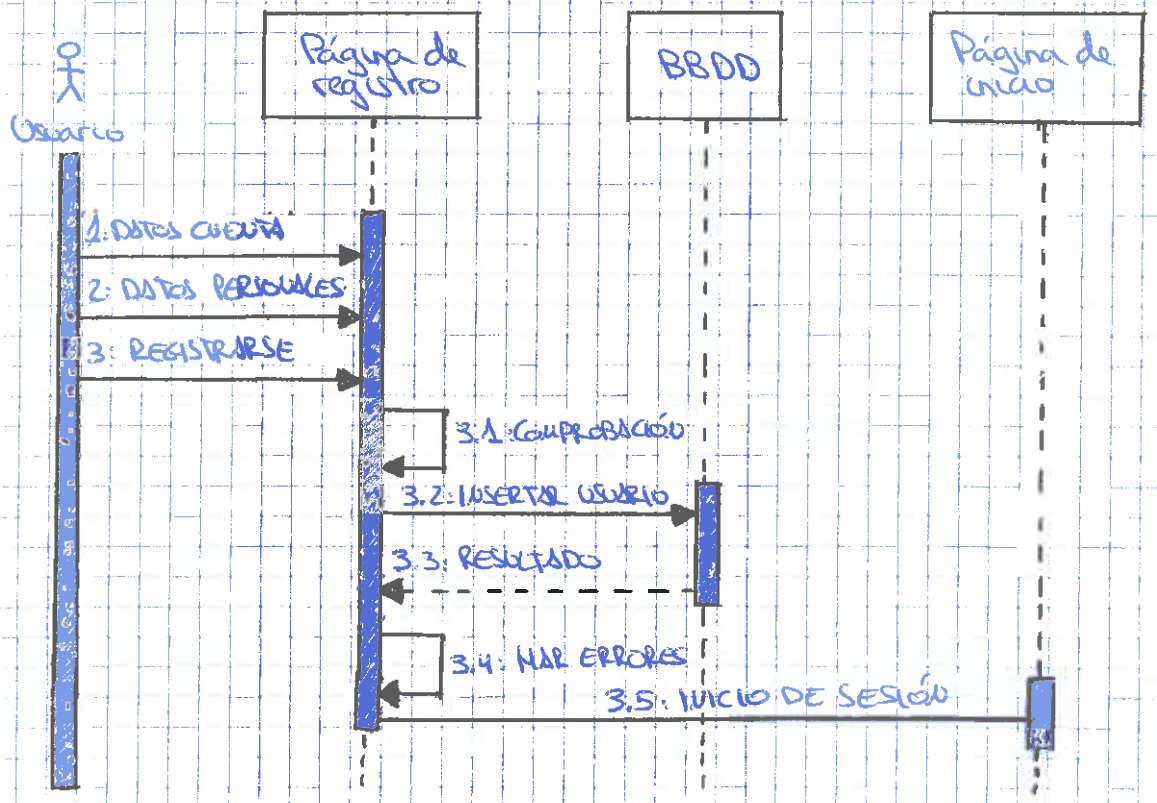
·Muestra la interacción de los usuarios con las diferentes funciones del sistema.



-Diagrama de secuencia:

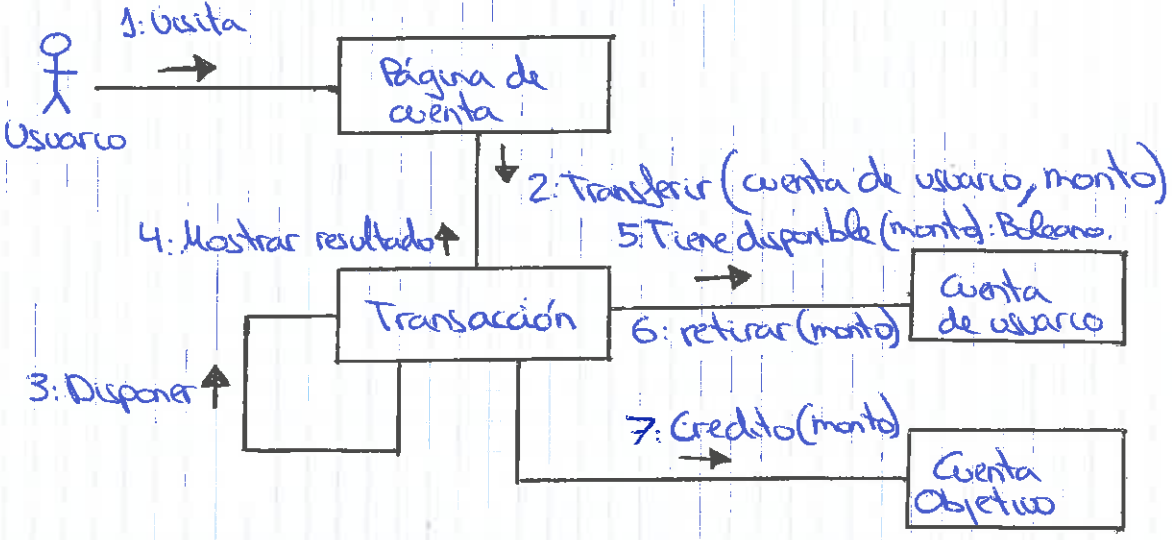
·Muestra la comunicación dinámica entre los diferentes objetos para llevar a cabo una tarea.

·Describen el orden de ejecución.



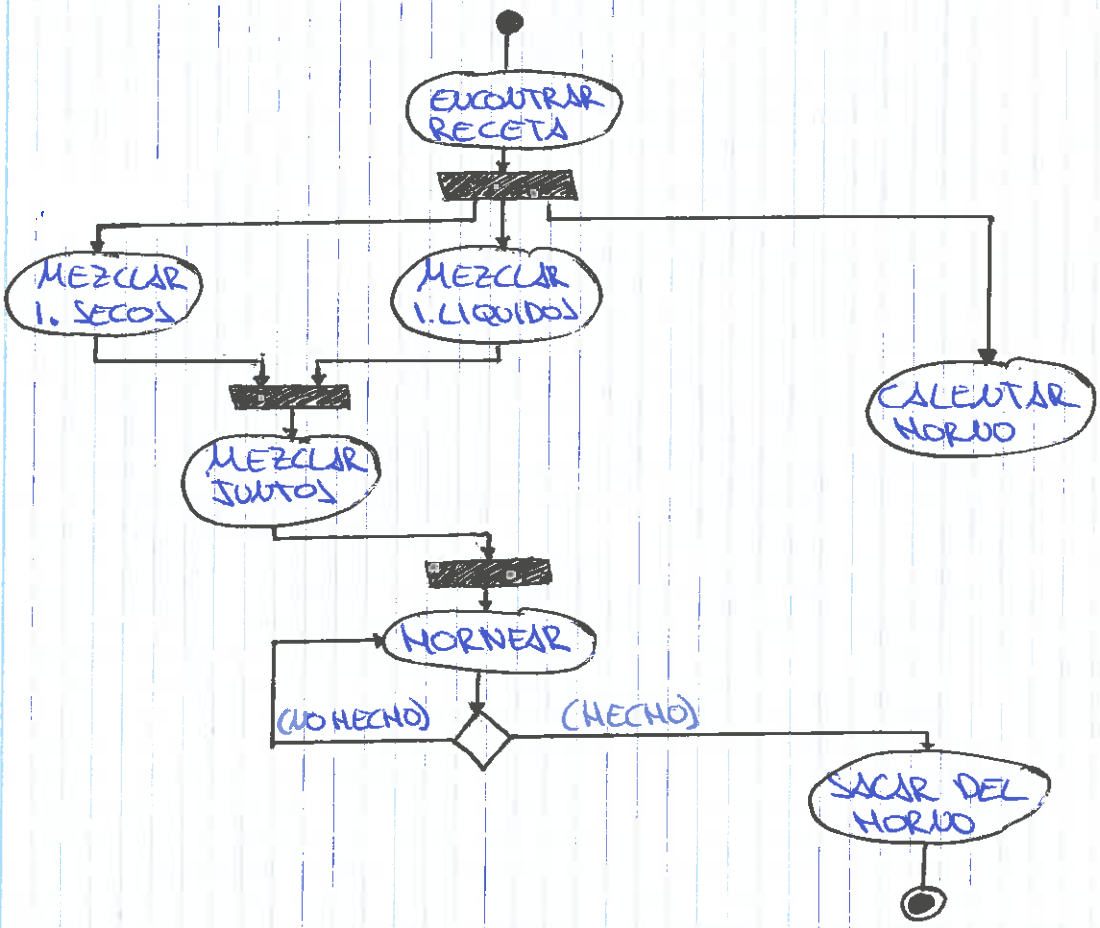
- Diagrama de comunicación:

.Muestra la comunicación dinámica entre los objetos para llevar a cabo una tarea, pero enfatiza las relaciones entre los objetos y clases en lugar del orden temporal.



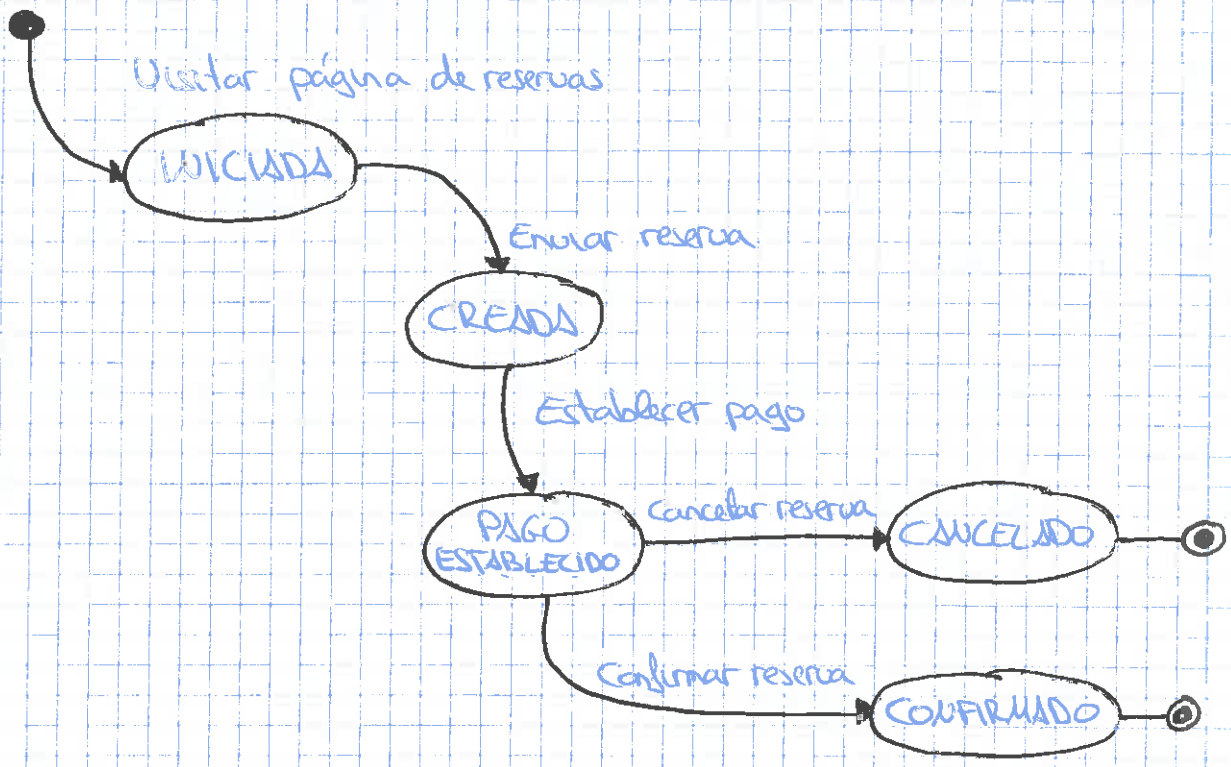
- Diagrama de actividad:

.Muestra el comportamiento dinámico del sistema mediante el flujo de control de las acciones que realiza el sistema



-Diagrama de estado.

· Se modelan los estados por los que pasa un objeto y las acciones que desencadena la transición entre estados.



*El proceso del software:

El proceso del software se compone de cinco actividades:

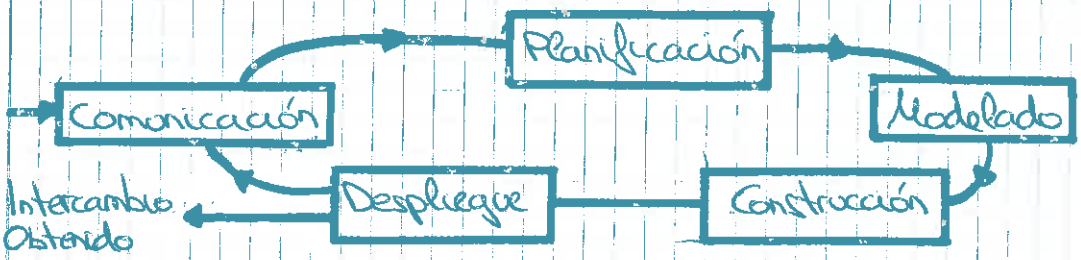
- Comunicación: Entender los objetivos de los participantes respecto de lo que se requiere.
- Planificación: Crear un plan de proyecto y la planificación de las actividades.
- Modelado: Crear un modelo para entender los requerimientos del software y el diseño que los cumplirá.
- Construcción: Generar el código y realizar las pruebas necesarias.
- Despliegue: Entregar el software al usuario para su evaluación y que nos de retroalimentación.

Dependiendo de la secuencia en que se realicen las actividades anteriores, se le da nombre al flujo del proceso:

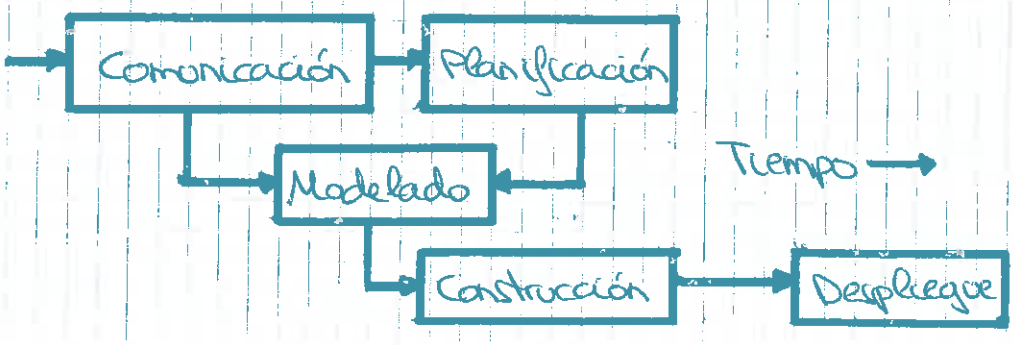
- Lineal: Ejecuta las actividades de forma secuencial.

- Iterativo: Repite una o más actividades antes de pasar a la siguiente.

- Evolutivo: Se realiza el proceso de forma circular. Cada circuito lleva a una versión más completa del software.



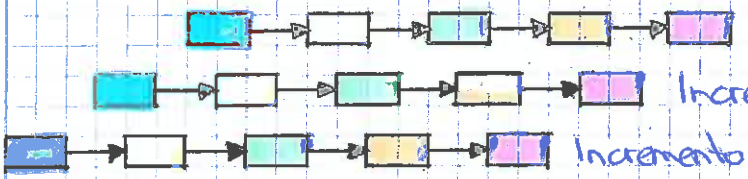
- Paralelo: Ejecuta una o más actividades en paralelo con otras.



Modelos de proceso prescriptivos?

-Modelo en cascada: Se inicia la primera actividad del proceso de software y cuando se finaliza comienza la siguiente actividad. Como el flujo lineal.

- Modelo incremental: Se realizan repetidas secuencias lineales de modo escalonado. Al final de cada secuencia se obtiene un incremento parcial del producto a entregar. En la primera secuencia ya podemos usar nuestro software.



-Modelo evolutivo: Sigue un flujo iterativo, en el que, en cada ciclo, se desarrollan versiones del software mas completas. El software solo vera la luz cuando este completamente acabado.

-Modelo en espiral: Podriamos considerarlo como una mezcla entre el modelo incremental y el modelo evolutivo, en el que la primera revolución acomete a una función de la aplicación y a medida que vamos haciendo revoluciones, el software va creciendo y evolucionando.
Este desarrollo sigue vivo hasta que el software desaparece.

-Modelo concurrente: Permite representar actividades concurrentes e iterativas, pudiendo estar cada una de ellas en un estado diferente, estableciendose una serie de eventos que originan los cambios de estado.

Modelos de proceso especializado:

- Modelo basado en componentes: Se estructura el software en componentes (fragmentos de software prefabricados) y, posteriormente, se diseñan, integran y prueban.

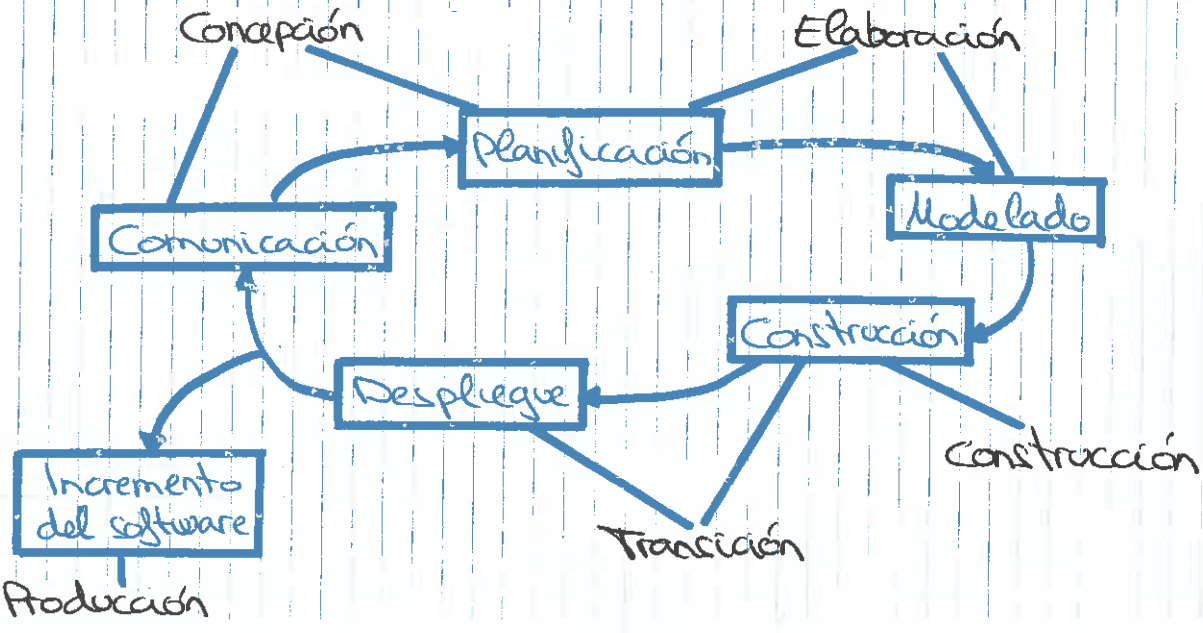
- Modelo de métodos formales: Hacen uso de la especificación matemática formal del software para especificar, desarrollar y verificar el sistema.

- Modelo orientado a aspectos: Se definen, especifican, diseñan y construyen aspectos.
  - Aspecto: Propiedades globales funcionales y no funcionales de componentes.
    - Ejemplos:
      - Propiedades de alto nivel (seguridad y tolerancia a fallos).
      - Funciones (aplicaciones de las reglas de negocio).
      - Sistémicas (sincronización de tareas paralelas o administración de recursos).

Modelos de proceso unificado:

- UML (Lenguaje de modelado unificado): Intenta mezclar las mejores características de los modelos tradicionales implementando características del desarrollo ágil.

El proceso unificado consta de cinco fases:



Modelos de proceso personal y del equipo:

- Proceso personal del software (PPS): Se basa en la medición personal del producto y su calidad.

Se organiza en cinco actividades:

- Planeación: Plan de trabajo.
- Diseño de alto nivel: Diseño de componentes.
- Revisión del diseño de alto nivel.
- Desarrollo: Se implementa, prueba y mejora el componente.
- Mediciones para la mejora del proceso.

- Proceso del equipo del software (PES): Tiene como objetivo construir un equipo capaz de autodirigirse para el desarrollo de un proyecto y que sean dueños de sus procesos y sus planes.

¿Que es la agilidad?

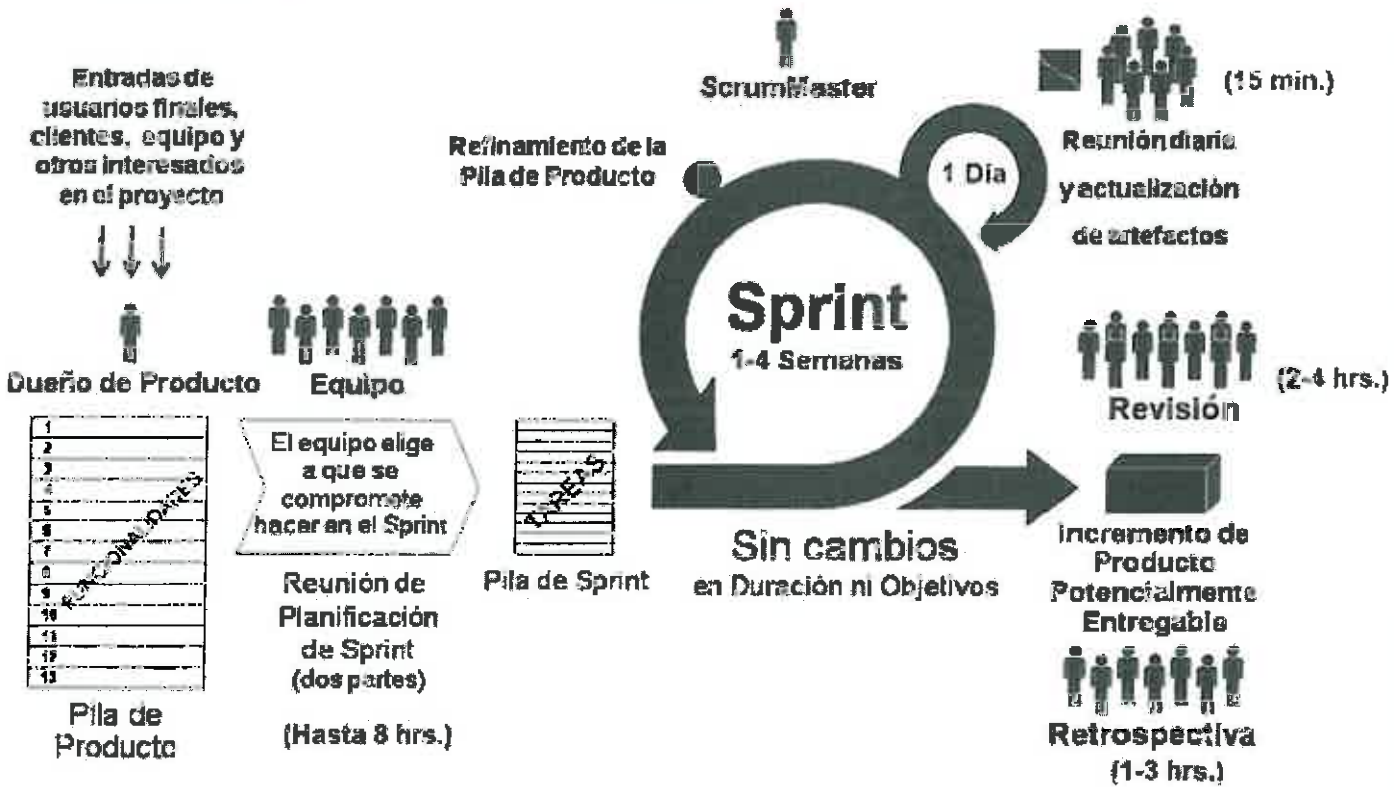
Un proceso de software agil es un proceso de adaptación incremental rápida a los cambios en las necesidades del proyecto, apoyandose en la retroalimentación constante del cliente. Cuanto mas avanzado esta un proyecto, más costoso es realizar un cambio. Los procesos ágiles tratan de reducir el coste de estos cambios.

Un proceso agil debe de adaptarse a las necesidades de las personas y el equipo.

- Deben de ser competentes.
- Tener un objetivo común.
- Colaborar entre si y con otros participantes.
- Organizarse ellos mismos.
- Tener habilidad para tomar decisiones y resolver problemas.
- Tener confianza y respeto.

Scrum: Es un método de desarrollo agil de software

ESTADOS DEL MODELO SCRUM:



*Principios que guían la práctica de la ingeniería de software.

La practica de la ingeniería del software incluye principios, conceptos, métodos y herramientas que los ingenieros de software aplican en todo el proceso de desarrollo.

-Principios fundamentales: son una serie de principios que se aplican al proceso como un todo.

Principios que guían el proceso:

1. Ser agil.
2. Centrarse en la calidad
3. Estar listo para adaptarse.
4. Formar un equipo eficaz.
5. Establecer mecanismos para la comunicación y la coordinación.
6. Administrar los cambios.
7. Evaluar los riesgos.
8. Crear productos de trabajo que agreguen valor para los demas.

Principios que guían la práctica:

1. Divide y venceras
2. Entender el uso de la abstracción.
3. Buscar la coherencia.
4. Centrarse en la transferencia de información.
5. Construir software que tenga modularidad eficaz.
6. Buscar patrones.
7. Representar el problema y la solución desde varias perspectivas.
8. Tener en mente que alguien dara mantenimiento al software.

- Principios estructurales. Se aplican a cada una de las actividades.

Principios de comunicación.

1. Escuchar.
2. Prepararse antes de comunicarse.
3. Toda reunión de comunicación ha de tener un líder.
4. La comunicación mejor cara a cara.
5. Tomar notas y documentar las decisiones.
6. Colaborar.
7. Permanecer centrado en el tema principal.
8. Si algo no está claro, hacer un dibujo.
9. Avanzar
10. La negociación no es un concurso y es mejor cuando todos ganan.

Principios de planificación:

1. Entender el alcance del proyecto.
2. Involucrar a todos los participantes.
3. Reconocer que la planificación es iterativa.
4. Estimar con base a lo que se sabe.
5. Tener en cuenta los riesgos.
6. Ser realista.
7. Ajustar el nivel de detalle cuando se defina el plan.
8. Definir como asegurar la calidad.
9. Describir como se trataran los cambios
10. Dar seguimiento al plan y hacer los ajustes que se requieran.

- Principios de modelado:

Los modelos deben representar la arquitectura, las funcionalidades y la información.

Solo se crearan los modelos necesarios lo mas sencillos posible.

1. El objetivo principal es crear software.
2. Crear solo los modelos necesarios.
3. Crear el modelo mas sencillo posible.
4. Construir modelos susceptibles al cambio.
5. Enunciar un proposito explicito a cada modelo.
6. Adaptar los modelos al sistema.
7. Construir modelos útiles, no perfectos.
8. No ser dogmático con la sintaxis del modelo.
9. Confiar en nuestro instinto.
10. Obtener retroalimentación.

- Modelos de requisitos: representan los requisitos del cliente a nivel de información, funcionalidad y comportamiento.

1. Entender y representar el dominio de información del problema.
2. Definir la funciones que realizara el software.
3. Representar el comportamiento del software.
4. Los modelos deben dividirse para que revelen los detalles de forma jerarquica.
5. Avanzar desde la información esencial hacia la implementación final.

- Modelos de diseño: representan las características que tendrá el software para facilitar el desarrollo a nivel de arquitectura, interfaz de usuario y componente.
  1. El diseño debe poderse rastrear hasta el modelo de requerimientos.
  2. Tener en cuenta la arquitectura del sistema.
  3. El diseño de los datos es tan importante como las funciones de procesamiento.
  4. Las interfaz deben diseñarse con cuidado.
  5. La interfaz de usuario debe ajustarse al usuario final.
  6. El diseño de componentes debe tener independencia funcional.
  7. Los componentes deben estar acoplados con holgura.
  8. Los modelos deben entenderse con facilidad.
  9. El diseño debe desarrollarse de forma iterativa.

*Comprensión de los requisitos:
Consiste en entender los requerimientos de un problema.

-Ingeniería de requisitos: Es el conjunto de tareas y técnicas que nos facilitan la comprensión de los requerimientos que debe cumplir el software.
Esta compuesta por 7 tareas:

- Concepción: Comienza cuando se identifica una necesidad de negocio o se descubre un nuevo mercado o servicio potencial. Establece el entendimiento básico del problema, las personas que requieren una solución, la naturaleza de la solución y la eficacia de la comunicación y colaboración.

- Indagación: Consiste en preguntar al cliente:
  Cuales son los objetivos del sistema.
  Como va a usarse el sistema.
  Como se ajusta el sistema a las necesidades del negocio.
  Que pretende lograrse con este sistema.
  En la tarea de indagación pueden presentarse algunos problemas:
  Problemas de alcance: La frontera entre los sistemas esta mal definida o los clientes especifican detalles técnicos innecesarios.
  Problemas de entendimiento: Los clientes no se expresan adecuadamente o no tienen claro lo que quieren.
  Problemas de volatilidad: Los requisitos cambian con el tiempo.

- Elaboración: Consiste en el desarrollo detallado de los requisitos.
  Creación y mejora de escenarios de uso que describan como interactua el usuario final con el sistema.

- Negociación: Muchas veces los clientes piden más de lo pueden o proponen requerimientos conflictuos. Para solventar esto existe la tarea de negociación.
  Se le pedira al cliente que ordenen sus requerimientos por prioridades y luego busquen los conflictos.
  Los conflictos se resolveran agradando a ambas partes.

- Especificación: Es muy ambiguo, puede ser un documento escrito, un modelo gráfico, un prototipo ...
Para proyectos grandes suele usarse una plantilla estándar:
Especificación de requerimientos del software (ERS):

INFORMACIÓN

Formato de especificación de requerimientos de software

Una *especificación de requerimientos de software* (ERS) es un documento que se crea cuando debe especificarse una descripción detallada de todos los aspectos del software que se va a elaborar, antes de que el proyecto comience. Es importante notar que una ERS formal no siempre está en forma escrita. En realidad, hay muchas circunstancias en las que el esfuerzo dedicado a la ERS estaría mejor aprovechado en otras actividades de la ingeniería de software. Sin embargo, se justifica la ERS cuando el software va a ser desarrollado por una tercera parte, cuando la falta de una especificación crearía problemas severos al negocio, si un sistema es complejo en extremo o si se trata de un negocio de importancia crítica.

Karl Wiegers [Wie03], de la empresa Process Impact Inc., desarrolló un formato útil (disponible en **www.processimpact.com/process_assets/srs_template.doc**) que sirve como guía para aquellos que deben crear una ERS completa. Su contenido normal es el siguiente:

Tabla de contenido
Revisión de la historia

1. Introducción
1.1 Propósito
1.2 Convenciones del documento
1.3 Audiencia objetivo y sugerencias de lectura
1.4 Alcance del proyecto
1.5 Referencias

2. Descripción general
2.1 Perspectiva del producto
2.2 Características del producto
2.3 Clases y características del usuario
2.4 Ambiente de operación
2.5 Restricciones de diseño e implementación
2.6 Documentación para el usuario
2.7 Suposiciones y dependencias

3. Características del sistema
3.1 Característica 1 del sistema
3.2 Característica 2 del sistema (y así sucesivamente)

4. Requerimientos de la interfaz externa
4.1 Interfaces de usuario
4.2 Interfaces del hardware
4.3 Interfaces del software
4.4 Interfaces de las comunicaciones

5. Otros requerimientos no funcionales
5.1 Requerimientos de desempeño
5.2 Requerimientos de seguridad
5.3 Requerimientos de estabilidad
5.4 Atributos de calidad del software

6. Otros requerimientos

Apéndice A: Glosario
Apéndice B: Modelos de análisis
Apéndice C: Lista de conceptos

Puede obtenerse una descripción detallada de cada ERS si se descarga el formato desde la URL mencionada antes.

- Validación: es la tarea donde se analiza la especificación a fin de garantizar que no hay ambigüedades, que se detectaron y corrigieron las inconsistencias, las omisiones y los errores.

- Administración: son el conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y dar seguimiento a los requisitos.

- Establecer las bases:
  - Identificar a los participantes que se beneficiaran del sistema a desarrollar.
  - Identificar las areas de interes común y de conflicto para elegir las más adecuadas.
  - Hacer las primeras preguntas: estas estaran libres de contexto.
    Las primeras se centran en el cliente y en las metas.
    Las siguientes nos ayudan a conocer mejor el problema.
    Las ultimas se centran en la eficacia de la relación establecida.

- Indagación de los requisitos:
  Los participantes trabajan juntos para identificar el problema, proponer soluciones, valorar los distintos puntos de vista y generar un conjunto inicial de requisitos.
  Hay tres tipos de requisitos:
  - Normales: Los que los objetivos y metas se establecen durante la reunión con el cliente.
  - Esperados: Son tan obvios que el cliente no los mencionara, pero si no estan presentes causara insatisfacción
  - Emocionantes: Son los que van mas allá de lo que espera el usuario.

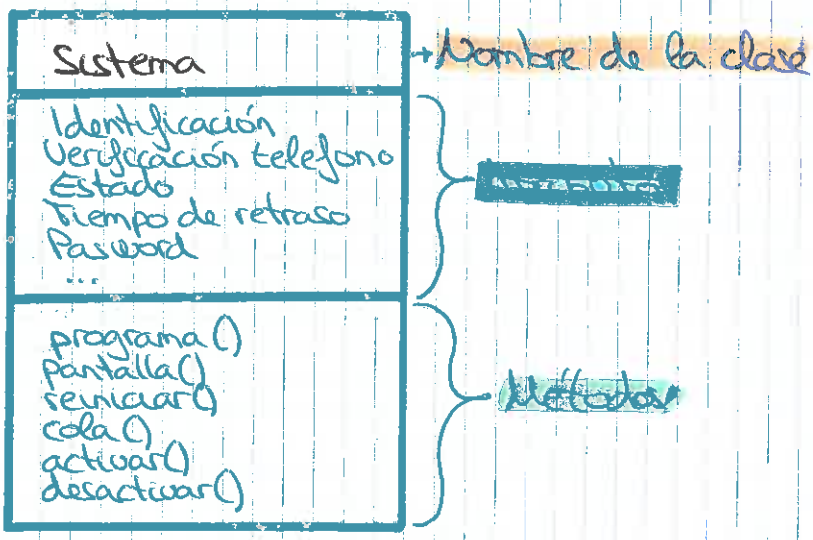
*Modelado de clases:

Representa las clases del sistema, identificando sus atributos y métodos, así como las interrelaciones entre ellas.

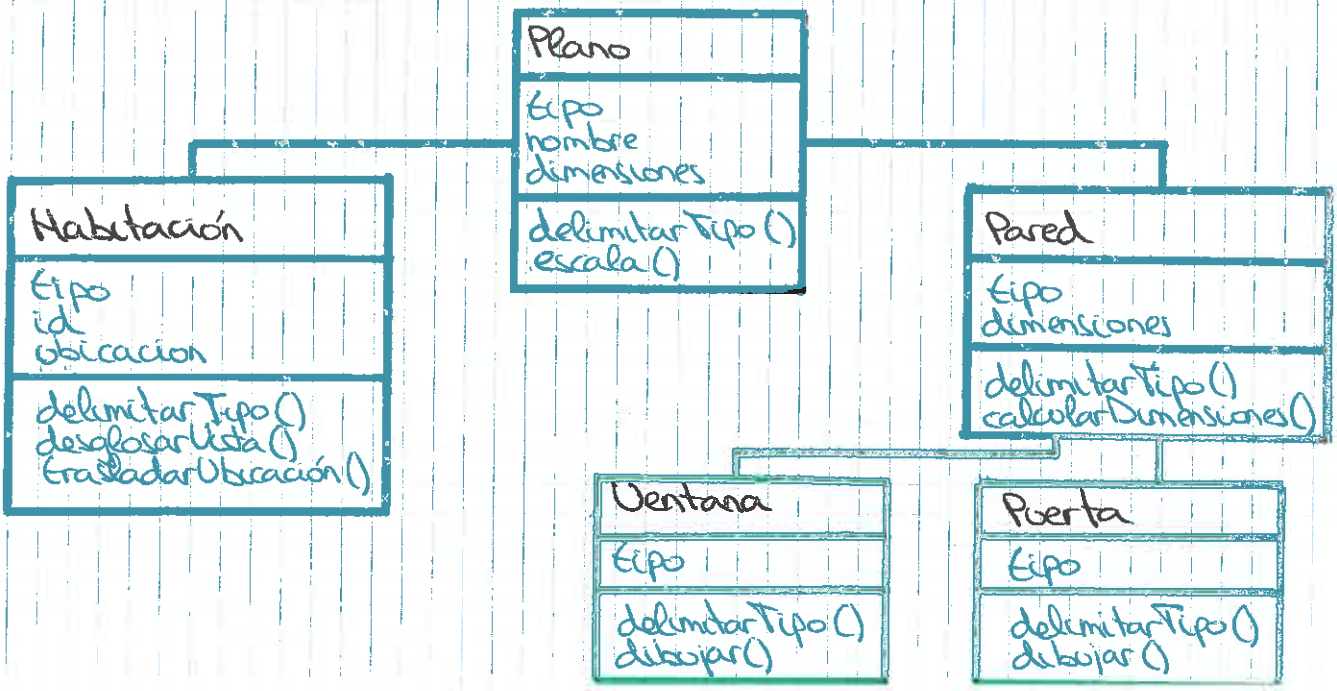
Sigue los siguientes pasos:

- Identificación de las clases: se analizan los casos de uso y se identifican los sustantivos, y de estos se identifican los que representan a clases del sistema.
- Identificación de los atributos.
- Identificación de los metodos.
- Identificación de las relaciones entre las diferentes clases para representar las jerarquías, relaciones, asociaciones, agregaciones y dependencias.

Ejemplo de clase:



Ejemplo de diagrama de clases:



Tipos de relaciones:

-Generalización o herencia: ——▷

Es una relación entre clases que comparte su estructura y comportamiento.
La clase hija hereda de la clase padre
Se distingue con "..es un...", por ejemplo:
Estudiante "es una" persona.
Clase hija — Clase padre.

-Asociación: ——

Es una relación simple entre clases que especifica que los objetos de un elemento estan unidos con los objetos de otro.
Se puede navegar de uno a otro y viceversa
Cliente —— Dirección
Puede ponerse una descripción o una flecha para indicar la dirección.
Persona —Tiene una→ mascota

-Dependencia: - - - - - →

Es una relación de uso, es decir, una clase utiliza a otra.
La clase A utiliza a la clase B.
Impresora - - - - - → Papel.

-Agregación: ——◇

Es una variante de la relación de asociación que especifica un todo y sus partes.
Tortugero ◇—— Tortugas
Una tortuga podra estar en el tortuguero, pero si el tortuguero desaparece, la tortuga seguira existiendo.

-Composición: ——◆

Es una relacion de asociación donde la parte no puede existir fuera del todo.
Oficina ◆—— Baños Sin oficina no hay baños.

-Realización: - - - - ▷

Es una relación entre dos clases que se utiliza para implementar una interface.
Carro - - - - - - - ▷ «interface» carro

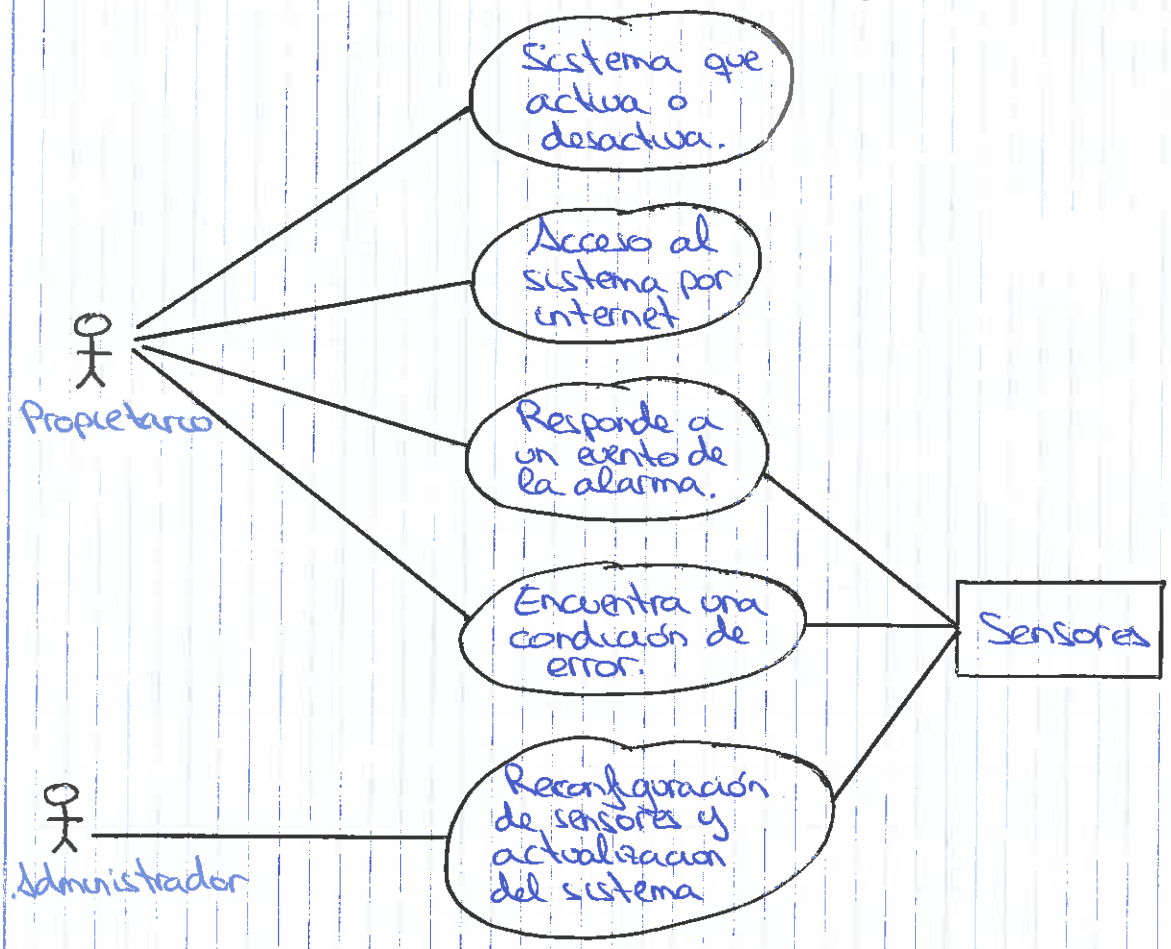
- Desarrollo de casos de uso:

Un caso de uso describe como interactua un usuario final con el sistema, definiendo los pasos requeridos para lograr un determinado objetivo.

La primera tarea a realizar será identificar a los actores; son las entidades que desempeñan un rol en el caso de uso (casi siempre personas).

En un diagrama UML de caso de uso, los actores se conectan mediante lineas a los casos de uso, que se representan como óvalos.

Caso de uso de una alarma (Diagrama UML):



Los casos de uso básico representan una historia de alto nivel que describe la interacción entre el actor y el sistema. A veces los casos de uso son mas elaborados y brindan mas detalles.

- Elaboración del modelo de requisitos:
  El modelo de requisitos describe la información, funcionalidad y comportamiento necesario para nuestro nuevo sistema.
  Los elementos del modelo de requisitos pueden ser:
  - Elementos basados en el escenario: se describe desde el punto de vista del escenario (casos de uso).
  - Elementos basados en clases: Los objetos que manipula un actor cuando interactúa con el sistema. (diagrama de clases).
  - Elementos de comportamiento: Modelados que ilustran el comportamiento (diagramas de estado).
  - Elementos orientados al flujo: Diagramas de flujo que muestra como se transforma la información cuando fluye a través del sistema.

- Negociación de los requisitos: Es el proceso de resolución de conflictos en los requisitos. Hay que valorar la funcionalidad del sistema en contraste con el tiempo y el coste del desarrollo.
  Estas negociaciones deben estar orientadas a que todos salgan ganando.

- Validación de los requisitos:
  Los distintos elementos que se elaboran en el modelo de requisitos deben revisarse para detectar inconsistencias, ambigüedades u omisiones.
  Los participantes asignan prioridades a los requerimientos y se agrupan en paquetes de requerimientos que se implementaran como incrementos del software.

*Modelado de los requisitos:

El analisis de los requisitos da como resultado la especificación de las caracteristicas operativas del software, indica su interface y establece las restricciones que lo limitan.

El modelado de requisitos puede ser:

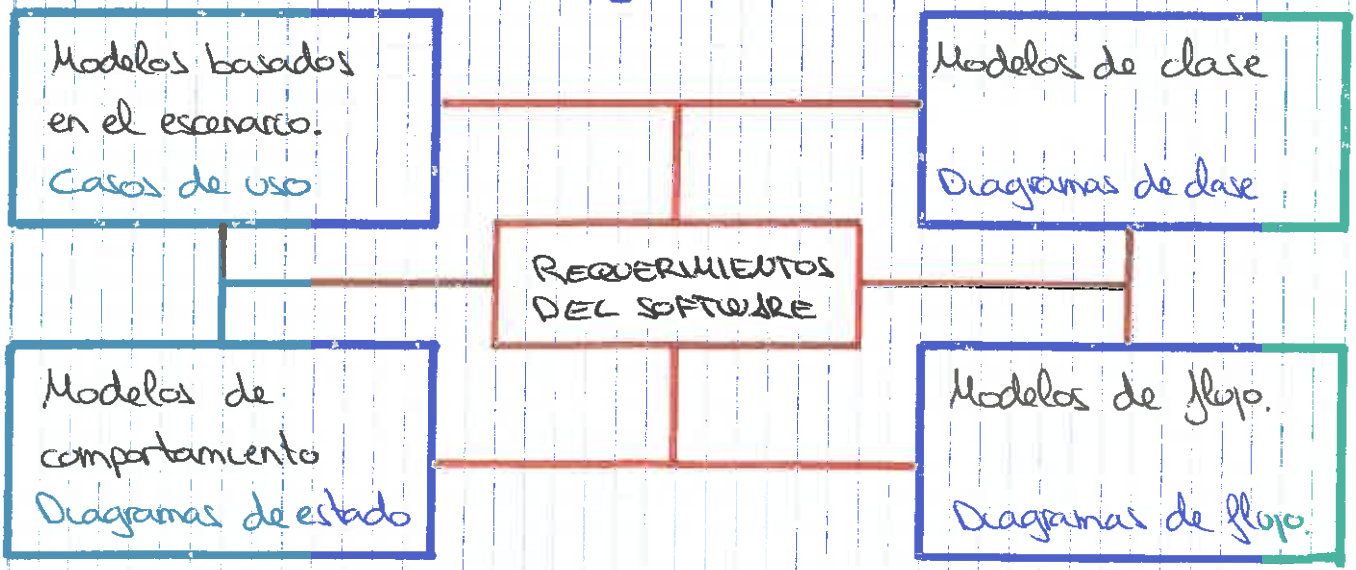
- Basado en el escenario de los requisitos: se describe desde el punto de vista del usuario (Casos de uso)
- Modelos de datos: nos centramos en la información del sistema.
- Modelos orientados a clases: representan las clases orientadas a objetos y las relaciones entre las clases (Diagrama de clase).
- Modelos orientados al flujo: representan funcionalidades del sistema que muestran como se transforman los datos a medida que fluyen a traves del sistema. (Diagrama de flujo).
- Modelos de comportamiento: ilustra el comportamiento del sistema como consecuencia de eventos externos.

Los objetivos del modelado de requisitos son:

- Describir lo que quiere el cliente.
- Establecer la base para el posterior diseño del software.
- Definir los requisitos que habrá que validar una vez construido el software.

Los analisis de requisitos pueden ser:

- Analisis estructurado: Analisis independiente de los datos y de los procesos que los transforman.
- Analisis orientado a objetos: Analisis de las clases del sistema (Atributos y metodos).



-Modelado basado en escenarios: se basa en analizar como desea actuar el usuario final con el sistema.

Para llevarlo a cabo:

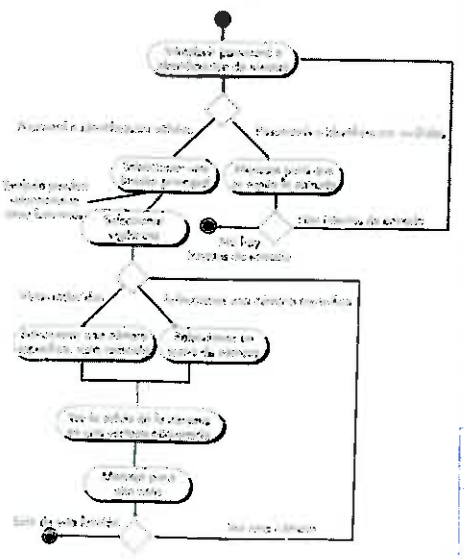
1. Se crean los casos de uso preliminares:
   - Definiendo la funcionalidad del sistema
   - Identificando a los actores.
   - Enumerando las características que debe cumplir la funcionalidad
2. Se refina cada caso de uso preliminar:
   - Se analiza en profundidad cada caso de uso para identificar escenarios adicionales, condiciones de error y comportamientos alternativos.
3. Se describen formalmente los casos de uso:
   - Actor principal.
   - Actores secundarios.
   - Objeto.
   - Precondiciones.
   - Lista de características del escenario.
   - Excepciones.
   - Prioridad.
4. Se representan graficamente con un diagrama UML.

-Modelos UML que proporcionan el caso de uso:

Los diagramas de actividades enriquecen la información.

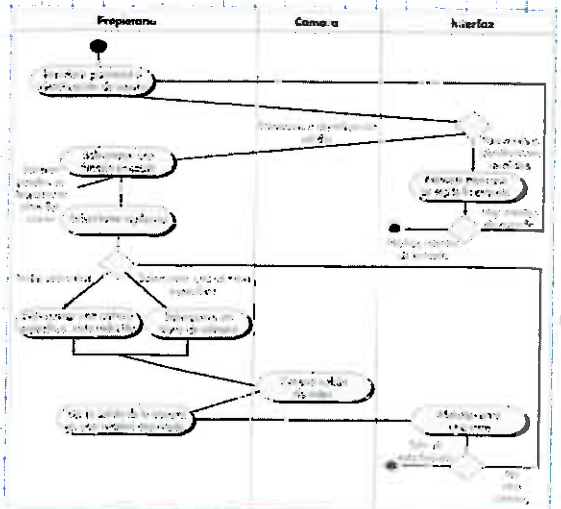
Diagramas de actividades

Representan el flujo de interacción dentro del escenario



Diagramas de canal

Es un diagrama de actividad que añade información sobre el actor responsable de cada acción.



-Concepto de modelado de datos:

- Modelado de datos: define los objetos de datos que se procesan en el sistema, los atributos que tiene y las relaciones entre ellos.
- Objeto de datos: Es una representación de información compuesta por atributos que permiten identificar la instancia, describir la instancia o hacer referencia a otra instancia de datos.

Ejemplo:

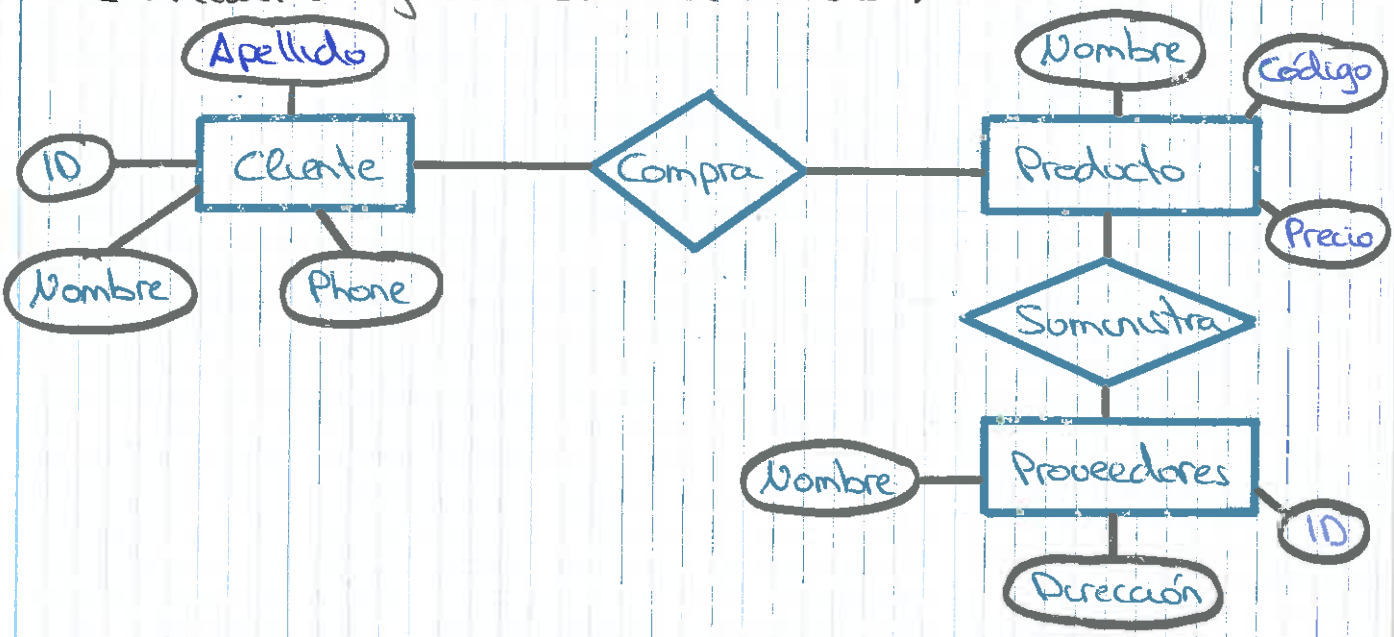
Nombrar los atributos

Fabricante	Modelo	Nº Serie	Carrocería	Color	Propietario
Lexus	LS400	AB123..	Sedan	Blanco	SGR
Ford	Mondeo	X467..	Sedan	Azul	JMP
BMW	M3	C1234..	Deportivo	Negro	JMO

Instancia

Identificador — Atributos descriptivos — Atributos referenciales

Se utilizan diagramas entidad-relación:



* Modelado de los requisitos:

El modelado de requisitos estudia como se transforman los objetos de datos cuando se mueven a traves del sistema.

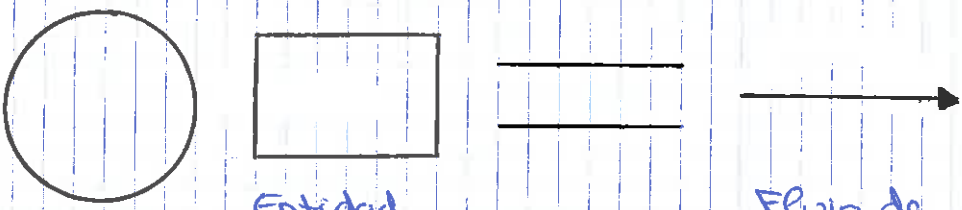
Ha de saber como se comporta una aplicación a consecuencia de eventos externos.

Comprobaremos si el conocimiento del dominio existente puede adaptarse al problema en cuestión.

-Modelado orientado al flujo (Diagrama de flujo de datos):

Se realiza con un diagrama de flujo de datos (DFD), es el que traza el flujo de la información para cualquier proceso E/S.

Elementos:



Procesos — Verbos

Entidad Externa — Sustantivos

Almacen

Flujo de datos

Composición: Se compone de 3 niveles:

- Nivel 0: Representa el sistema como un todo, representando las entradas y salidas que tendrá.
- Nivel 1: Aumenta el nivel de detalle del sistema.
- Nivel 2: Es un diagrama de detalle o expansión.

Se mantendra la continuidad del flujo de la información.

Proceso: Tendra una entrada y una salida.

Almacen: Tendra una entrada y una salida de flujo de datos.

Datos almacenados: Deben pasar por un proceso.

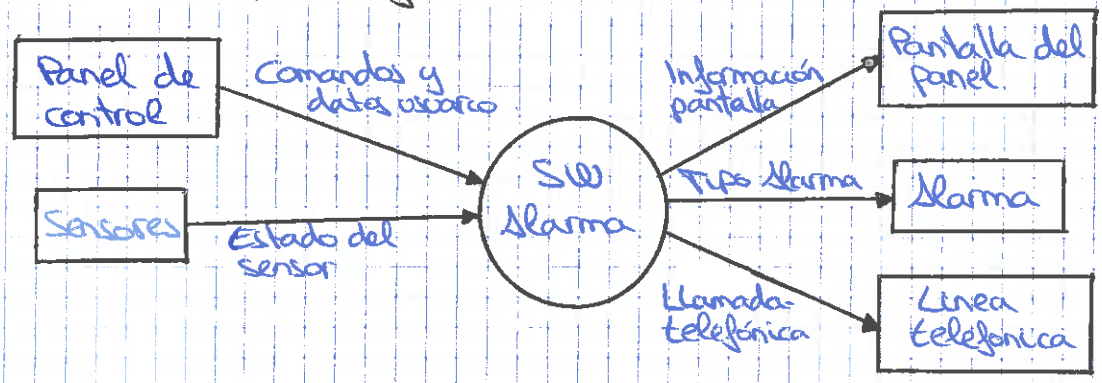
Conexiones permitidas:

Entidad ↔ Proceso

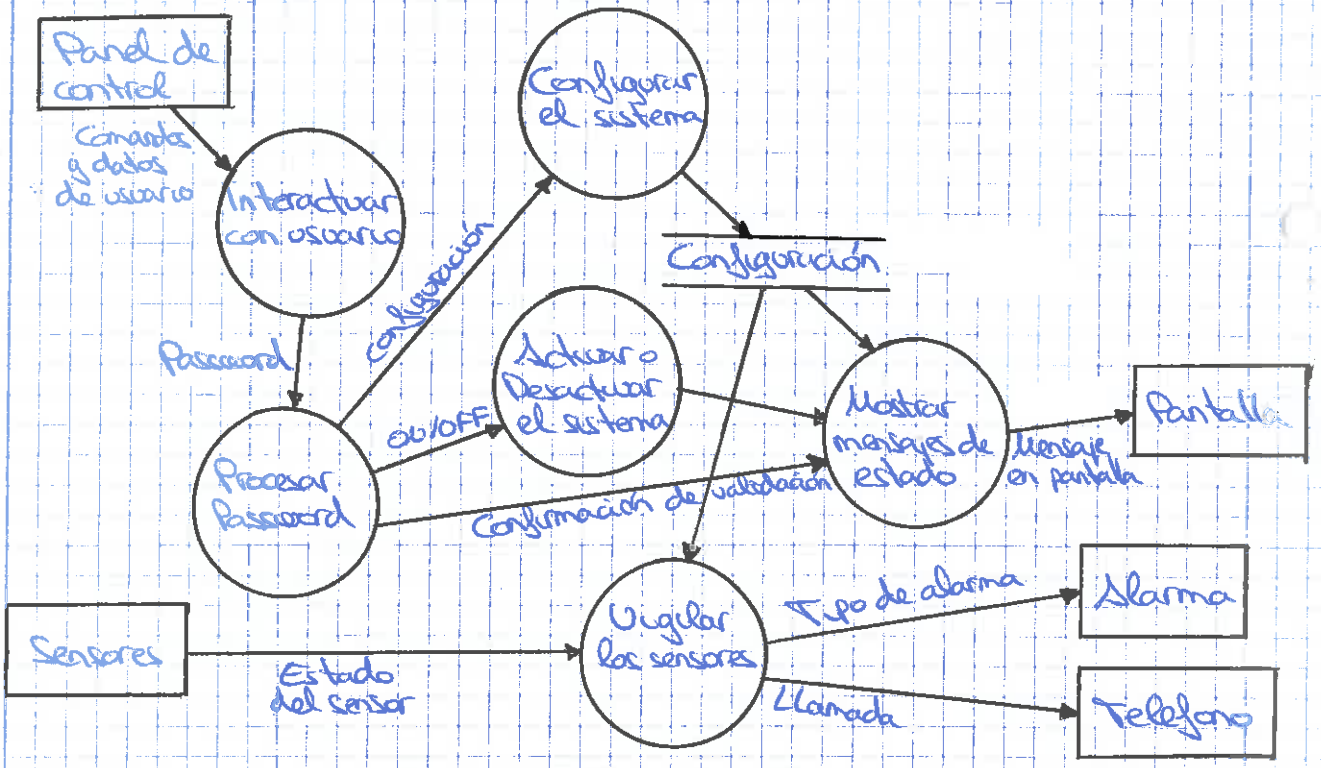
Almacen ↔ Proceso

Proceso → Proceso

DFD Nivel 0: Diagrama de contexto



DFD Nivel 1: Diagrama de Nivel superior



DFD Nivel 2: Diagrama de detalle: (Sensores)

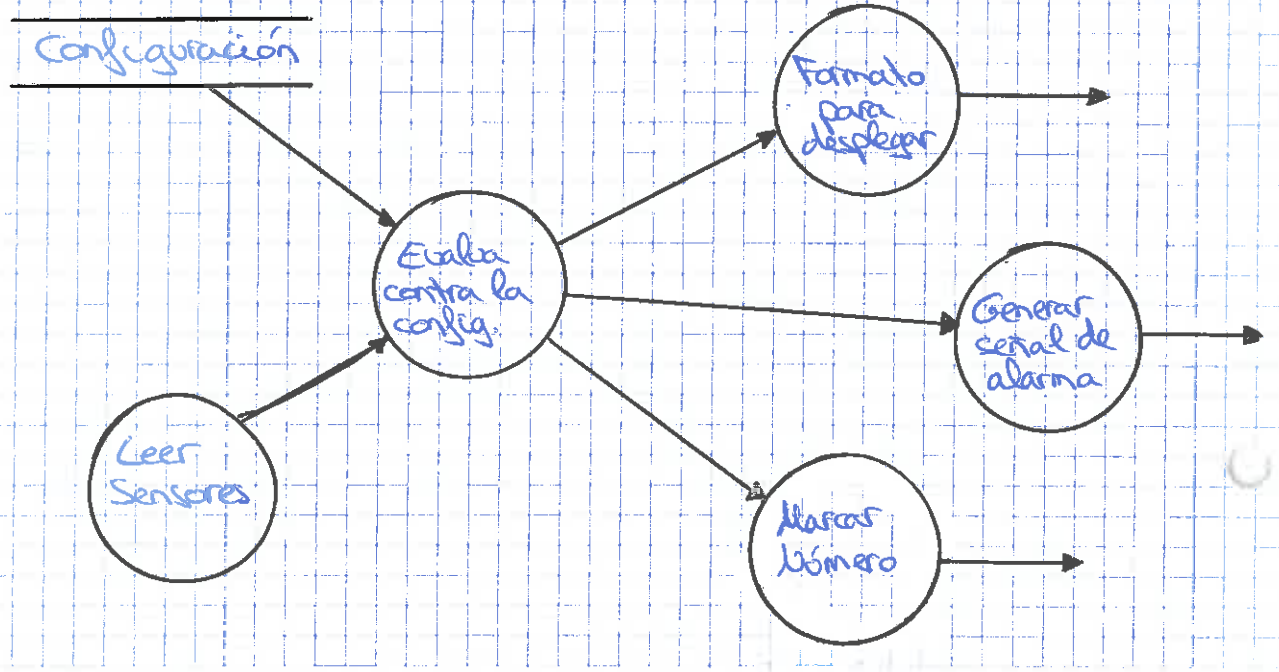
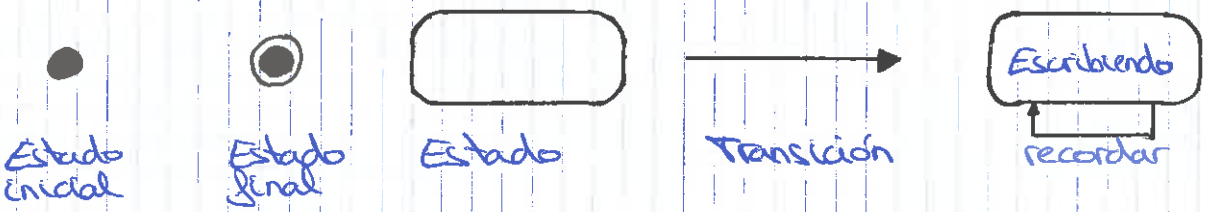
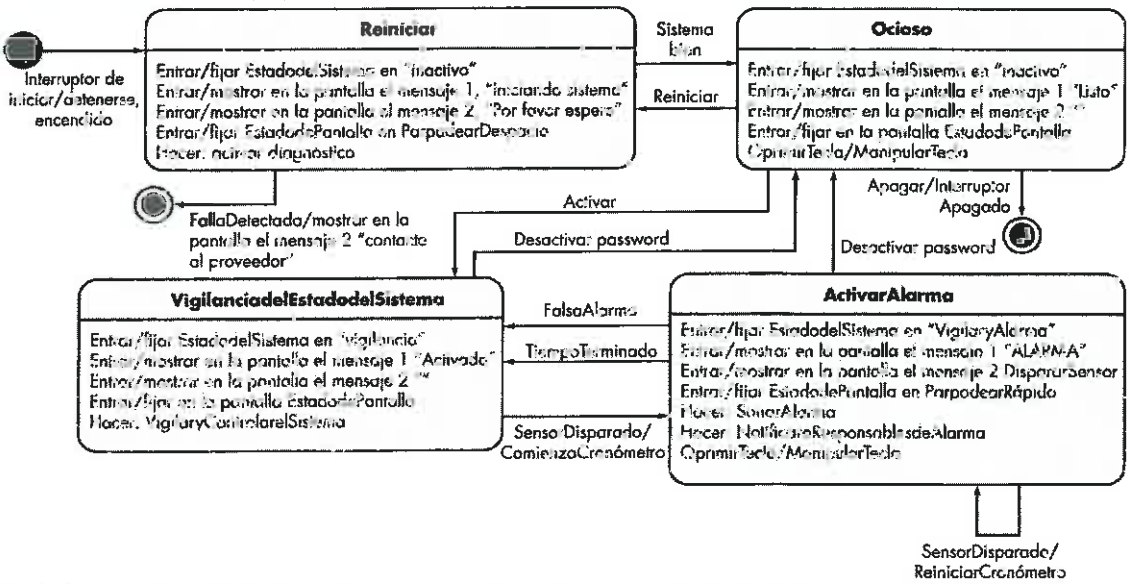


Diagrama de estado: Es aquel que proporciona un estado inicial, uno o varios estados intermedios y un estado final.
Representa un proceso de actividades que deben realizarse para llegar del estado inicial al estado final.
Esto nos permite representar el ciclo de vida del sistema.

Elementos:



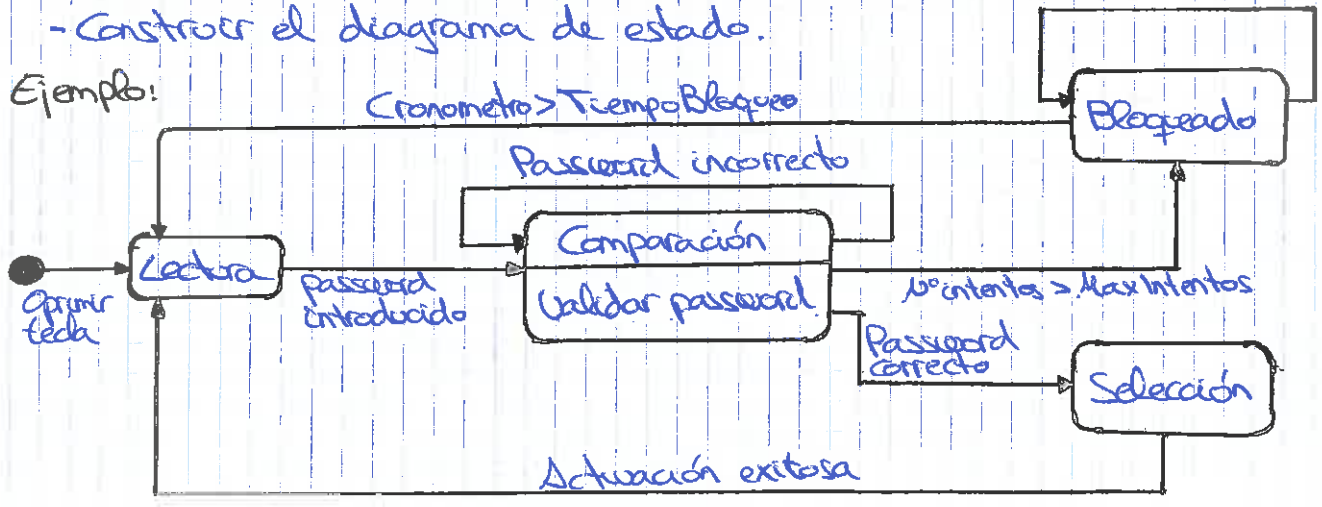
Ejemplo:



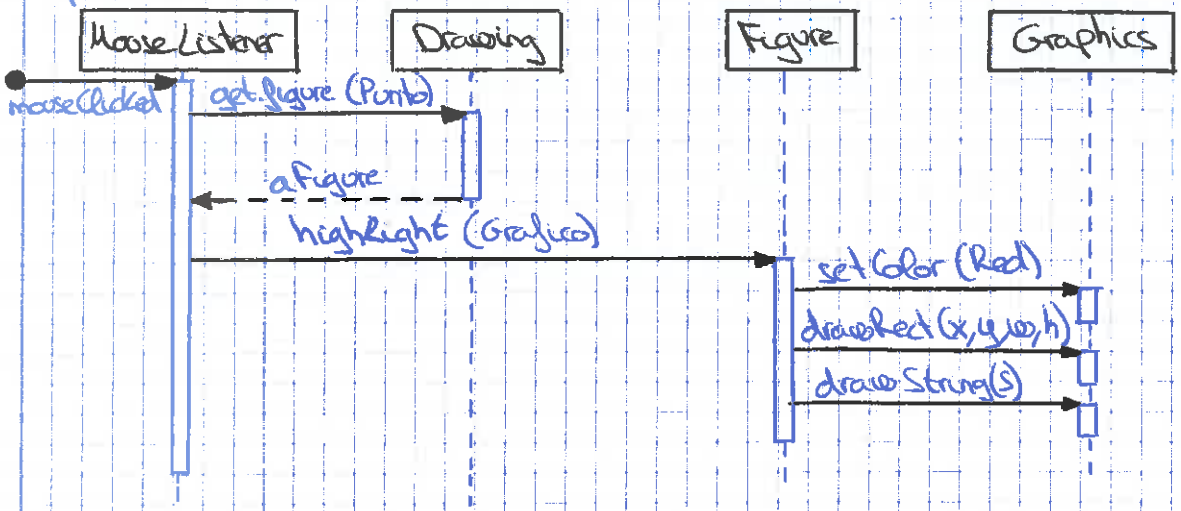
Modelado del comportamiento: Muestra la forma en la que responde el software a eventos externos.
Para generar el modelo han de seguirse los siguientes pasos:
- Evaluar los casos de uso para entender la secuencia de interacción.
- Identificar los eventos que desencadenan las interacciones.
- Crear la secuencia para cada caso de uso.
- Construir el diagrama de estado.

Cronometro bloqueo

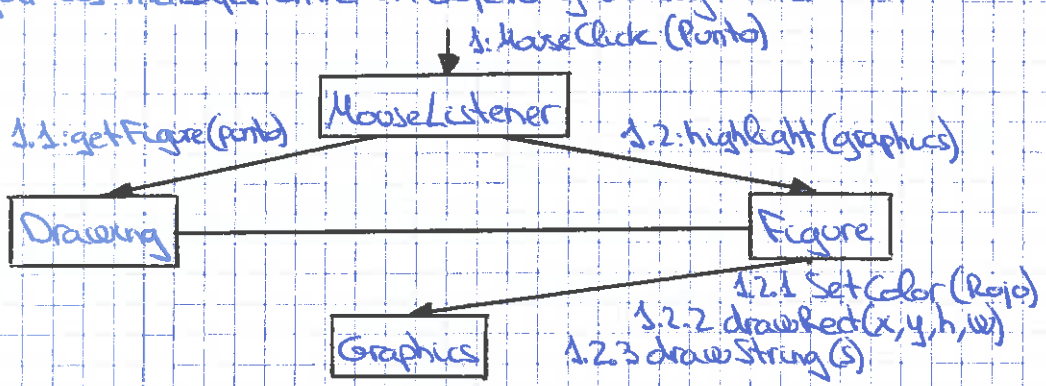
Ejemplo:



Diagramas de secuencia: Es un diagrama que describe como y en que orden un grupo de objetos funcionan en un conjunto para llevar a cabo una tarea.



Diagramas de comunicación: Muestra la comunicación entre los objetos para llevar a cabo una tarea, pero enfatiza las relaciones entre los objetos y clases en lugar del orden temporal. Los mensajes son etiquetados por un número cronológico para poder seguir los mensajes entre un objeto y el siguiente.



Patrones: Son soluciones a un problema que se ha presentado en multiples ocasiones y nos sirve de base para buscar soluciones a problemas de desarrollo de software.
Pueden modificarse para adaptarlos a las necesidades de la aplicación.
Deben cumplir con los requisitos de efectividad y reusabilidad.

*Conceptos del diseño:

El objetivo del diseño, es crear software que cumpla con tres parametros:

- Resistencia: No ha de tener errores que impidan su funcionamiento
- Funcionalidad: Debe cumplir con su proposito.
- Belleza/comodidad: Ha de ser intuitivo y facil de usar.

El diseño del software agropa el conjunto de principios, conceptos y practicas que llevan al desarrollo de un sistema de calidad.

Para conseguir el objetivo del diseño debemos practicar:

- Diversificación: adquisición de alternativas y materia prima.
- Convergencia: Hacer que toda la información adquirida durante la diversificación, concerja en una configuración particular y en la creación del producto final.

El diseño comienza tras el analisis y modelado de requisitos y precede a la construcción del software.

El diseño se realiza a cuatro niveles:

- Diseño de datos o clases: Transforma los modelos de clases en relaciones de clases de diseño.
- Diseño de la arquitectura: Se obtiene del modelado de requisitos.
- Diseño de la interface: Describe como se comunica el software con los sistemas y humanos que interactuan con él. Se obtiene del modelado de casos de uso.
- Diseño de componentes: Describe los componentes en cuanto a la operación que realizan.

La importancia del diseño del software se resume en calidad. Sin diseño se corre el riesgo de tener un sistema inestable que falle y sea dificil someter a pruebas.

El diseño es un proceso iterativo por medio del cual se plasman los requisitos en un plano para construir el software.

1. Implementa todos los requisitos.
2. Ha de ser legible y comprensible para la generación de código.
3. Proporciona una vision completa del software.

- Lineamientos de la calidad: Son los criterios técnicos de un buen diseño y se consiguen aplicando los principios de diseño fundamentales, una metodología sistemática y una supervisión constante.

1. Una arquitectura:
   - Creada con estilos o patrones reconocibles.
   - Compuesta por buenos componentes.
   - Su implementación sea evolutiva.
2. Debe ser modular, es decir, debe de estar dividido en elementos o subsistemas.
3. Debe contener distintas representaciones de datos, arquitectura, interfaces y componentes.
4. Debe conducir a estructuras de datos apropiadas.
5. Debe llevar a componentes con funcionalidades independientes.
6. Debe conducir a interfaces sencillas.
7. Obtenerse con un método que sea repetible.
8. Usar una notación eficaz.

- Atributos de la calidad: Son el conjunto de factores que se usan para establecer las métricas de calidad en las actividades del proceso de desarrollo de software. (FURPS)

Funcionalidad: características y funcionalidades del programa.
Usabilidad: Toma en cuenta factores humanos, la estética, la consistencia y la documentación.
Confiabilidad: Se evalua con la frecuencia y gravedad de los fallos.
Rendimiento: Se mide en base a la velocidad de procesamiento.
Mantenimiento: Capacidad del programa ampliable, adaptable y serviceal.

Conceptos fundamentales:

- Abstracción: Consiste en aislar una propiedad o función concreta de un objeto.
  Por ejemplo, tenemos la clase "puerta". Una abstracción sería la función "abrir".
  - Abstracción de procedimiento: secuencia de instrucciones que tiene una función específica. Siguiendo el ejemplo anterior, sería la descripción de las acciones de la función "abrir".
  - Abstracción de datos: Conjunto de atributos del objeto.

- Arquitectura del software: Es la estructura de organización de los componentes de un programa, la forma en que interactúan y la estructura de datos que utilizan.
  Propiedades:
  - Estructurales: Definen los componentes de un sistema, la manera en que están agrupados e interactúan.
  - Extrafuncionales: Definen la forma en la que se satisfacen los requerimientos.
  - Familias de sistemas relacionados: Debe basarse en patrones.

- Patrones: Son soluciones a un problema que se ha presentado en múltiples ocasiones y nos sirve de base para buscar soluciones a nuestro desarrollo de software.

- División de problemas: Consiste en utilizar la estrategia de divide y vencerás, pues es más fácil resolver un problema complejo si se divide en elementos manejables.

- Modularidad: División del software en componentes que puedan resolverse de forma independiente (módulos) y que cuando se integran, satisfacen los requerimientos del problema.

- Ocultamiento de información: Deben de diseñarse módulos para que la información contenida sea inaccesible.

- Independencia funcional: Cada módulo debe resolver un conjunto específico de requisitos y evitar excesivo contacto con otros módulos.

- Refinamiento sucesivo: Primero se enuncia una función general y mas adelante se va profundizando en los detalles.

- Rediseño: Es el proceso de cambiar un sistema sin alterar el comportamiento externo. Sirve para mejorar el diseño.

Conceptos de diseño orientado a objetos (DOO):

Tipos:

- Clases de interfaz de usuario: Funciones de interacción persona-ordenador.
- Clases de dominio de negocio: Atributos y métodos que implementan elementos del dominio.
- Clases de proceso: Funciones para gestionar las clases de dominio de negocio.
- Clases persistentes: Para el almacenamiento de datos.
- Clases de sistemas: Para la administración y control del software.

Características:

- Completa y suficiente: La clase solo tendra los atributos y metodos necesarios para lograr su objetivo.
- Primitivismo: Solo habra un metodo encargado de realizar algo.
- Mucha cohesión: Atributos y metodos de objetivo único.
- Poco acoplamiento: La colaboración entre clases ha de ser mínima.

* Diseño de la arquitectura:

- Arquitectura del software: es la estructura del sistema, lo que incluye la estructura y organización de los componentes, sus propiedades y conexiones.
Permite analizar la efectividad del diseño para cumplir con los requisitos, considerar alternativas, reducir riesgos y facilitar la comunicación entre todos los participantes.

¿Que es un componente?
Un componente es un modulo de un programa que se puede desarrollar de forma aislada y ofrece un conjunto de servicios a traves de una interfaz.
  - Propiedades: Caracteristicas necesarias para entender como interactuan unos con otros.

¿Cual es la diferencia entre arquitectura y diseño?
La arquitectura sirve para crear muchos diseños, mientras que un diseño usa la arquitectura para resolver un problema especifico.

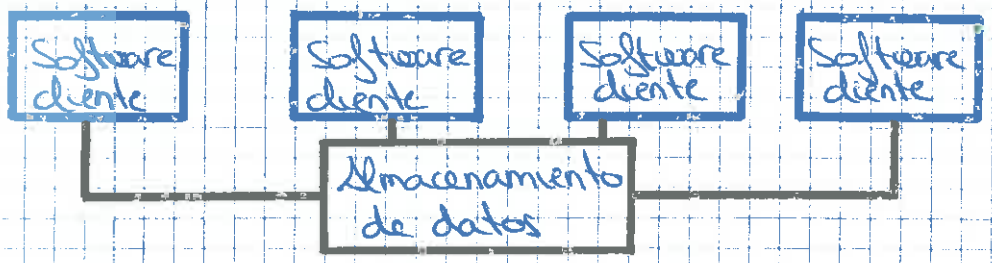
La Arquitectura del software:
- Permite la comunicación entre todos los participantes.
- Resalta las primeras decisiones.
- Muestra de manera simple como esta estructurado el sistema y sus componentes y como estos van a trabajar juntos.

- Géneros arquitectonicos: Describen un enfoque especifico del software que va a construirse.
  - Inteligencia artificial
  - Comunicaciones
  - De autor.
  - Entretenimiento
  - Financiero
  - Medicina
  - Videojuegos
  - Militar
  - Comerciales
  - Sistemas operativos.

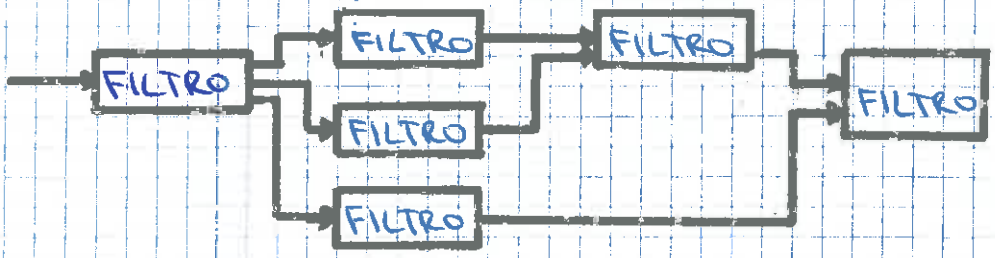
- Estilos arquitectónicos: Es la transformación que se impone al diseño del sistema, su objetivo es establecer la estructura de los componentes del sistema.
Cada estilo define:
  - Un conjunto de componentes
  - Un conjunto de conectores que permiten la comunicación, coordinación y cooperación entre los componentes.
  - Restricciones que definen como se integran los componentes para formar el sistema.
  - Modelos semánticos que permiten entender las propiedades generales del sistema.

Estilos:

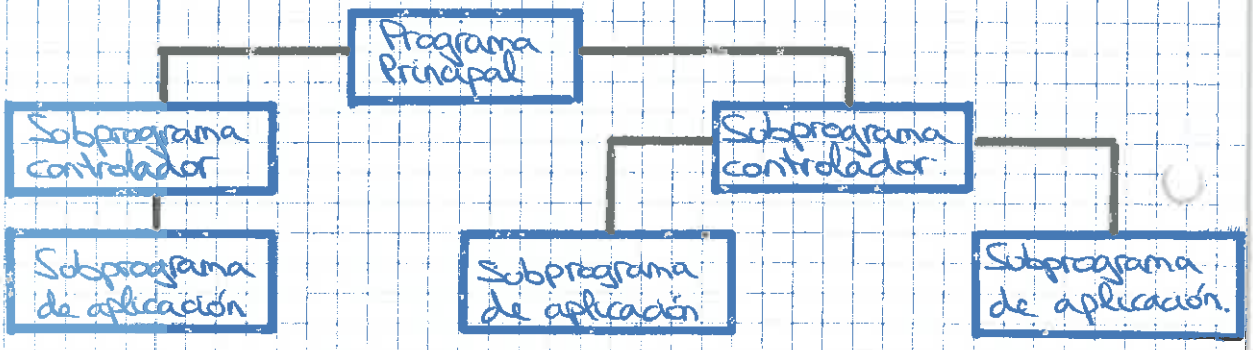
- Arquitectura centrada en datos: Se centralizan los datos y el resto de componentes accede a ellos.



- Arquitectura de flujo de datos: Los datos de entrada van transformandose en datos de salida a través de una serie de componentes.



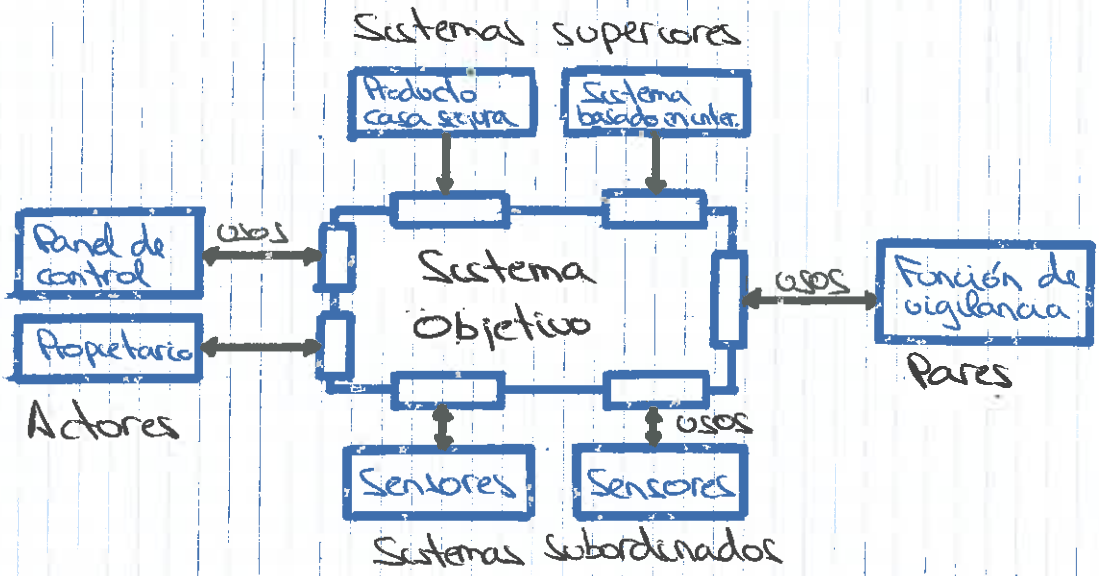
- Arquitectura de llamada y retorno: Arquitectura tradicional de programa principal y subprograma. Facil de modificar.



-Arquitecturas orientadas a objetos: Los componentes incluyen los datos y las operaciones que deben aplicarse para manipularlos. La comunicación y coordinación entre los componentes se consigue mediante la transmisión de mensajes.

-Arquitectura en capas: Se define por una serie de capas, donde cada una ejecuta operaciones del software.

Diseño arquitectonico: Define las entidades externas con las que interactua el software y la naturaleza de la interacción.



Evaluación de los diseños arquitectónicos: El diseño da como resultado varias arquitecturas alternativas, las cuales se deberan evaluar para elegir la mas apropiada.
Hay dos tipos de evaluación:

-Método de negociación para analizar la arquitectura:
- Se desarrollan los casos de uso del sistema.
- Se obtienen los requisitos, restricciones y descripción del entorno.
- Se describen los estilos arquitectonicos.
- Se evalua cada atributo de calidad independientemente.
- Se evalua la criticidad de los atributos de calidad.
- Se evaluan las arquitecturas candidatas.

-Complejidad arquitectónica:
- Dependencias compartidas: si hay componentes que usan los mismos datos.
- Dependencia de flujos: si debe finalizar uno para comenzar otro.
- Dependencia de restricción: dos no pueden ejecutarse al mismo tiempo.

*Diseño en el nivel de componentes:

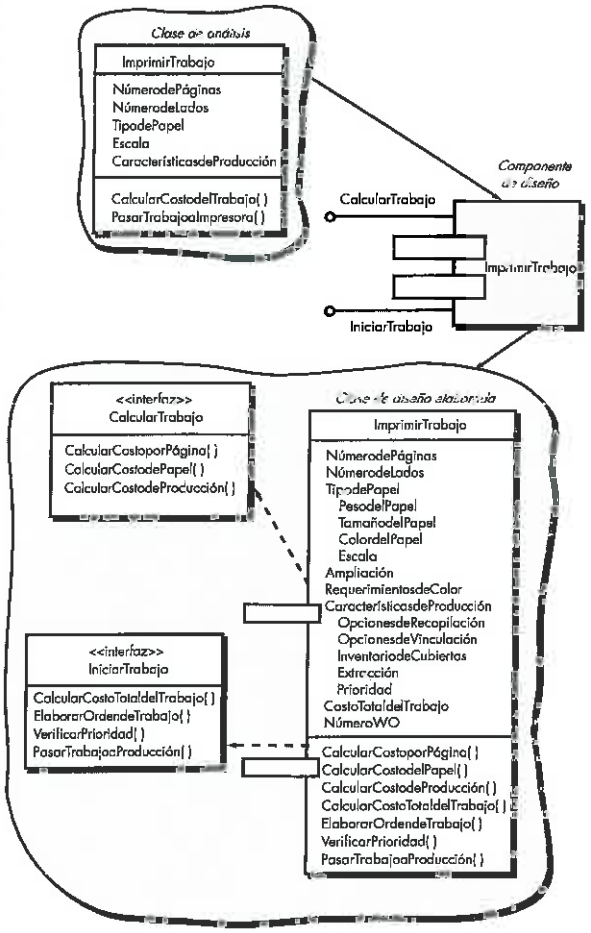
Comienza cuando acaba el diseño de la arquitectura, cuando esta establecida la estructura general de los datos y del programa.

-Componente: Modelo del programa que implementa un conjunto de funcionalidades.

Se puede sustituir con independencia del resto de la arquitectura y es accesible a través de un conjunto de interfases.

-Visión orientada a objetos:

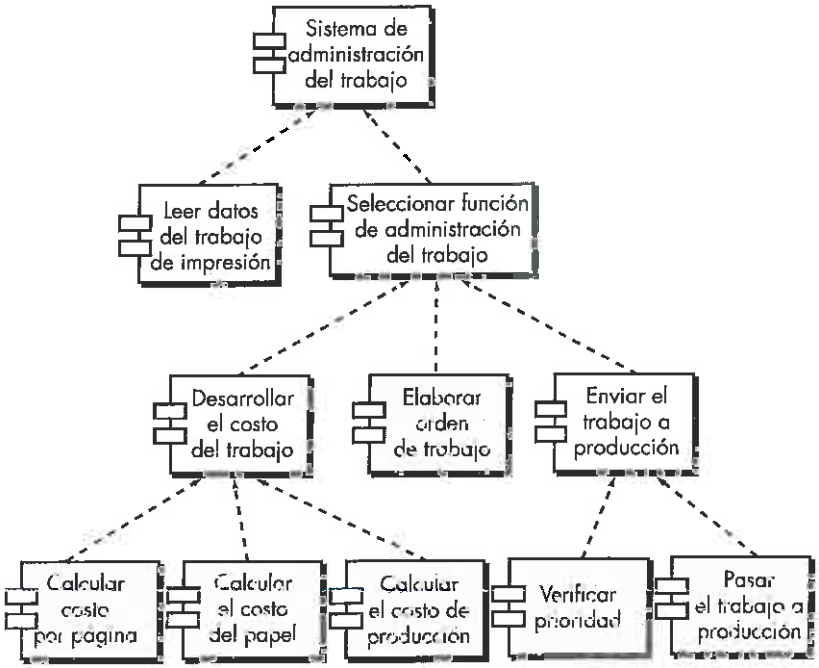
El componente contiene un conjunto de clases que colaboran



-Visión tradicional:

El componente es un elemento funcional del programa. (modulos)

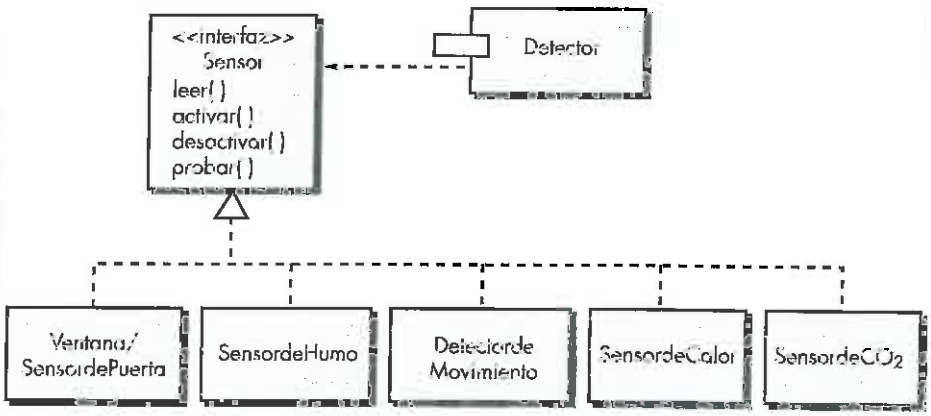
- Componente de control: Invocación de los demas componentes del sistema.
- Componentes del dominio del problema: Implementa una función que requiere el cliente.
- Componente de infraestructura: Apoyo al procesamiento en el dominio del problema.



- Visión relacionada con el proceso:
  El componente se recoge de un catalogo de componentes ya creados o patrones de diseño.
  Se dispone de la descripción de la interface, de las funciones y de la comunicación y colaboración que requieren.

Diseño de componentes basado en clases:

- Principios básicos:
  - Principio Abierto - Cerrado (PAC):
    Un componente debe ser abierto para la extensión y cerrado para la modificación.
    Se crean abstracciones que sirven como intermediarios entre la funcionalidad que sea probable extender y la clase.



  - Principio de sustitución de Liskov (PSL):
    Las subclases deben ser sustituibles por sus clases de base.
  - Principio de inversion de la dependencia (PID):
    Depender de las abstracciones, no de las concreciones.
  - Principio de segregación de la interface (PSI):
    Es mejor tener muchas interfaces especificas que una de proposito general.

- Principios de agrupación de componentes:
  - Principio de equivalencia de la liberación de la reutilización:
    En componentes reutilizables, las actualizaciones han de ser de a pocos, para que el cliente pueda adaptarse.
  - Principio de cierre común:
    Las clases de la misma agrupación cambiaran juntas.
  - Principio de la reutilización común:
    Las clases que no se reutilizan juntas no deben agruparse juntas.

Todo componente debe tener un alto grado de cohesión, es decir, solo contendra los atributos y metodos relacionados con la funcionalidad que desarrolla.

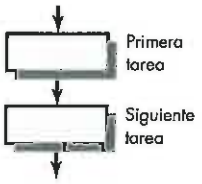
Todo componente debe tener un bajo grado de acoplamiento, reduciendo la interdependencia entre clases.

Tareas a realizar para el diseño de componentes:

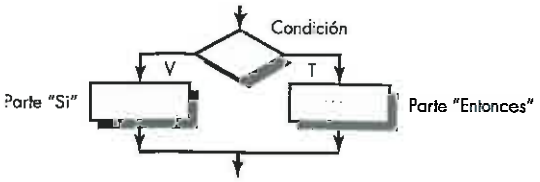
1. Identificar las clases de diseño que correspondan con el dominio del problema.
2. Identificar las clases de diseño que correspondan con el dominio de la infraestructura.
3. Elaborar las clases de diseño que no sean componentes reutilizables.
4. Describir las fuentes de datos e identificar las clases necesarias para administrarlos.
5. Desarrollar y elaborar representaciones del comportamiento para cada clase o componente.
6. Elaborar diagramas de despliegue para dar más detalles a la implementación.

- Diseño de componentes tradicionales:

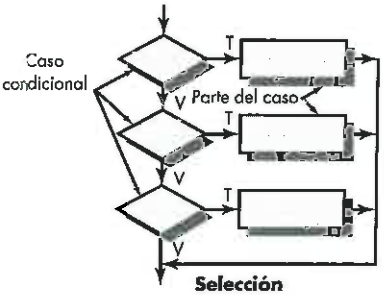
• Con diagramas UML de actividad o de flujo representamos gráficamente la funcionalidad que desarrolla un componente.



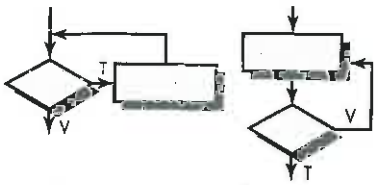
Secuencia



Si-entonces-en otro caso (If-then-else)



Selección



Repetición

• Tablas de decisión: permiten representar combinaciones complejas de condiciones y acciones a realizar en cada caso.

	Reglas					
Condiciones	**1**	**2**	**3**	**4**	**5**	**6**
Cliente regular	T	T				
Cliente plateado			T	T		
Cliente dorado					T	T
Descuento especial	F	T	F	T	F	T
Acciones						
Sin descuento	✓					
Aplicar 8% de descuento			✓	✓		
Aplicar 15% de descuento					✓	✓
Aplicar un porcentaje adicional de descuento		✓		✓		✓

• El pseudocódigo permite representar, con un lenguaje intermedio entre el lenguaje humano y el de las computadoras, la funcionalidad del componente.

Desarrollo basado en componentes:

La ingeniería de software basada en componentes propone el diseño y construcción de sistemas reutilizando componentes ya existentes.

Para que esto sea usable, es necesario:

- Realizar ingeniería de dominio para identificar, catalogar y difundirlos.
- Calificar la idoneidad del componente, identificar las adaptaciones necesarias y las posibles combinaciones con otros componentes.
- Realizar el análisis y diseño para la reutilización del componente.
- Tener una biblioteca que permita la clasificación y recuperación de componentes.

*Diseño basado en patrones:

El diseño basado en patrones crea una aplicación nueva, encontrando un conjunto de soluciones comprobadas para un conjunto de problemas.

Un patrón de diseño es una solución a un problema que se ha presentado en múltiples ocasiones y nos sirve de base para buscar soluciones a problemas de desarrollo del software. En resumen, un patrón contiene una solución probada a un problema en un determinado contexto.

Un sistema de fuerzas son un conjunto de requisitos que influyen en la interpretación del problema en el contexto y en la aplicación de la solución.

Los patrones pueden ser de dos tipos:
- Generativos: Nos dan solución al problema
- No generativos: No ofrecen ninguna solución.

Un patron de diseño eficaz:
- Resuelve un problema.
- Está probado.
- La solución no es obvia.
- Describe una relación.

Clasificación de patrones:
- Arquitectonicos: referentes a arquitecturas del sistema.
- De datos: Modelado de datos.
- De componentes: Desarrollo de componentes.
- De interface: Problemas comunes de interfaz de usuario.
- De creación: Centrados en la creación, composición y representación de objetos.
- Estructurales: Relacionados con como se organizan e integran las clases y objetos para construir una estructura más grande.
- De comportamiento: Asignación de responsabilidad entre los objetos y al modo en que se comunican.

Pasos a seguir para el diseño basado en patrones:

- Examinar el modelado de requisitos y generar una jerarquía de problemas.
- Determinamos si se ha desarrollado un patrón confiable.
- Determinar si se dispone de patrones.
- Si hay patrones diseñaremos con ellos, si no existen aplicaremos otros metodos.
- Repetiremos estos pasos hasta que se refine el diseño.
- Buscaremos el patrón adecuado para la interfaz de objeto.
- Evaluar el diseño.
- Refinar el diseño.

*Calidad del software:

- Calidad del diseño: Grado en el que el diseño cumple las funciones y características especificadas en el modelo de requisitos.

- Calidad de la conformidad: Grado en el que la implementación se apega al diseño y en el que el sistema cumple sus metas de requerimientos y desempeño.

Satisfacción del usuario = Producto que funciona + Buena calidad + Entrega dentro del presupuesto y a tiempo.

La calidad del software es un proceso eficaz de software para crear un producto útil que proporciona un valor medible a quien lo produce y a quien lo utiliza.

Atributos de la calidad (FURPS):

1. Funcionalidad.
2. Fiabilidad.
3. Usabilidad.
4. Eficiencia.
5. Mantenibilidad.
6. Portabilidad.

Hay que ofrecer calidad con costes y tiempos razonables.
Equilibrio entre coste, tiempo y calidad.

Objetivos: Garantizar la calidad de:

- Requisitos
- Diseño
- Código
- Control de calidad.

Que hacer para lograr la calidad del software:

- Metodos de la ingeniería del software:
  - Aplicar correctamente el analisis de requisitos y diseño de software.
  - Diseñar software que cumpla con los factores de calidad.
- Técnicas de administración de proyectos:
  - Crear un plan de proyecto.
  - Gestionar la calidad y el cambio.
  - Gestión de riesgos.
  - Seguimiento permanente del proyecto.
- Control de la calidad:
  - Revisar los modelos
  - Inspeccionar el código.
  - Probar el software
- Aseguramiento de la calidad:
  - Realizar auditorias e informes.
  - Control de calidad sobre el control de la calidad.
    1. Revisiones y auditorias.
    2. Pruebas del software.
    3. Colección y analisis de software
    4. Gestión del cambio
    5. Formación del equipo
    6. Gestión de los proveedores.
    7. Gestión de la seguridad.
    8. Gestión de riesgos.

*Administración de proyectos:

- Personal: El factor humano es el efectivo más importante
  - Gerentes ejecutivos de empresa.
  - Gerentes técnicos de proyectos.
  - Profesionales
  - Clientes
  - Usuarios finales.

El Líder de equipo debe de motivar, organizar y aportar ideas.
Las características del líder son:
- Carisma
- Resolución de problemas y conflictos.
- Equilibrio en el control
- Delegar, reconocer y valorar la productividad del equipo.
- Empatía con el equipo.
- Autocontrol.

Deben evitar:
- Ambiente de trabajo estresante.
- Conflictos entre miembros del equipo.
- Mala organización y coordinación de los procesos.
- Vaga definición de los roles y responsabilidades de los miembros del equipo.
- Sensación de fracaso en el equipo.

Paradigmas de la estructura organizativa:
1. Paradigma cerrado: Jerarquía de autoridad tradicional.
2. Paradigma aleatorio: Organización poco estricta.
3. Paradigma abierto: Mixto entre cerrado y aleatorio.
4. Paradigma síncrono: Dividimos el problema en partes más pequeñas y los miembros del se dividen para ocuparse de cada una de las partes.

- Producto: Es el objetivo de todo proyecto que siempre hay que tener presente.

Tareas:

Determinar el ámbito del producto identificando:

- Contexto
- Información
- Funciones del software a desarrollar.
- Rendimiento del producto
- Descomposición del problema que el producto debe resolver.

- Proceso: Ofrece el entorno para llevar a cabo el plan completo de desarrollo de software.

Fases del proceso:

- Comunicación
- Planificación
- Modelado
- Construcción
- Despliegue

- Proyecto: Planificación, supervisión y control del proyecto.

Practicas fundamentales:

- Administración del proyecto basada en métricas y pautas.
- Estimación empírica de coste y calendario.
- Seguimiento del porcentaje logrado.
- Seguimiento de los objetivos de calidad.
- Correcta gestión del personal.

Como evitar los riesgos:

- Entender el problema a resolver.
- Establecer objetivos realistas.
- Formar un equipo competente y dotarle de autonomía, autoridad y la tecnología necesaria.
- Proporcionar incentivos al equipo.
- Hacer seguimiento constante del proceso.
- Tomar decisiones simples e inteligentes.
- Realizar un análisis tras finalizar el proyecto para extraer las lecciones aprendidas y mejorar el proceso.