



# INGENIERÍA DEL SOFTWARE

Temario completo

## DESCRIPCIÓN:

Resumen completo de todas las transparencias de la asignatura subidas a Moodle.

Nota: En **rojo** se indican los apartados que suelen caer en los exámenes, ya sea de manera teórica o práctica.

Eloy G.

# CONTENIDO

Tema 1: Introducción	Pág. 1
1.1 Crisis del software	Pág. 1
1.2 Ingeniería del software	Pág. 1
Tema 2: Proceso de desarrollo software	Pág. 1
2.1 Modelos de desarrollo	Pág. 1
2.2 Modelos de ciclos de desarrollo	Pág. 2
2.3 Modelos de ciclos de evolución	Pág. 3
Tema 3: Planificación de Sistemas Software	Pág. 3
3.1 Plan de proyecto	Pág. 3
3.2 Planificación temporal	Pág. 4
3.3 Análisis de riesgo	Pág. 6
Tema 4: Análisis de requisitos	Pág. 6
4.1 Definición de requisitos	Pág. 6
4.2 Ingeniería de requisitos	Pág. 7
4.3 Especificación de requisitos	Pág. 7
Tema 5: Técnicas de modelado y especificación	Pág. 7
5.1 Modelo conceptual UML	Pág. 7
5.2 Modelo de comportamiento en UML	Pág. 9
5.3 El modelo estructural en UML	Pág. 12
5.4 Técnicas estructuradas	Pág. 14
Tema 6: Introducción al diseño	Pág. 18
6.1 Principios del diseño	Pág. 18
6.2 Conceptos fundamentales del diseño	Pág. 18
6.3 Diseño modular	Pág. 19
Tema 7: Introducción a las pruebas del software	Pág. 20
7.1 Proceso de pruebas	Pág. 21
7.2 Técnicas de diseño de casos de prueba	Pág. 21



## Tema 1: introducción

### 1.1 Crisis del software

La crisis del software fue un problema que surgió en los **años 70** debido a:

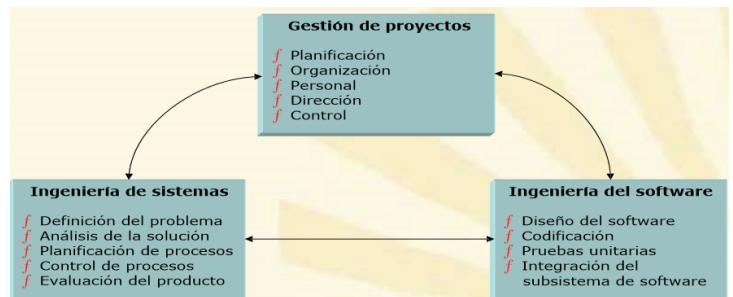
- Nulo cumplimiento de las **expectativas**
- Baja **fiabilidad**
- Altos **costes**
- Necesidad de mucho **mantenimiento**
- Nulo cumplimiento de **plazos**
- Baja **portabilidad**
- Baja **eficiencia**

Sus consecuencias fueron una **baja productividad y calidad**. Para solucionarlo, comenzó a aplicarse la Ingeniería del Software en la construcción de sistemas informáticos tras una conferencia de la OTAN en 1968.

### 1.2 Ingeniería del software

La Ingeniería del Software trata de **planificar, diseñar y reutilizar los diseños al desarrollo de sistemas software utilizando las herramientas apropiadas** (diagramas, actividades, proyectos...).

Los ingenieros de software buscan una **solución adecuada formulando, analizando y especificando** dicha solución mediante la obtención de requerimientos, análisis, diseño del sistema e implementándolo.

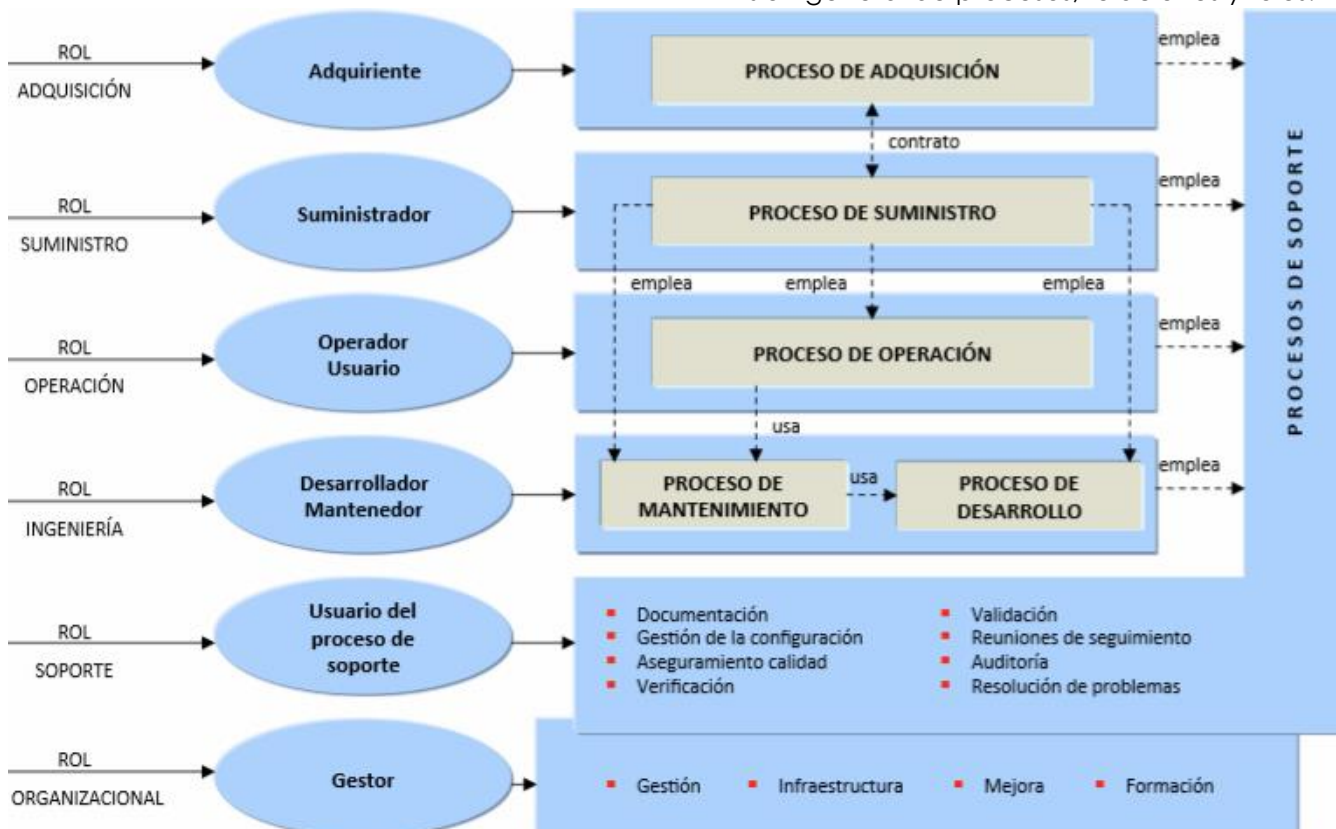


## Tema 2: Proceso de desarrollo software

### 2.1 Modelos de desarrollo

Un proceso de software es el **conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software**. Cuando implica la construcción de un producto, se suele llamar **ciclo de vida** e incluye el periodo de tiempo comprendido desde la definición de los requisitos hasta el fin de su uso. El ciclo de vida presenta unos procesos primarios, como se reflejan en la siguiente imagen.

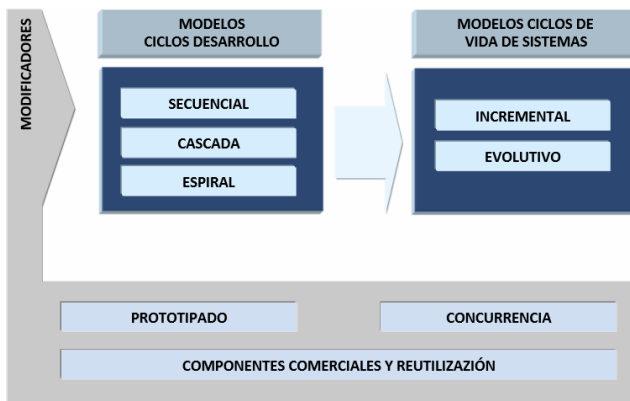
Visión general de procesos, relaciones y roles:





## 2.2 Modelos de ciclos de desarrollo

La aplicación de procesos, tanto en el desarrollo como en el posterior mantenimiento y operación del software siguen unos “patrones fijos” que configuran el esquema de mapa de situación, relación y continuidad entre los diferentes procesos, actividades y tareas. La etapa de desarrollo da paso al ciclo de vida del sistema y en las diferentes etapas del ciclo de vida pueden intervenir modificadores.



### 2.2.1 Secuencial

Este modelo refleja un desarrollo marcado por la sucesión escalonada de las etapas que lo componen: **requisitos – diseño – codificación – pruebas – integración**.

Cada etapa da lugar a la siguiente sólo cuando termina, por lo que resulta muy rígido y genera problemas. Resulta apropiado para desarrollar versiones de sistemas ya veteranos en los que el desconocimiento de las necesidades de usuarios o del entorno no plantea riesgos y para sistemas pequeños.

### 2.2.2 Cascada

Modelo prácticamente idéntico al secuencial, **añade la posibilidad de retornar desde una fase hacia las anteriores con la información generada al avanzar el desarrollo**. Este modelo reconoce la importancia de disponer de unos requisitos y un diseño previo antes de comenzar con la codificación del sistema. Resulta apropiado para los mismos casos que el secuencial.

### 2.2.3 Espiral

Este diseño presenta un desarrollo evolutivo, además de introducir el análisis de riesgo para guiar la evolución del proceso de desarrollo.

El ciclo de iteración de este modelo se convierte en una espiral, que al representarse sobre ejes cartesianos muestra en cada cuadrante una clase particular de actividad: planificación, análisis de riesgo, ingeniería y evaluación. La dimensión angular representa el avance relativo en el desarrollo de las actividades de cada cuadrante. En cada ciclo de la espiral se realiza una parte del desarrollo total, a través de los cuatro tipos de actividades:

- En la **planificación** se establece el contexto del desarrollo y se decide qué parte del mismo se abordará en el ciclo siguiente.
- En **análisis de riesgo** se evalúan las alternativas posibles para la ejecución de la siguiente parte del desarrollo, seleccionando la más ventajosa y previendo los riesgos posibles.
- Las actividades de **ingeniería** corresponden a las indicadas en los modelos lineales: análisis, diseño, codificación, etc.
- Las actividades de **evaluación** analizan los resultados de la fase de ingeniería, tomando el resultado de la evaluación como punto de partida para el análisis de la siguiente fase.

### 2.2.4 Proceso Unificado de desarrollo

Está basado en componentes interconectados a través de interfaces, **utiliza UML** para desarrollar los esquemas y diagramas de un sistema software. Está **dirigido por casos de uso**, se centra en la arquitectura y es **iterativo e incremental**.

Los casos de uso especifican los requisitos de un sistema y representan los requisitos funcionales, juntos constituyen el modelo de casos de uso, que describe la funcionalidad total del sistema. Además, guían el proceso de desarrollo: a partir de él se crean y prueban modelos de diseño e implementación conforme a ellos.

El proceso unificado se repite a lo largo de una serie de ciclos, cada ciclo concluye con una versión del producto y consta de cuatro fases:

- **Inicio:** descripción y análisis del producto final a partir de una idea inicial, se buscan las principales funciones del sistema, los usuarios más importantes, se plantea la



arquitectura y se crea el plan del proyecto.

- **Elaboración:** Se especifican en detalle los principales casos de uso, se diseña la arquitectura del sistema y se planifican las actividades y recursos necesarios para finalizar el proyecto.
- **Construcción:** Se crea el producto añadiendo el software a la arquitectura.
- **Transición:** periodo en el cual los usuarios prueban el producto e informan de defectos y deficiencias (beta). Se corrigen problemas e incorporan sugerencias.

## 2.3 Modelos de ciclos de evolución

### 2.3.1 Incremental

Este modelo mitiga la rigidez del modelo en cascada, descomponiendo el sistema en partes, en las cuales se aplica un ciclo de desarrollo.

Las ventajas que ofrece son: el usuario tiene al alcance subsistemas que ayudan a perfilar las necesidades, además de permitir entregas parciales en periodos cortos de tiempo.

### 2.3.2 Evolutivo

Modelo formado por varios ciclos de desarrollo, cada uno produce un sistema completo.

Es un ciclo de vida común a todos los sistemas de desarrollo que se mejoran en versiones sucesivas.

## 2.4 Modificadores de los modelos

### 2.4.1 Prototipado

El prototipado consiste en la **construcción de modelos de prueba**, que simulen el funcionamiento que se pretende conseguir en el sistema, pueden ser ligeros u operativos.

Aunque ofrece muchas ventajas, implica algunos riesgos:

Si se trata de un prototipo operativo, puede empezar a crecer más allá de los objetivos previstos desbordando agendas y recursos.

Si se trata de un prototipo ligero desarrollado fuera del departamento de desarrollo puede

mostrar al cliente funcionalidades no implementables.

### 2.4.2 Concurrencia

Consiste en el solapamiento de un proceso sobre otro, puede aportar beneficios sobre la planificación de un proyecto de software, o por el contrario ser origen o consecuencia de problemas. Se deben tener en cuenta:

- **Índice de concurrencia:** Se produce en un grado reducido, generando un escaso flujo de modificaciones; o por el contrario se da de forma intensiva generando situaciones problemáticas en la planificación o en la distribución del trabajo.
- Resulta importante la labor de **gestión del proyecto** para tratarla de forma adecuada con el mayor beneficio, o el menor perjuicio a los planes y calidad del proyecto.

### 2.4.3 Componentes comerciales y reutilización

El uso de componentes o partes ya desarrolladas tienen implicaciones en el ciclo de desarrollo, diferentes según las circunstancias. Así, por ejemplo, si gran parte del sistema consta de componentes ya desarrollados y probados, el periodo de pruebas se acortará.

Si un proyecto va a delegar funcionalidades críticas en un componente comercial, que no ha empleado previamente la organización desarrolladora, es posible que incorpore en su ciclo de desarrollo una fase de pruebas de ese componente para obtener la certeza de que el componente se comporta como se espera.

## Tema 3: Planificación de Sistemas Software

### 3.1 Plan de proyecto

Un proyecto es un conjunto de etapas, actividades y tareas que tiene como finalidad alcanzar un objetivo. El contenido de un Plan de Proyecto es siempre variable, sin embargo, es recomendable incluir, al menos, los siguientes elementos:

- Objetivos del proyecto
  - Definición y funciones principales
- Estimaciones del proyecto
  - Datos históricos utilizados



- Técnicas utilizadas
- Resultados de las estimaciones
- Riesgos del proyecto
  - Identificación y evaluación
  - Planificación y control de riesgo
- Recursos
  - Personal, Hardware y Software
- Organización del personal
- Agenda
  - Red de tareas (Grafos PERT/ROY)
  - Diagrama de Gantt
  - Recursos por tarea
- Mecanismos de seguimiento y control

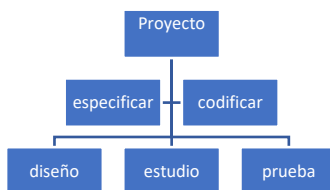
### 3.2 Planificación Temporal

Para la planificación temporal de un proyecto, primero se descompondrá en un número manejable de tareas, determinando las dependencias de cada una y asignándoles un cierto de unidades de trabajo, una fecha de inicio y otra de finalización, así como un responsable.

Se deberá definir el resultado y asociarlo con hito del proyecto. Un hito se considera cuando se ha revisado la calidad de uno o más productos y se han aceptado.

#### 1. Descomposición de tareas

La estructura de descomposición de trabajo (WBS) es un método para representar de forma jerárquica los componentes de un proceso o producto. Por ejemplo:



#### 2. Especificación de tareas

Se especifican las tareas descomponiéndolas y añadiéndoles detalles como nombre, descripción, esfuerzo estimado, etc. Por ejemplo:

Especificación de tarea	
<b>Número:</b>	3.1.
<b>Nombre:</b>	Diseño B.D.
<b>Descripción:</b>	Se diseñará la base de datos, partiendo del modelo entidad-relación propuesto en el análisis y con el objetivo de tener un sistema funcionando sobre DB2.
<b>Esfuerzo Estimado:</b>	2 semanas/hombre
<b>Entregables:</b>	Estructura de implementación de la B.D.
.....:	.....

#### 3. Paralelismo de tareas

Cuando más de una persona está involucrada en un proyecto, puede que varias actividades puedan llevarse en paralelo. El planificador debe determinar la dependencia entre las tareas para asegurar el progreso continuo del proyecto.

#### 4. Distribución de esfuerzos

Una distribución recomendada es la que potencia la fase de análisis y diseño y las pruebas del software, es decir:

- 40% Análisis y diseño
- 20% Codificación
- 40% Prueba y depuración

#### 3.2.1 Pasos en la Planificación Temporal

El punto de partida debe ser el diagrama WBS y las fichas de cada tarea signadas. En función del uso de recursos y las necesidades del usuario ordenaremos las tareas y crearemos un calendario y camino crítico.

#### Ordenación de tareas

Para ello se deben identificar y documentar las dependencias, que pueden ser:

- Restricciones: factores impuestos por el cliente que limitan las opciones del equipo de desarrollo.
- Supuestos: Factores que se consideran verdaderos durante la planificación.
- Dependencias obligatorias: Son los aspectos técnicos. Por ejemplo: "prueba del programa x", debe ser precedida de "codificación del programa x".
- Dependencias discrecionales: Las define el equipo del proyecto, se basan en "mejoras prácticas".
- Dependencias externas: Vienen impuestas desde el exterior, referentes a la interdependencia con otros proyectos o empresas.

#### 3.2.2 Métodos de Planificación Temporal

Los principales métodos son:

- Diagrama de Gantt
- Diagrama de precedencias
- Diagrama de flechas

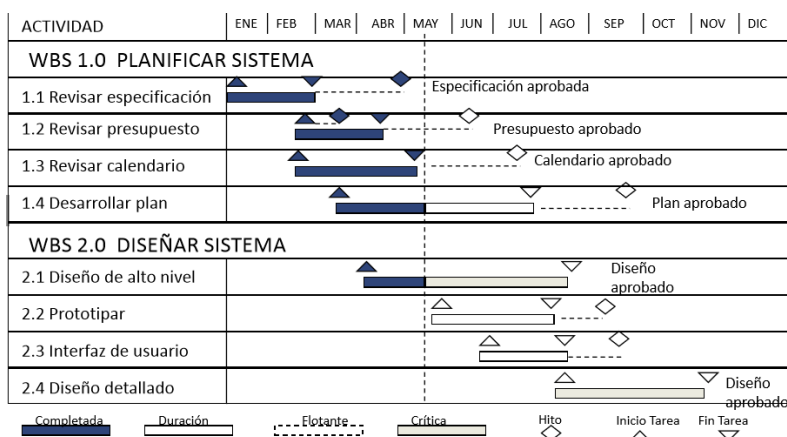




Sus características comunes son las de identificar las tareas que determinan la duración del proyecto, establecer estimaciones de tiempo para las tareas y calcular los tiempos límite que definan el espacio temporal de cada tarea.

### Diagrama de Gantt

Es un método de la planificación temporal de un proyecto que sirve para representar de manera gráfica la secuencia de tareas del proyecto, así como el tiempo de dedicación previsto para cada tarea a lo largo de un tiempo total determinado. Como inconveniente, este diagrama no indica las relaciones existentes entre actividades. Ejemplo:



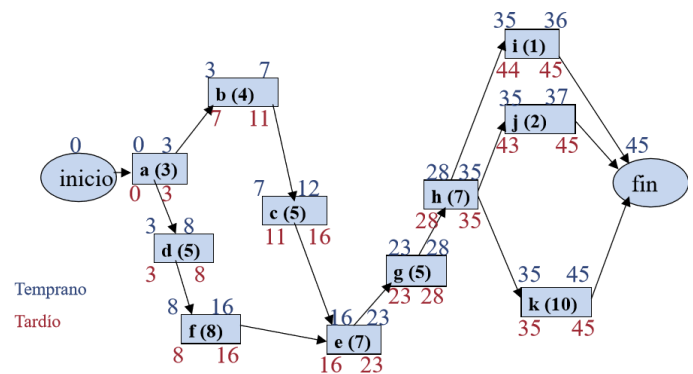
### Diagrama de precedencia (CPM)

Es una herramienta para la programación de actividades en la planeación de un proyecto. Es un método de construcción de un diagrama de red del cronograma del proyecto que utiliza nodos para representar actividades y las conecta con flechas que muestran las dependencias, formando un grafo ordenado.

Todos los nodos tienen el mismo tamaño, representan la siguiente información:

Etiqueta actividad		Duración	
Inicio temprano	DESCRIPCIÓN DE LA ACTIVIDAD	Final temprano	
Inicio tardío		Final tardío	
Máximo tiempo disponible		Holgura	

- Descripción de la actividad: Nombre
- Etiqueta de la actividad: Número que la identifica.
- Duración: Tiempo que calculamos que se tardará en completar la tarea.
- Inicio temprano: Fecha más temprana en que puede comenzar la tarea
- Final temprano: Fecha más temprana en que puede finalizar la tarea.
- Inicio tardío: Fecha más retrasada en la que se puede comenzar sin que afecte a la fecha de terminación del proyecto.
- Final tardío: Fecha más retrasada en la que se puede terminar la tarea sin que afecte a la fecha final del proyecto.
- Máximo tiempo disponible: Tiempo máximo que puede durar una tarea en caso de comenzar en su Inicio temprano y concluir en su Final tardío
- Holgura: Tiempo que disponemos para jugar con el inicio de la tarea sin afectar al proyecto.



Para calcular las fechas de cada tarea en el proyecto: Partimos del diagrama de precedencias (mostrado arriba). Asignamos como inicio temprano "0" a todas las tareas sin predecesor.

El final temprano de cada tarea es el inicio temprano más su duración.

Si la tarea tiene predecesoras y todas éstas tienen calculado su final temprano, asignamos como inicio temprano el máximo de todos ellos.

Para obtener la fecha de final del proyecto partimos de la máxima fecha de final temprano. A todas las tareas que no tengan sucesoras se le asigna esta fecha como final tardío.

El inicio tardío se calcula restando al final tardío la duración. Aquellas tareas con sucesoras, se les asigna como final tardío el mínimo de los inicios tardíos de estas.



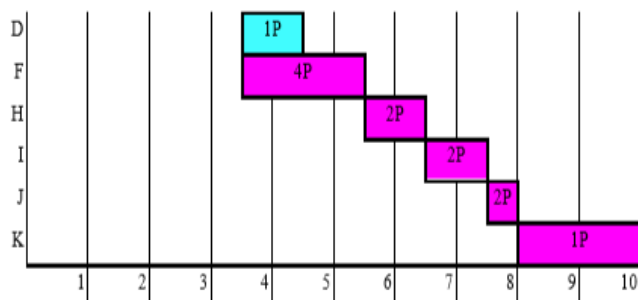
Máximo tiempo disponible = Final tardío – inicio temprano.

Holgura = Máximo tiempo disponible – duración.

El camino crítico es el conjunto de tareas con holgura cero y termina en una tarea sin sucesoras

### Diagrama de Flechas o PERT

Es un método para analizar las tareas involucradas en completar un proyecto dado, especialmente el tiempo para completar cada tarea e identificar el tiempo mínimo necesario para completar el proyecto total.



Las diferencias principales entre CPM y PERT son: PERT es más utilizado en ingenierías clásicas y no de software, en PERT los nodos se representan con principio y fin y en CPM son actividades. Además, la conexión entre nodos PERT indica el inicio de tareas, en CPM indica el tiempo entre tareas.

### 3.3 Análisis de riesgo

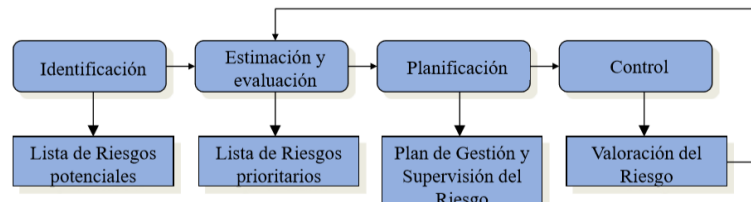
El análisis del riesgo tiene que ver con la identificación de los riesgos y los planes para minimizar sus efectos en el proyecto. Existen tres categorías:

- **Riesgos del proyecto.** Identifican problemas de recursos y de requisitos. Afectan al coste y duración del proyecto.
- **Riesgos del producto.** Problemas en el diseño, implementación, interfaz, etc. Afectan a la calidad del Software resultante
- **Riesgos del negocio.** Cambio en la dirección, de estrategia y de mercado.

El proceso de gestión del riesgo sigue las siguientes pautas:

- **Identificación del riesgo:** identificamos los posibles riesgos de proyecto en cada situación.

- **Estimación y evaluación del riesgo:** determina la probabilidad y consecuencia del riesgo, puede ser expresada de manera cuantitativa o cualitativa.
- **Planificación del riesgo:** Trazar un plan para evitar o minimizar ocurrencia de un riesgo, diseñan estrategias para evitar riesgos.
- **Control del riesgo:** controla la ocurrencia de riesgos a lo largo del proyecto.



## Tema 4: Análisis de requisitos

### 4.1 Definición de requisitos

Los requisitos describen los servicios que debe proporcionar el sistema y sus restricciones operativas, deben ser completos y consistentes. Existen dos niveles de requisitos:

- **Requisitos de Usuario,** descripción a grandes rasgos sobre lo que se espera del sistema y las restricciones bajo las cuales debe funcionar
- **Requisitos de Sistema.** Establecen con detalle las funciones, servicios y restricciones operativas del sistema, pueden ser:
  - **Funcionales:** Servicios que debe proporcionar el sistema, cómo se debe reaccionar a determinadas entradas, etc.
  - **No funcionales:** Son restricciones de los servicios o funciones que ofrece el sistema. Surgen de las necesidades del usuario y suelen ser más críticos que los funcionales, dentro de éstos tenemos: **Del producto, organizacionales y externos.**
  - **De dominio:** Reflejan características y restricciones sobre requisitos funcionales que ya existían.

Las características que debe tener un requisito son: Identificador, Autor, Tipo de requisito,





Descripción, Prioridad, Estado y Fecha de creación.

## 4.2 Ingeniería de requisitos

Es un proceso de **descubrimiento, refinamiento, modelado y especificación**. Se refinan en detalle los requisitos del sistema y el papel asignado al software. Permite al ingeniero especificar las características operacionales del software, incluye la interfaz del software con otros elementos del sistema y establece las restricciones que debe de cumplir el software.

## 4.3 Especificación de requisitos

El objetivo de la especificación de requisitos es obtener un documento que defina de forma completa, precisa y verificable los requisitos que debe cumplir el sistema tanto funcionales como no funcionales y de diseño.

Una buena especificación de requisitos debe cumplir una serie de características.

- No ambigua
- Completa
- Fácil de verificar
- Consistente
- Clasificada por importancia/estabilidad
- Fácil de modificar
- Fácil identificación del origen y consecuencias de cada requisito
- De fácil utilización durante la fase de explotación y mantenimiento.

Las principales actividades que se llevan a cabo para obtenerla son:

- **Extracción de requisitos:** Proceso por el cual los clientes descubren, revelan y comprenden los requisitos que desean.
- **Análisis de requisitos:** Procesado de los anteriores requisitos con el fin de eliminar inconsistencias y conflictos entre ellos para coordinarlos.
- **Especificación de requisitos:** Proceso de redacción y registro de requisitos.
- **Validación de requisitos.** Comprobación de la validez, consistencia y completitud de los requisitos especificados.

Las principales técnicas de captura y análisis de requisitos son: entrevistas, desarrollo conjunto de apps, prototipado y observación.

## Tema 5: Técnicas de modelado y especificación

### 5.1 Modelo Conceptual UML

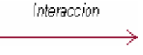
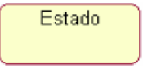

UML es un lenguaje para **visualizar, especificar, construir y documentar a través de varias perspectivas** los modelos de un sistema que involucra gran cantidad de software, desde una perspectiva orientada a objetos.


Para comprender UML, se necesita adquirir un modelo conceptual del lenguaje. Esto requiere aprender a utilizar tres elementos principales:

#### 5.1.1 Elementos

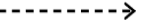


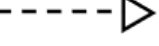
Elementos estructurales		
Clase		Es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones y relaciones.
Caso de uso		Es una descripción de un conjunto de secuencias de acciones que un sistema ejecuta y que produce un resultado observable.
Interfaz		Es una colección de operaciones que especifica un servicio de una clase o componente.
Componente		Es un empaquetamiento físico de diferentes elementos lógicos como clases, interfaces, etc.
Artefacto		Representa el empaquetamiento físico de código fuente o información en tiempo de ejecución
Nodo		Un nodo es un elemento físico que existe en tiempo de ejecución, representado por un recurso computacional que por lo general dispone de algo de memoria o capacidad de procesamiento

Elementos de agrupación		
Paquetes		Sólo existen en tiempo de ejecución. Su propósito general es ser un mecanismo para organizar elementos

Elementos de comportamiento		
Interacción		Comprende un conjunto de mensajes que se intercambian entre un conjunto de objetos, para cumplir un objetivo específico
Máquinas de estados		Especifica la secuencia de estados por los que pasa un objeto o una interacción, en respuesta a eventos
Actividad		Es un comportamiento que especifica la secuencia de pasos que ejecuta un proceso computacional. Un paso de una actividad se denomina acción.

Elementos de notación		
Notas		Son comentarios que se aplican para describir, clarificar y hacer observaciones sobre cualquier elemento de un modelo

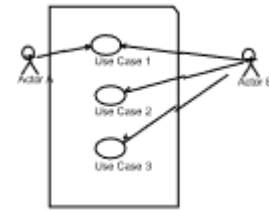
### 5.1.2 Relaciones

Tipos de relaciones		
Dependencia		Relación semántica en la cual un cambio de un elemento puede afectar la semántica de otro
Asociación		Relación estructural que describe un conjunto de conexiones entre objetos.
Generalización		Es una relación de especialización en la cual los objetos de un elemento hijo son consistentes con los objetos de un elemento padre.
Realización		Relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro garantiza que se va a llevar a cabo

### 5.1.3 Diagramas

Un diagrama es la representación gráfica de un conjunto de elementos conectados entre sí. Estos diagramas tienen forma de grafos conectados donde los vértices representan elementos y los arcos relaciones. Sirven para visualizar un sistema desde diferentes perspectivas:

- **Estructurales:** Para aspectos estáticos de un sistema.



1-Diagrama de Casos de Uso



2-Diagrama de Clases



3-Diagrama de Componentes



4-Diagrama de Objetos

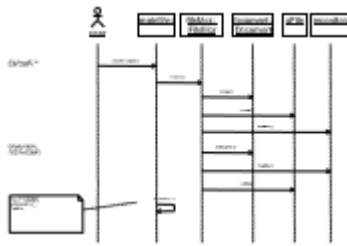


5-Diagrama de Despliegue

- **De comportamiento:** Para elementos dinámicos de un sistema. Diagramas de Actividad, de Estados;



6-Diagrama de Colaboración



7-Diagrama de Secuencia

#### 5.1.4 Reglas de uso

Los bloques de construcción no se pueden combinar de cualquier manera, existen reglas semánticas para:

- Nombres
- Alcance
- Visibilidad
- Integridad
- Ejecución

## 5.2 Modelo de comportamiento en UML

### 5.2.1 Interacciones

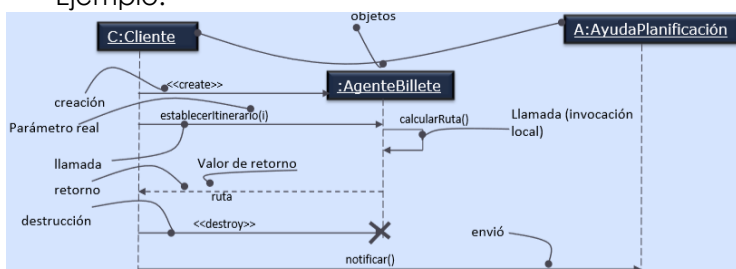
Una interacción es un comportamiento que incluye mensajes intercambiados por objetos dentro de un contexto para lograr un propósito. Son utilizadas para modelar el comportamiento dinámico de las colaboraciones, en este contexto podemos encontrar clases, interfaces, componentes, nodos y casos de uso.

Los aspectos dinámicos se visualizan, se especifican, se construyen y se documentan como flujos de control.

Un mensaje es la especificación de una comunicación entre objetos que transmiten información con la expectativa de que se desencadenará una actividad que puede ser:

- Llamada
- Retorno
- Envío
- Creación
- Destrucción

Ejemplo:



**Secuenciación:** Cuando un objeto envía un mensaje a otro y éste envía un mensaje a un tercero, se forma una secuencia, cada proceso o hilo dentro de un sistema define un flujo de control separado y en él los mensajes se ordenan en secuencia según se suceden en el tiempo.

### 5.2.2 Diagramas de Casos de Uso

Un caso de uso es un diagrama que describe las principales secuencias de iteración entre el sistema y los actores externos.

Un diagrama de casos de uso especifica el comportamiento de un sistema sin tener que especificar como se implementa dicho comportamiento, y es una descripción de unas secuencias o acciones que ejecuta el sistema para producir un resultado, tienen un equilibrio entre genérico y específico. Pueden servir de herramienta de validación del sistema

**Actores:** Pueden ser **principales** (personas que usan el sistema), **secundarios** (personas que lo mantienen y administran), ser **material externo u otros sistemas** con los que se interactúa. La misma persona puede interpretar varios papeles y sus nombres describen el papel desempeñado.

Tipos de casos de uso:

- **Trazo grueso:** Sólo se incluyen las alternativas de interacción entre el actor y el sistema más relevantes.
- **Trazo fino:** Completan todos los detalles que no se han especificado anteriormente además de añadir todas las posibles alternativas.
- **Temporales:** Aquellos en los que el inicio de dicho caso está determinado por el paso del tiempo.
- **Primarios:** Son los que se corresponden con los procesos de negocio.
- **Secundarios:** Son necesarios para que el sistema funcione normalmente.

### Proceso de Análisis de Requisitos:

1. Identificar los actores
2. Identificar los principales casos de uso de cada actor
3. Identificar nuevos casos a partir de los existentes
4. Crear descripciones de casos de uso de trazo grueso

5. Definir prioridades y seleccionar casos de la primera iteración

6. Escribir los casos de trazo fino y crear prototipos de interfaces.

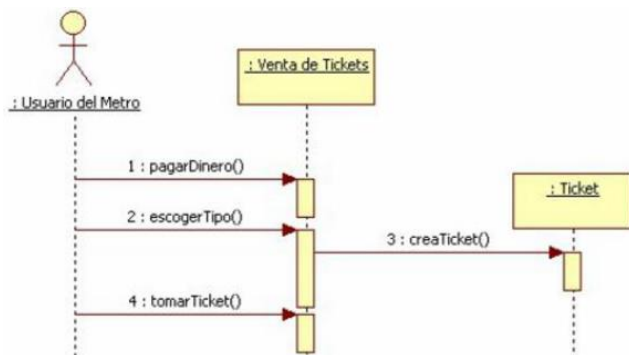
### 5.2.3 Diagramas de interacción:

Modelan el comportamiento dinámico del sistema, describen la interacción entre objetos y los elementos que contienen son: objetos, enlaces y mensajes. Existen dos tipos:

- Diagramas de Secuencia (temporal)
- Diagramas de Colaboración (estructural)

### Diagrama de Secuencia

Es un diagrama en el que se destaca la ordenación temporal de los eventos. Gráficamente es una tabla que representa a objetos, dispuestos a lo largo del eje X, y mensajes, ordenados según se suceden en el tiempo, a lo largo del eje Y. Tienen una **línea de vida** (discontinua vertical) que representa la existencia de un objeto a lo largo del tiempo. También tienen un **foco de control**, que es un rectángulo delgado y estrecho que representa el periodo de tiempo durante el cual un objeto ejecuta una acción.



Tipos de mensajes:

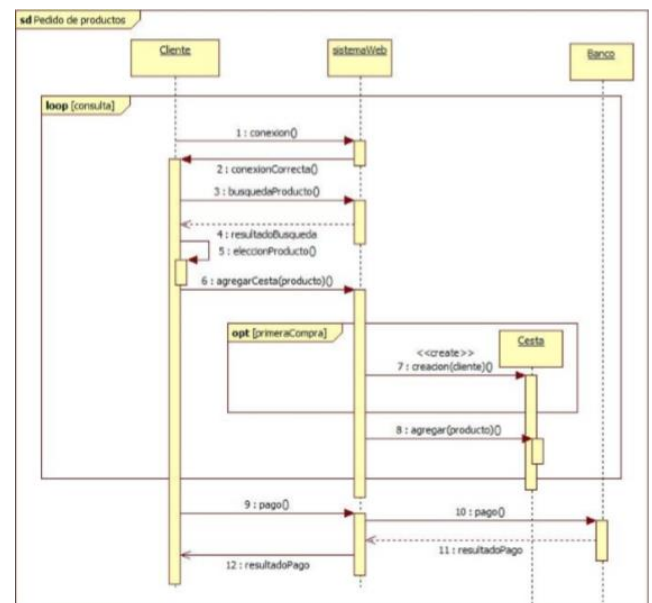
- **Mensaje Simple / Sincrónico**  
*No se dan detalles de la comunicación cuando no son conocidos o no son relevantes.*
- - - - - → **Respuesta / Resultado**
- **Mensaje Asincrónico**

Como operadores de control, en UML 2.X se utilizan marcos de iteración, que son una parte del diagrama asociado a una etiqueta, la cual contiene un operador de control que determina

el modo de ejecución de esa secuencia, que puede ser:

- Ejecución opcional (opt): Se trata de una alternativa, que se obtiene utilizando el operador *opt* seguido de una condición de test. El marco se ejecuta si dicha condición es cumplida.
- Bucle (loop): El marco se ejecuta mientras se cumple una condición, se efectuará mediante el operador *loop*, seguidos de los parámetros opcionales *min* y *max*

Ejemplo de operador de control:



### Diagrama de colaboración

Destacan la organización de los objetos que participan en una interacción. En este grafo los nodos son objetos y los arcos enlaces que se etiquetan con los mensajes que envían y reciben los objetos.

Dan una visión del flujo de control en el contexto de la organización de los objetos que colaboran. Tienen dos características que los diferencian de los diagramas de secuencia:

- El **camino**: Para indicar cómo se enlazan los objetos se utilizan estereotipos en los extremos de los enlaces (local, global y self)
- El **número de secuencia**: Para indicar la ordenación de los mensajes se utiliza la numeración decimal de Dewey (1, 1.1, ...)



Ejemplo:

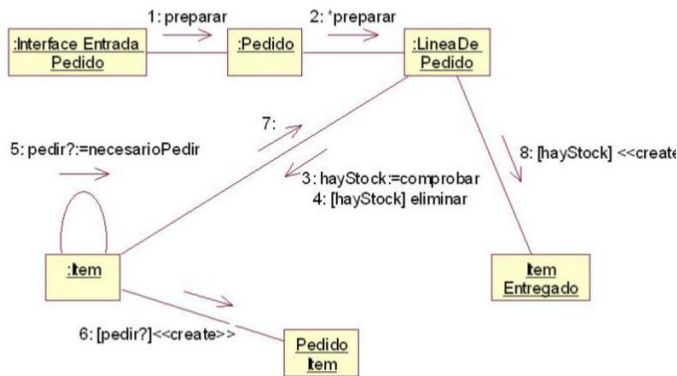
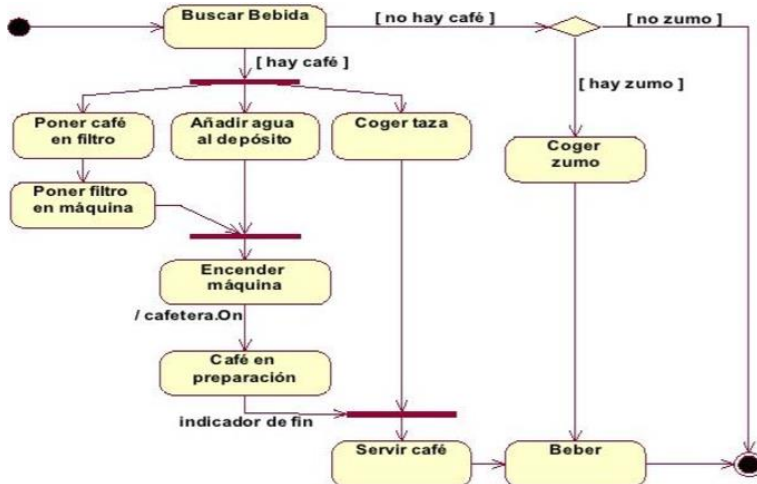


Diagrama de Colaboración para la función "ingresar un ítem".

Tanto los diagramas de colaboración como los de secuencia son proyecciones del mismo modelo de los aspectos dinámicos de un sistema, son útiles para ver el comportamiento de varios objetos en un caso de uso.

#### 5.2.4 Diagramas de actividades

Los diagramas de actividades se utilizan para modelar los aspectos dinámicos de un sistema y es una variación directa del diagrama de estados, pero enfocado a actividades y eventos que hacen cambiar una actividad.

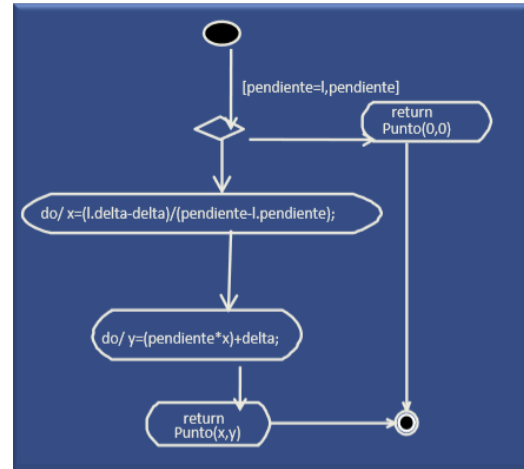


Elementos:



Sus dos usos más extendidos son:

- Modelar un flujo de trabajo, donde se hace hincapié en actividades tal y como las ven los actores.
- Modelar una operación:



#### 5.2.5 Diagramas de Estado

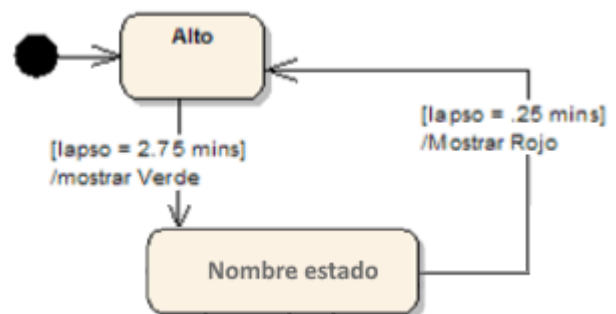
Eventos: Especificación en UML de un acontecimiento significativo a lo largo del tiempo. En el contexto de una máquina de estado modelan los estímulos. Pueden ser:

- **Síncronos:** llamadas o invocación de operaciones.
- **Asíncronos:** señales, paso del tiempo y el cambio de estado, representan suceso que pueden acaecer en cualquier instante.

Y de tipo:

- **Externos:** Entre el sistema y sus actores.
- **Internos:** Entre los objetos del sistema.

Para representarlos hay que tener en cuenta que cada objeto está en un estado en un cierto instante, que el estado va caracterizado por los valores de los atributos del objeto y que el estado en el que se encuentra el objeto caracteriza sus condiciones dinámicas.







Se pueden modelar 4 tipos de eventos: señales, eventos de llamada, eventos de tiempo y eventos de cambio.

### Señales

Son objetos enviados **asíncronamente** por un objeto y capturados por otro, el tipo más común de señales internas son las excepciones y se generan como resultado de una transición de un objeto.

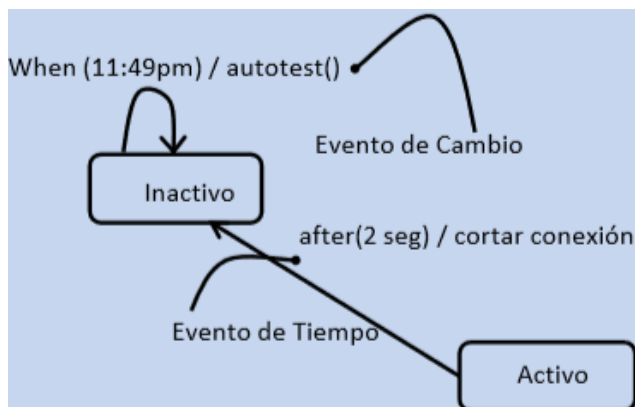
### Eventos de llamada

Representan la invocación de una operación de un objeto. Son eventos síncronos, cuando un objeto invoca una operación sobre otro objeto, el control pasa del emisor al receptor disparando la transición y acabando la transición, el receptor pasa a un nuevo estado y el control regresa al emisor.

### Eventos de tiempo y cambio

Un evento de tiempo representa el paso del tiempo, se modela con la palabra *after* seguida de una expresión, el tiempo se mide desde el instante en el que se entra en el estado.

Un evento de cambio representa un cambio de estado o cumplimiento de alguna condición, se modela con la palabra *when* seguida de una expresión booleana. Un ejemplo:



La **máquina de estados** especifica la secuencia de estados por la que pasa un objeto durante su vida, la evolución se produce a causa de eventos. También se pueden utilizar para modelar el comportamiento dinámico de otros elementos de modelado. La máquina de estados modela el comportamiento de un objeto individual. Además, se centra en el flujo de control entre estados.

**Estados:** Un estado es una condición o situación en la vida de un objeto durante la cual se

satisface una condición, se realiza alguna actividad o se espera algún evento. Son finitos y tienen varias partes:

- Nombre.
- Acciones de e/s.
- Transiciones internas.
- Subestados
- Eventos diferidos.

**Transiciones:** Son una relación entre dos estados que indica que un objeto que esté en el primer estado realizará ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y si se cumplen ciertas condiciones.

Puede contener los siguientes elementos:

- Evento: (definición de evento)
- Guarda: Son condiciones lógicas que retornan true/false. La transición solo ocurre si la guarda devuelve true.
- Acción: Consecuencia de una transición.

**Notación:** Evento [Guarda]/Acción

## 5.3 El modelo estructural en UML

Las partes estáticas del sistema se representan mediante:

- Diagramas de Clases
- Diagramas de Objetos
- Diagramas de Paquetes
- Diagramas de Componentes
- Diagramas de Despliegue

Los aspectos estáticos son aquellos que cubren la parte más estable del sistema software.

### 5.3.1 Diagramas de Clases

Son los más utilizados en el modelo orientado a objetos y nos muestran un conjunto de clases, interfaces y colaboraciones, se utilizan para modelar la vista estática de un sistema. Realiza una abstracción a través de:

- Clasificación/Instanciación
- Composición/Descomposición
- Agrupación/Individualización
- Especialización/Generalización

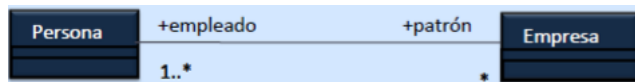
Se pueden considerar desde dos perspectivas, **conceptual** donde se presentan los conceptos del dominio estudiado y **especificación**, donde se representan los tipos que representan una interfaz. O **implementación**, en la que se representan las clases tal y como serán implementadas.





La **multiplicidad** define cuántas instancias de tipo A pueden asociarse a una instancia de tipo B, puede ser:

- \* (cero o más)
- 1..\* (uno o más)
- 1..40 (de uno a cuarenta)
- 5 (exactamente 5)
- 2,4,6 (exactamente 2,4,6)



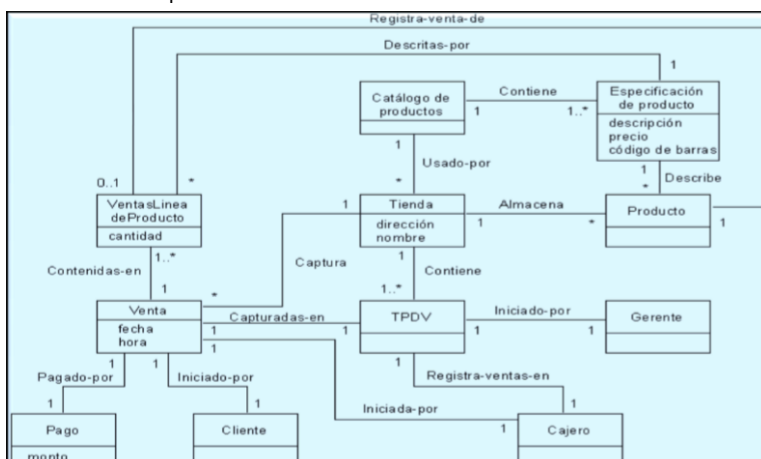
### Tipos de relaciones entre clases:

Columna 1	Columna 2	Columna 3
Agregación	Forma Parte de	
Composición	Forma Parte de	
Asociación	Conoce a ...	
Dependencia	Utiliza ...	
Generalización	Es un ...	
Realización	Implementa	

La diferencia entre agregación y composición es que esta última es una forma de agregación con fuerte pertenencia y un tiempo de vida coincidente entre las partes y el todo.

### Un diagrama de clases bien estructurado:

- Se centra en comunicar un aspecto de la vista del sistema
- Contiene sólo aquellos elementos que son esenciales para comprender ese aspecto
- Proporciona detalles consistentes con el nivel de abstracción
- Debe tener un nombre que comunique su propósito
- Hay que intentar no mostrar demasiados tipos de relaciones.

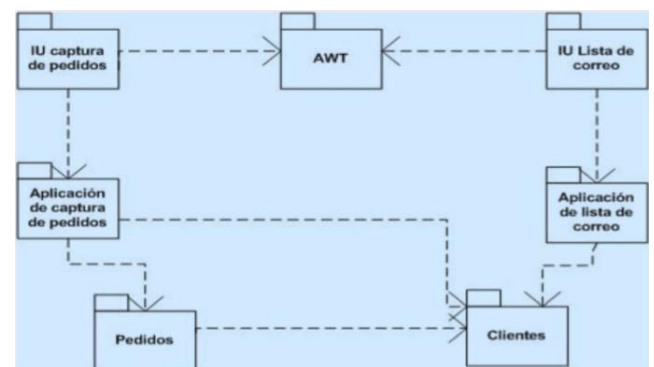


### 5.3.2 Diagramas de paquetes

Los paquetes se utilizan para organizar los elementos cercanos semánticamente del modelado en partes mayores que se pueden manipular como un grupo, los diagramas de paquetes sirven para obtener diferentes vistas de la arquitectura del sistema.

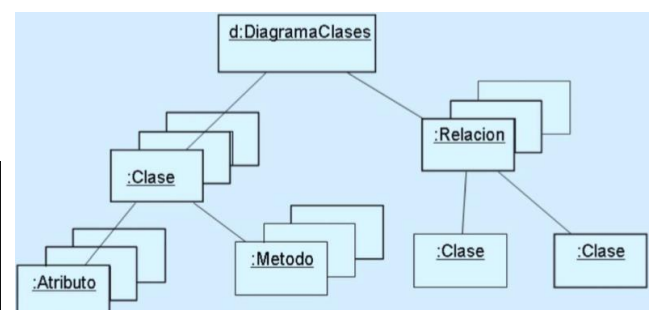
Un paquete puede contener otros elementos, incluyendo clases, interfaces, componentes, nodos, casos de uso, diagramas, etc. Cada elemento pertenece exclusivamente a un único paquete.

La distinción entre clase y paquete es que las clases son abstracciones de cosas encontradas en el sistema y los paquetes son los mecanismos que se emplean para organizar los elementos del modelo.



### 5.3.3 Diagramas de objetos

Su objetivo es modelar las instancias de los elementos contenidos en los diagramas de clases, modelan la vista de diseño y de procesos estática de un sistema. Representa un conjunto de objetos y sus relaciones en un momento concreto:



### 5.3.4 Diagramas de componentes

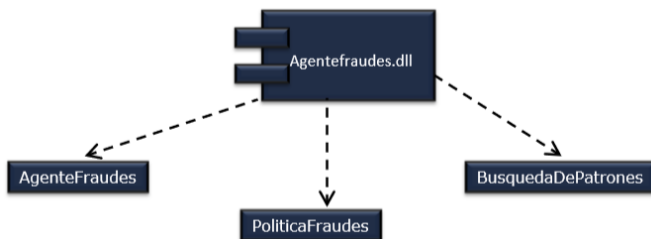
Los componentes se utilizan para modelar los elementos físicos que pueden hallarse en un nodo (ejecutables, bibliotecas, tablas, archivos...). Representa el empaquetamiento físico de elementos que por el contrario son



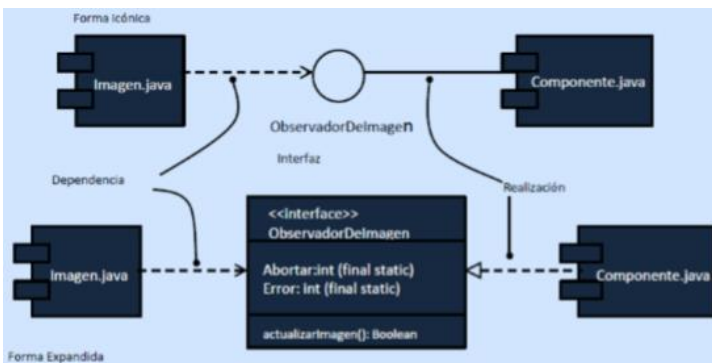
lógicos (clases, interfaces, colaboraciones...) Su representación gráfica sería:



La relación entre componentes y las clases que implementa puede representarse como una relación de dependencia, aunque la mayoría de las veces no es necesario.



**Interfaces:** Colección de operaciones que especifica un servicio de una clase o un componente, se puede representar mediante un icono o de forma expandida (mostrando sus operaciones). Por ejemplo:



Una interfaz que es realizada por un componente se denomina interfaz de exportación, cuando es usada por un componente se denomina de importación

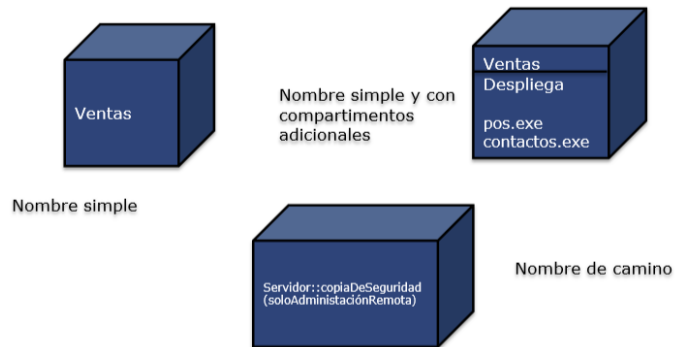
**Componentes:** Existen tres tipos:

- **Componentes de despliegue:** son necesarios y suficientes para formar un sistema ejecutable.
- **Componentes producto del trabajo:** son productos finales del proceso de desarrollo (código fuente, etc)

- **Componentes de ejecución:** se crean como consecuencia de un sistema en ejecución.

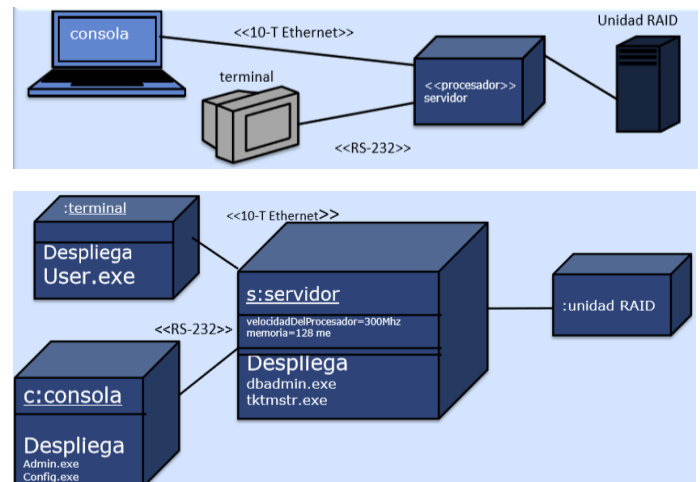
### 5.3.5 Diagramas de despliegue

Se utiliza para modelar la topología del hardware sobre el que se ejecuta el sistema. Los nodos representan un recurso computacional y son elementos físicos en tiempo de ejecución:



La mayoría de las veces los nodos se utilizan para el modelado de procesadores y los dispositivos que conforman los sistemas.

Un procesador es un nodo con capacidad de procesamiento. Un dispositivo es un nodo sin capacidad de procesamiento y representa algo que interrelaciona con el mundo real. Por ello hay que estereotipar como procesadores o dispositivos pudiendo mostrar un icono distinto para cada uno:



## 5.4 Técnicas estructuradas

A la hora de organizar estas técnicas, consideramos dos enfoques, ya sea de modelado o de la forma de representación:



clasificación según la forma de representación (tabla 1):

- **Gráficas:** Se utilizan técnicas que representan componentes de un aspecto particular del modelo. Se combinan con otros tipos de técnicas.
- **Textuales:** Especifican más en detalle las técnicas gráficas.
- **Marcos o plantillas:** Especifican la información de un componente.
- **Matriciales:** Son técnicas de comprobación de exactitud y compleción entre modelos.

Clasificación según el enfoque de modelado (tabla 2):

- **Dimensión de la Función:** Los DFD se utilizan para mostrar las funciones del sistema y sus interfaces. Se solapa con la dimensión de la información. Se apoya en el Diccionario de Datos y la Especificación de Procesos.
- **Dimensión de la Información:** Los diagramas Entidad-Relación se utilizan para describir las entidades existentes en el sistema y las relaciones entre las mismas.
- **Dimensión del Tiempo:** Las listas de eventos se utilizan para describir cualquier cosa que ocurra y sobre la que el sistema deba responder.

	Información	Función	Tiempo
Información	<ul style="list-style-type: none"> <li>• Especificación de los tipos de entidad</li> <li>• Especificación de los tipos de interrelación</li> <li>• Especificación de los tipos abstractos de datos TAD</li> </ul>		
Función	<ul style="list-style-type: none"> <li>• Diccionario de datos</li> <li>• Especificación de los procesos</li> <li>• Especificación de entidades Externas</li> </ul>		
Tiempo	<ul style="list-style-type: none"> <li>• Definición de funciones</li> <li>• Especificación de Eventos</li> </ul>		

Tabla 1

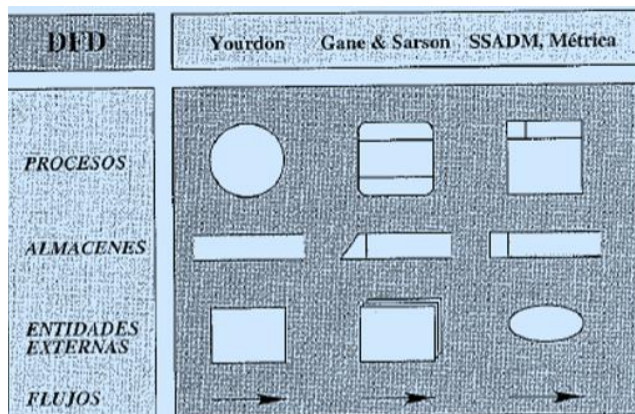
	Información	Función	Tiempo
Información	<ul style="list-style-type: none"> <li>• Diagrama Entidad Interrelación</li> <li>• Diagrama de estructura de datos</li> <li>• Matriz Entidad/Entidad</li> </ul>		
Función	<ul style="list-style-type: none"> <li>• Diagrama Flujo de datos</li> <li>• Matriz Entidad/Función</li> </ul>		
Tiempo	<ul style="list-style-type: none"> <li>• Diagrama Historia Vida de Entidad</li> <li>• Matriz Entidad/Evento</li> </ul>		

Tabla 2



### 5.4.1 Diagramas de Flujo de Datos

Es una representación en forma de red que refleja el flujo de la información y las transformaciones que se aplican sobre ella al moverse desde la entrada hasta la salida del sistema y se utiliza para modelar las funciones y los datos del sistema a distintos niveles de abstracción. Sus componentes son **procesos**, **almacenes**, **entidades externas** y **flujo**. Su representación gráfica depende de la metodología utilizada:



#### Procesos

Representan un componente que transforma los flujos de datos de entrada en uno o varios flujos de salida, son funciones que tiene que realizar el sistema.

El proceso debe ser capaz de generar los flujos de datos de salida a partir de los de entrada, se conoce como "la regla de conservación de datos". Cuando no le llegan todos los datos necesarios para obtener los datos de salida decimos que hay un "error de conservación de datos".

Deben ir numerados y nominados, cuando un flujo de datos muere dentro de un proceso, se pierde información.

#### Almacenes de datos

Representan la información del sistema almacenada de forma temporal, representando datos que se encuentran "en reposo". Si en un DFD hay un almacén que sólo tiene conexión con un proceso, se dice que el almacén es local y por tanto no debe aparecer a ese nivel.

Cuando es de tipo registro se dice que tiene estructura simple. El contenido siempre debe definirse.

#### Entidades externas

Representan un generador o consumidor de información del sistema que no pertenece al mismo. Los flujos que fluyen en ellas o los que llegan a ellas definen la interfaz entre el sistema y el mundo exterior.

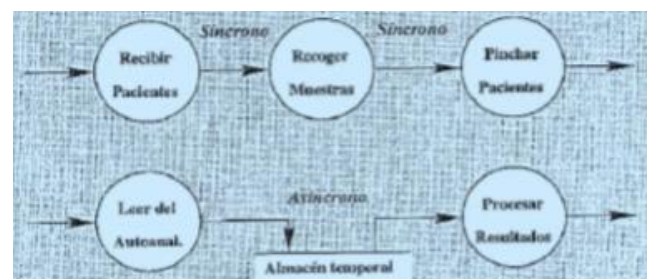
Normalmente sólo aparecen en el DFD de mayor nivel denominado Diagrama de Contexto, aunque se puede incluir en otros diagramas de nivel inferior.

#### Flujo de datos

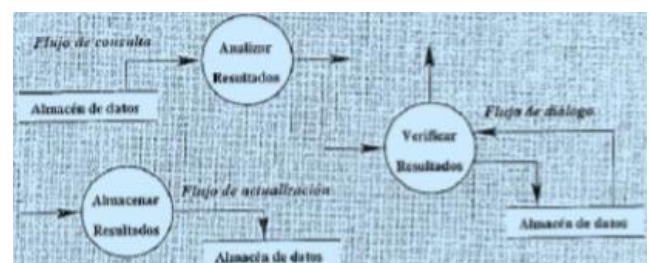
Se define como "un camino a través del cual viajan los datos de composición conocida de una parte del sistema a otra". Se representan como arcos dirigidos y según su persistencia en el tiempo pueden ser discretos o continuos. Las conexiones permitidas son:

Destino/Fuente	Proceso	Almacén	Entidad Externa
Proceso	SI	SI	SI
Almacén	SI	NO	NO*
Entidad Externa	SI	NO*	NO

La conexión directa entre dos procesos mediante un flujo de datos es posible siempre y cuando la información sea síncrona, es decir, que el proceso destino comience en el momento en el que el proceso origen finaliza su función:



Las diferentes conexiones que pueden aparecer entre los procesos y los almacenes son:



- **Flujo de Consulta:** muestra la utilización de la información del almacén por el proceso.
- **Flujo de Actualización:** Indica que el proceso va a alterar la información mantenida en el almacén.





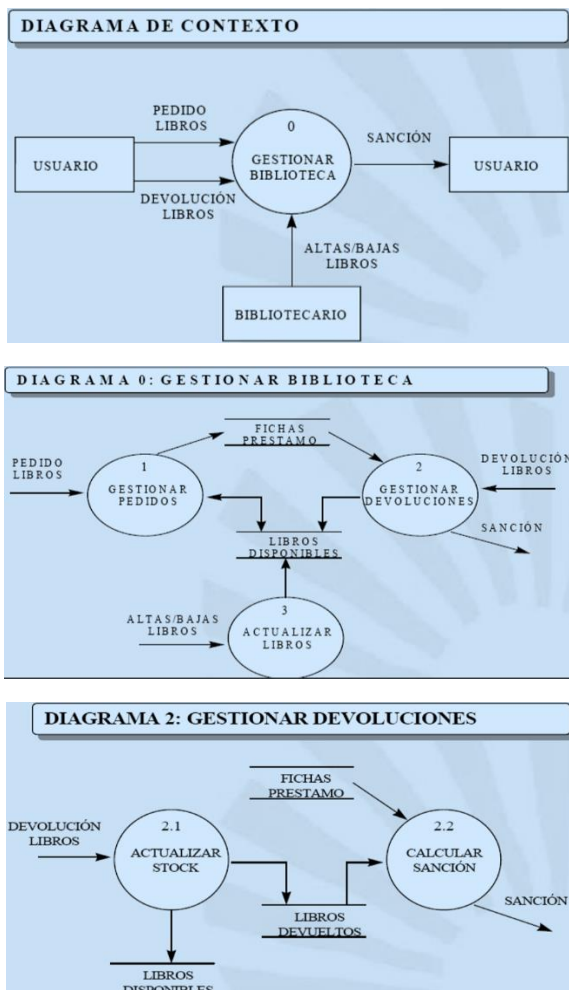
- **Flujo de Dialogo:** Representa como mínimo un flujo de consulta y uno de actualización que no tienen relación directa.

### Niveles de abstracción en los DFD

- **Diagramas de Contexto:** (nivel 0), delimita la frontera del sistema con el mundo exterior y define sus interfaces.
- **Diagrama de subsistemas:** Representan las funciones principales que debe realizar el sistema y sus relaciones
- **Procesos primitivos:** Para cada función primitiva habrá una especificación que la describa

Es necesario comprobar que la información que entra y sale de un proceso representado en un DFD de nivel N, sea consistente con la información que entra y sale del DFD.

A la hora de numerar los DFD cada diagrama recibe el número y nombre del proceso que se descompone, el diagrama de contexto se numera como 0, los diagramas de nivel 1 se numeran desde el 1 hasta el N y el resto se numeran según la numeración Dewey. Ejemplo:



### 5.4.2 Diccionario de Datos

Es una lista organizada de los datos utilizados por el sistema y que gráficamente se encuentran presentes en los flujos de datos y en los almacenes de datos del conjunto de DFD.

Contiene las definiciones de todos los elementos de los diagramas y su implementación puede ser mediante manuales, procesadores de textos, bases de datos, etc.

El DD contiene dos tipos de descripciones para los flujos de datos en el sistema:

- **Elemento dato,** es el nivel más importante, siendo el bloque básico e indivisible para todos los demás datos del sistema
- **Estructuras de datos,** es el grupo de datos elementales relacionados. Los flujos de datos y los almacenes son estructuras de datos.

### 5.4.3 Diagramas de Estructura

En el diseño estructurado, el diseño lógico de los requisitos del nuevo sistema de información se convierte en el diagrama de estructura. Al realizar este paso, se hace un análisis de las transacciones y las transformaciones.

Tienen los siguientes elementos:

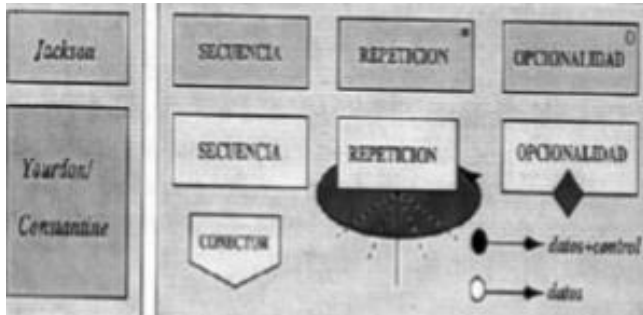
- **Módulo:** unidad de diseño que representa una división de software clara y manejable con sus interfaces definidas. Puede representar un programa, subprograma o rutina.
- **Módulo predefinido:** Módulo disponible en la biblioteca del sistema
- **Conexión:** Representa una llamada de un módulo a otro.
- **Parámetro:** Información que se intercambian los módulos: de datos y de control.
- **Almacén de datos:** lugar donde están almacenados los datos
- **Dispositivo físico:** Representación de cualquier dispositivo por el que se envíe o reciba información que necesite el sistema.

Las llamadas a módulos pueden ser:

- **Secuencial:** Un módulo llama a otro sólo una vez y se ejecuta de izquierda a derecha y de arriba abajo.



- **Repetitiva:** cada uno de los módulos inferiores se ejecuta varias veces mientras se cumpla una condición.
- **Alternativa:** El módulo superior llama, en función de una condición, a uno de los módulos inferiores.



### Diagramas de Jackson

Su objetivo es permitir representar las construcciones lógicas de secuencialidad, iteración y opcionalidad a nivel arquitectónico y estructural del sistema. Características:

- Cada módulo, función o proceso se representan mediante una estructura jerárquica mediante un rectángulo etiquetado
- La jerarquía es representada mediante arcos sin dirección
- El diagrama se interpreta de arriba hacia abajo y de izquierda a derecha
- Las construcciones son representadas, las iteraciones con "\*" y las selecciones con "o"
- En un mismo nivel sólo se representan módulos del mismo tipo.
- Un módulo sólo puede tener un módulo de repetición subordinado

### Diagramas de Yourdon

Son diagramas de Descomposición funcional tipo II que utilizan todos los mecanismos anteriormente especificados y son los que complementan a los DFD, se aplican en la estrategia de diseño y están basados en la estructura inicial del DFD

#### 5.4.4 Técnicas matriciales

Nacen de la necesidad de verificar las distintas actividades desarrolladas, el objetivo es ayudar a la verificación y validación entre el mundo de los datos y los procesos. Ejemplo:

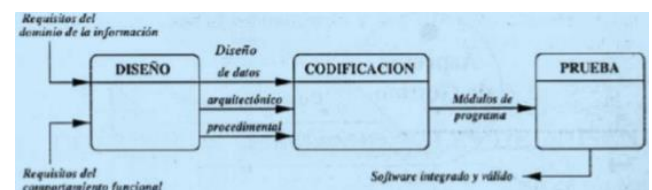
## Tema 6: Introducción al diseño

### 6.1 Principios del diseño

El proceso de diseño consiste en aplicar distintas técnicas y principios con el propósito de definir un dispositivo, un proceso o un sistema con suficiente detalle como para permitir su realización física. Las actividades del diseño consisten en:

- Diseño de datos
- Diseño arquitectónico
- Diseño de Interfaz
- Diseño de las funciones

Proceso de diseño:



Desde el punto de vista de la gestión:

- Diseño Preliminar: se refiere a la transformación de los requisitos en datos y arquitectura del software.
- Diseño detallado: se enfoca hacia los refinamientos de la representación arquitectónica que conduce a una estructura de datos detallada y a representaciones algorítmicas del software.

### 6.2 Conceptos fundamentales del diseño

- Abstracción
- Refinamiento
- Modularidad
- Jerarquía de control
- Procedimientos software
- Ocultación de la información
- Concurrencia
- Verificación
- Estética

#### Abstracción

Permite trabajar con los conceptos independientemente de las instancias particulares de éstos, permitiendo la separación de los aspectos conceptuales de los procedimientos. Los tipos a considerar son: funcionales, de datos y de control.





### Refinamiento

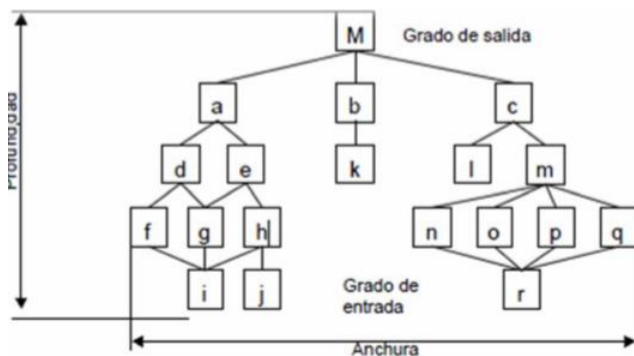
Es un proceso de elaboración que permite ampliar una declaración original, dando cada vez más detalles.

### Modularidad

Atributo del software que permite que sea intelectualmente manejable al estar dividido en componentes que se integran para satisfacer los requisitos.

### Jerarquía de control

La estructura del programa representa la organización de los componentes e implica una jerarquía de control. Representa dos características de la arquitectura del programa: la visibilidad y la conectividad.



La profundidad es el nº de niveles de control, la anchura es la amplitud global de control, el grado de salida es el nº de módulos controlados por otros módulos, el grado de entrada es el nº de módulos que controlan directamente un módulo dado.

### Procedimientos software

Se centra en los detalles de procesamiento de cada módulo individualmente.

### Ocultación de la información

Capacidad de un componente para que la información contenida dentro del mismo sea inaccesible a otros componentes que no la necesitan.

Se puede conseguir una modularidad eficaz definiendo un conjunto de módulos independientes que se comunican entre ellos sólo la información necesaria para conseguir la función del software.

### Concurrencia

Los sistemas software pueden ser clasificados en secuenciales y concurrentes. En unos sólo una porción está activa. En otros los procesos son independientes y activados de forma simultánea.

En los sistemas concurrentes existen problemas específicos como:

- Condición de bloqueo: Condición indeseable que ocurre cuando todos los procesadores de un sistema se quedan esperando a otros
- Exclusión mutua: Necesaria para evitar la actualización de los mismos componentes de un sistema compartido.
- Sincronización: Requerida para que los procesos concurrentes puedan comunicarse en los puntos adecuados.

### Verificación

Un diseño es verificable si puede demostrarse que el diseño generará el producto que satisface los requisitos del cliente. Y se desarrolla generalmente en dos pasos:

1. Verificando que la definición de los requisitos de programación satisface las necesidades del usuario
2. Verificando que el diseño satisface la definición de los requisitos.

### Estética

La simplicidad, elegancia y claridad de un propósito distinguen a los productos de alta calidad de los mediocres, un producto estéticamente agradable es fácilmente reconocido.

## **6.3 Diseño modular**

Un diseño modular reduce la complejidad y da una vista más fácil de implementación en el desarrollo en paralelo del sistema. La abstracción y la ocultación de la información se usan para definir módulos dentro de una arquitectura del software. Tipos de módulos:

- **Secuenciales:** Se referencia y ejecuta sin interrupción aparente. Son los que se presentan más frecuentemente



- **Incremental:** Puede ser interrumpido antes de la terminación por software de la aplicación. Tales módulos son útiles en sistemas conducidos por interruptores.
- **Paralelo:** Se ejecuta simultáneamente con otro módulo en entornos de multiprocesadores paralelos. Se encuentran en cálculos de alta velocidad.

Los módulos contienen instrucciones, lógica de los procesos y estructuras de datos, pueden ser compilados independientemente y pueden quedar incluidos dentro del programa. Los módulos pueden usar otros módulos.

### Independencia funcional

Se consigue desarrollando módulos con una función única y una disminución de la interacción entre los módulos. Se mide usando dos criterios cualitativos: Cohesión y acoplamiento.

### **Cohesión**

Es una medida de la fuerza funcional estática de un módulo. Es una extensión del concepto de ocultación de la información. El objetivo de la cohesión es diseñar servicios robustos y altamente cohesionados cuyos elementos estén fuerte y genuinamente relacionados entre sí. La cohesión tiene varios niveles:

	Niveles de cohesión	Niveles de acoplamiento
Bajo	Coincidental	<b>Sin acoplamiento directo</b>
	Lógica	De datos
	Temporal	Por estampado
	Procedimental	De control
	Comunicativa	Externo
	Secuencial	Común
Alto	<b>Funcional</b>	<b>Del contenido</b>

- **Coincidental:** un módulo ejecuta un conjunto de tareas que están relacionadas entre sí.
- **Lógica:** Cuando un módulo contiene actividades de la misma categoría.
- **Temporal:** Las tareas están relacionadas por producirse a un mismo tiempo.
- **Procedimental:** Los elementos de procesamiento de un módulo están relacionadas y deben ejecutarse en un orden específico.
- **Comunicación:** Cuando todos los elementos de procesamiento se concentran en un área sobre una estructura de datos.

- **Secuencial:** Ocurre cuando la salida de un elemento es la entrada del siguiente.
- **Funcional:** Realiza una única tarea.

### **Acoplamiento**

Es una medida de la independencia relativa entre los módulos, depende de la complejidad de la interfaz. Un bajo acoplamiento indicará que un cambio en un componente no supone un cambio en otro.



El objetivo es conseguir el acoplamiento más bajo posible, que nos indica que el sistema está bien dividido. Se consigue mediante la eliminación o reducción de relaciones innecesarias. Tipos de acoplamiento:

- Normal
- De datos
- Por estampado
- De control
- Externo
- Común
- De contenido

Mejor  
Bajo  
Débil  
Alto  
Peor

## **Tema 7: Introducción a las pruebas del software**

Las pruebas son una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto.

Las pruebas deben centrarse en dos objetivos: probar si el software no hace lo que debe hacer y si el software provoca efectos secundarios. Se debe planificar y diseñar las pruebas de manera sistemática para poder detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo.



## 7.1 Proceso de pruebas (imagen)



En el proceso de pruebas se realizan una serie de etapas que son:

1. Generación del plan de pruebas en base a la documentación.
2. Diseño de las pruebas específicas.
3. Se debe considerar la configuración software usada.
4. Comparar la salida de la prueba con la salida esperada, posteriormente depurar y analizar las estadísticas de errores.

Si no se detectan defectos en el software al ejecutar las pruebas podemos tener cierto nivel de confianza en que el programa no tiene defectos. La dificultad estará en saber elegir los casos de prueba adecuados en cada situación.

## 7.2 Técnicas de diseño de casos de prueba

Pueden ser:

- Enfoque estructural o de caja blanca
- Enfoque funcional o de caja negra
- Enfoque aleatorio

### 7.2.1 Pruebas estructurales o de caja blanca

Están basados en la elección de caminos que nos den la seguridad de que se descubren defectos en el software. Myers propone una clasificación para los criterios de cobertura lógica, que son:

- Cobertura de sentencias
- Cobertura de decisiones
- Cobertura de condiciones

- Criterio de decisión/condición
- Criterio de condición múltiple
- Criterio de cobertura de caminos.

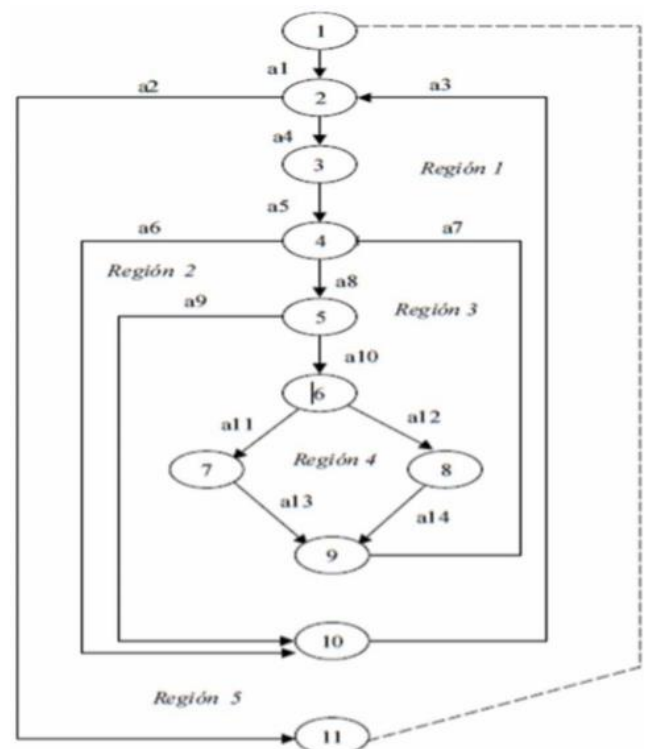
La cobertura de caminos es el criterio más importante a considerar en este tipo de pruebas. Como el número de posibilidades es muy grande se utiliza la prueba de caja blanca, que es un indicador de caminos independientes, que viene dado por la Complejidad Ciclomática  $V(G)$ .

La Complejidad Ciclomática aporta el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada secuencia al menos una vez. Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. En términos del grafo de flujo, un camino independiente se debe de mover por lo menos por una arista que no haya sido recorrida anteriormente.

La complejidad se puede calcular de tres formas:

1.  $V(G) = a - n + 2$ , siendo  $a$  el número de arcos o aristas del grafo y  $n$  el número de nodos.
2.  $V(G) = r$ , siendo  $r$  el número de regiones cerradas del grafo.
3.  $V(G) = c + 1$ , siendo  $k$  el número de nodos de condición.

Un **nodo predicado** es aquel que contiene una condición y se caracteriza porque del emergen dos o más aristas.





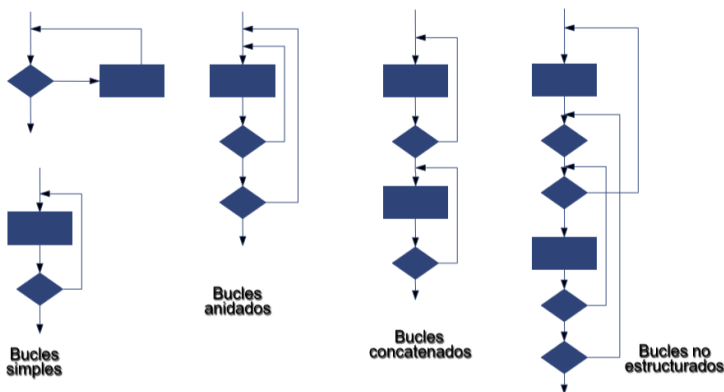
1.  $V(G) = 14 - 11 + 2 = 5$
2.  $V(G) = 5$  regiones cerradas
3.  $V(G) = 5$  condiciones

Es posible mecanizar la determinación del conjunto básico de caminos. Para ello se utilizará una **matriz de conexiones**, consistente en una matriz cuadrada igual al número de nodos y cuyas entradas corresponden a las aristas, es decir, las conexiones entre los nodos.

### Pruebas de condiciones

Tipos de errores que pueden aparecer en una condición: error con un operador lógico, con un paréntesis lógico, con un operador relacional y con una expresión aritmética.

### Pruebas de bucles



Es una técnica de prueba estructural que se centra exclusivamente en la validez de las construcciones de bucles. Consideramos cuatro diferentes construcciones de bucle (arriba).

## 7.2.2 Pruebas funcionales o de caja negra.

Se centra en el estudio de la especificación del software, análisis de las funciones que debe realizar, de las entradas y de las salidas. Las pruebas a realizar son:

1. Ejecutar el máximo número de posibilidades de entrada diferentes para así reducir el total de casos
2. El que cubre un conjunto extenso de otros posibles casos

Estas pruebas intentan descubrir defectos diferentes a los de la caja blanca, por lo que se consideran métodos complementarios y no alternativos. Las categorías de errores que se intentan descubrir son: funciones incorrectas, errores de interfaz, errores en estructuras de datos, de rendimiento y de inicialización.

## 7.2.3 Prueba de partición o clases de equivalencia

Las cualidades que debe cumplir un buen caso de prueba de partición son:

- Cada caso debe cubrir el número máximo de entradas
- El método de diseño consistirá en identificar las clases de equivalencia y crear los casos de prueba correspondientes.
- Para identificar los casos de prueba se debe de identificar las condiciones de las entradas de programa e identificar las clases de equivalencia, ya sea de datos válidos o no válidos.

Existen algunas reglas que ayudan a identificar las clases de equivalencia:

- **Regla 1:** Si se especifica un rango de valores para los datos de entrada. Por ejemplo el número estará comprendido entre 5 y 25, se creará una clase válida ( $5 \leq \text{número} \leq 25$ ) y dos clases no válidas ( $\text{número} < 5$ ) y ( $\text{número} > 25$ ).
- **Regla 2:** Si se especifica un número de valores (por ejemplo, se pueden registrar de uno a cuatro titulares en una cuenta bancaria, se creará una clase válida ( $1 \leq \text{titulares} \leq 4$ ) y dos no válidas ( $\text{titulares} < 1$ ) y ( $\text{titulares} > 4$ ).
- **Regla 3:** Si se especifica una situación del tipo «debe ser» o booleana (por ejemplo, «el primer carácter debe ser una letra»), se identifican una clase válida («es una letra») y una no válida («no es una letra»)
- **Regla 4:** Si se especifica un conjunto de valores admitidos y se sabe que el programa trata de forma diferente cada uno de ellos, se identifica una clase válida por cada valor y una no válida (cualquier otro caso)
- **Regla 5:** En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores



Ejemplo:

Condición de entrada	Clases válidas	Clases inválidas
Código área Nº de 3 dígitos que no empieza con 0 ni 1)	(1) 200 ≤ código ≤ 999	(2) código < 200 (3) código > 999 (4) no es número
Nombre para identificar la operación	(5) seis caracteres	(6) menos de 6 caracteres (7) más de 6 caracteres
Orden  Una de las siguientes →	(8) «cheque» (9) «depósito» (10) «pago factura» (11) «retirada de fondos»	(12) ninguna orden válida