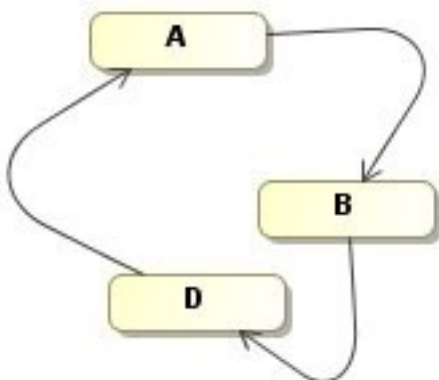
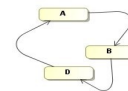


# Algunos temas sobre la ingeniería de software

Fases de diseño. Casos de uso. Diagramas UML. Patrones de diseño. Documentación





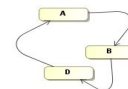


## Índice de contenidos

<b>Introducción.....</b>	<b>6</b>
Fases tradicionales del ciclo de vida.....	6
Ejecución del ciclo de vida.....	6
La aplicación de metodologías de desarrollo.....	9
<b>Fases generales del diseño.....</b>	<b>10</b>
Otras denominaciones.....	10
<b>Fase de: Planificación.....</b>	<b>11</b>
Definir el plan preliminar.....	11
Informe preliminar de investigación.....	11
Análisis de requerimientos.....	12
Glosario.....	13
Implementar un prototipo.....	14
Casos de uso de alto nivel y esenciales.....	14
Definir el modelo conceptual preliminar.....	17
Definir la arquitectura preliminar del sistema.....	24
Perfeccionar el plan.....	25
<b>Fase de: Construcción.....</b>	<b>26</b>
Perfeccionamiento del plan.....	26
Análisis funcional.....	27
Diseño orgánico.....	33
Implementación.....	45
Pruebas.....	46
<b>Apéndices.....</b>	<b>47</b>
Capas de un sistema ordinario de información orientado a objetos.....	47
Fichas CRC.....	49
Plantilla simplificada del diseño de software.....	50

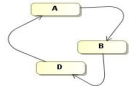
## Índice de tablas

Taula 1: Relación de ciclos de vida clásicos.....	9
Taula 2: Otras denominaciones comunes a las fases de diseño tradicionales.....	10
Taula 3: Plantilla para la descripción de funcionalidades.....	13
Taula 4: Ejemplo de glosario.....	14
Taula 5: Plantilla para casos de uso de alto nivel.....	14
Taula 6: Ejemplo de caso de uso esencial.....	16
Taula 7: Ejemplo de caso de uso real.....	17
Taula 8: Plantilla para la obtención de entidades.....	18
Taula 9: Plantilla para la asociación de entidades.....	21
Taula 10: Plantilla para la definición de contrato de operacioens.....	31



## Índice de ilustraciones

Dibuix 1: Situación de las metodologías en el proceso de diseño.....	9
Dibuix 2: Denominación: Greig Larman, (UML y patrones. Introducción al análisis y diseño OO).....	10
Dibuix 3: Tareas de la fase de planificación.....	11
Dibuix 4: Ejemplos de diagramas de casos de uso.....	15
Dibuix 5: Determinación del alcance del sistema en un caso de uso.....	16
Dibuix 6: Ejemplo de elección de entidades.....	19
Dibuix 7: Mas ejemplos de elección de entidades.....	19
Dibuix 8: Especificación de asociaciones en entidades.....	22
Dibuix 9: Representación de atributos en entidades. Caso de equiparación incorrecta con estructuras de BBDD.....	23
Dibuix 10: Representación de atributos en entidades. Caso de simplificación de entidades.....	23
Dibuix 11: Ejemplo de diagrama de entidades.....	24
Dibuix 12: Ejemplo de diagrama de arquitectura.....	24
Dibuix 13: Ejemplo de dependencia entre paquetes.....	25
Dibuix 14: Tareas de la fase de construcción.....	26
Dibuix 15: Subtareas de la tarea de análisis funcional.....	27
Dibuix 16: Ejemplo de transformación en clase asociación.....	28
Dibuix 17: Ejemplo de atributo derivado.....	29
Dibuix 18: Ejemplo de asociación calificada.....	29
Dibuix 19: Ejemplo de diagrama de secuencia.....	30
Dibuix 20: Subtareas de la tarea de diseño orgánico.....	33
Dibuix 21: Ejemplo de definición de interfaz gráfica.....	34
Dibuix 22: Ejemplo de polimorfismo.....	35
Dibuix 23: Ilustración del patrón "No hables con extraños".....	36
Dibuix 24: Ilustración del uso de gestores.....	37
Dibuix 25: Ejemplo de diagrama de colaboración.....	38
Dibuix 26: Ilustración del patrón experto.....	39
Dibuix 27: Ilustración del patrón Creador.....	39
Dibuix 28: Ilustración del patrón Bajo acoplamiento.....	40
Dibuix 29: Ilustración del patrón Comando.....	41
Dibuix 30: Presentación y Dominio conectados directamente.....	41
Dibuix 31: Presentación y Dominio conectados mediante gestores y subscripción a eventos.....	41
Dibuix 32: Ejemplos de diagramas de clases.....	42
Dibuix 33: Ejemplo de definición de interfaces.....	43
Dibuix 34: Ejemplo de diagrama ER.....	44
Dibuix 35: Subtareas de la tarea de implementación.....	45
Dibuix 36: Modelo de pruebas.....	46
Dibuix 37: Ilustración de arquitectura multicapa.....	47
Dibuix 38: Modelo de comunicaciones multicapa.....	47



---

## Introducción

El presente documento es una introducción a las principales técnicas en la gestión del ciclo de vida del desarrollo de Sistemas de Información, documentación y metodologías de desarrollo.

---

## Fases tradicionales del ciclo de vida

Tradicionalmente se distinguen las siguientes fases en el ciclo de vida en el desarrollo y mantenimiento de un Sistema de Información:

- Planificación
- Análisis
- Diseño
- Codificación
- Pruebas
- Implantación/Integración
- Mantenimiento

---

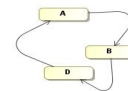
## Ejecución del ciclo de vida

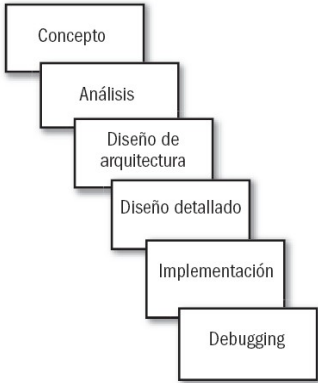
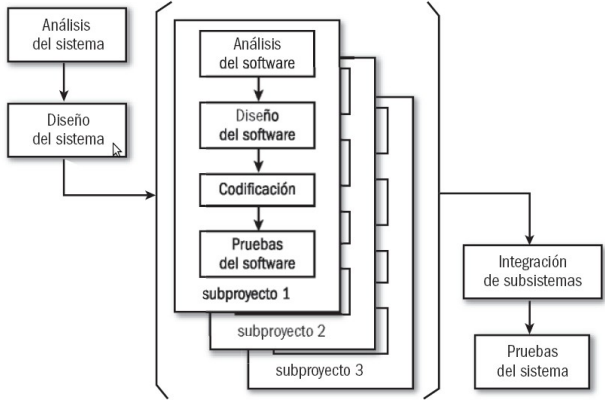
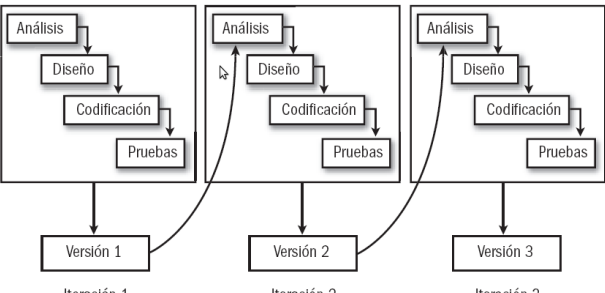
Cada fase del ciclo de vida puede ejecutarse de diversas formas, dependiendo entre otros:

- Del nivel de conocimiento de los requerimientos
- Del coste, (no sólo económico, sino por ejemplo temporal), de desarrollo del proyecto.
- De la intervención o no de prototipos

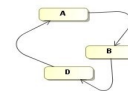
Así tenemos los siguientes ciclos de vida clásicos:

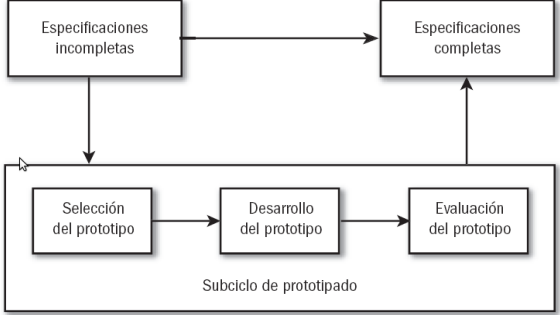
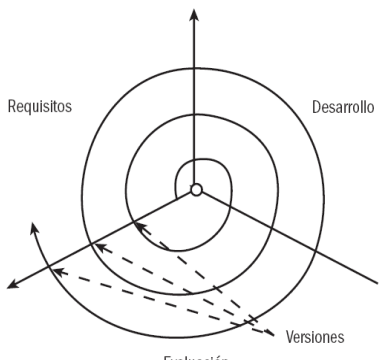
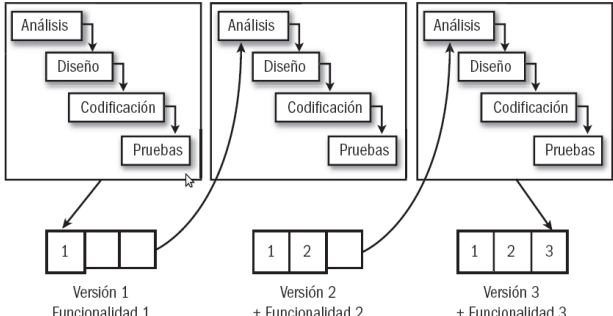
- Cascada
- Subproyectos
- Iterativo
- Prototipo
- Evolutivo
- Incremental
- Espiral

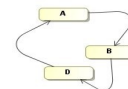


Ciclo de vida <sup>1</sup>	Descripción	Características
<p><b>Ciclo de vida en cascada</b></p>	<p>Las fases se ejecutan de forma encadenada i sólo una vez.</p> 	<p>Es poco ajustado a la realidad, (los requerimientos cambian a mitad de proceso).</p> <p>Es difícil determinar todos los requisitos al principio.</p> <p>El Sistema de Información sólo está disponible una vez terminado el proceso.</p>
<p><b>Ciclo de vida con subproyectos</b></p>	<p>Es una variación del ciclo de vida en cascada. En proyectos en que la funcionalidad puede subdividirse en partes operativas. Cada subproyecto se comporta como proyecto independiente que puede ser encargado a equipos de desarrollo independiente.</p> 	<p>Es una solución factible cuando se dispone de grandes recursos.</p> <p>Cada subproyecto hereda los problemas del desarrollo en cascada.</p>
<p><b>Ciclo de vida iterativo</b></p>	<p>Derivado también del ciclo de vida en cascada. La diferencia está en que el ciclo se repite en n iteraciones. Cada iteración aporta mejoras sobre el SI que el usuario puede validar. Un fracaso en una iteración no es tan grave como sería si se hubiera realizado todo el desarrollo en un sólo ciclo.</p> 	<p>En casos en que los usuarios necesitan partes funcionales aunque no estén completas, o bien no están claros todos los requerimientos. En contrapartida, el mantenimiento evolutivo se complica.</p>

1 Referència: Ciclos de vida del software, ([http://www.intep.edu.co/intep3/f\\_docente/66887/capitulo%20ciclo%20de%20vida%20de%20software.pdf](http://www.intep.edu.co/intep3/f_docente/66887/capitulo%20ciclo%20de%20vida%20de%20software.pdf))



Ciclo de vida	Descripción	Características
<p><b>Ciclo de vida prototipado</b></p>	<p>El ciclo de vida prototipado puede utilizarse en conjunción a cualquier ciclo de vida anterior. Añade al ciclo de vida el mecanismo de validar los requerimientos mediante el uso de prototipos. Un prototipo es una visión no operativa del Sistema de Información.</p> 	<p>Es una solución al problema “no te entiendo” y del “no se que quiero”.</p> <p>Plantea problemas respecto a la idea de producto terminado que puede tener el usuario.</p>
<p><b>Ciclo de vida evolutivo</b></p>	<p>En casos en que no se conocen gran parte de requerimientos esenciales. Se desarrolla una “versión” del SI que cubre aquello que se conoce. Posteriormente se dedarrollan versiones que evolucionan el SI.</p> 	<p>Plantea el problema de que nuevas versiones pueden interferir con funcionalidades actuales.</p>
<p><b>Ciclo de vida incremental</b></p>	<p>Consiste en realizar partes funcionales de un SI definido. Esto permite entregar al usuario módulos funcionales antes de haber acabado el proyecto completamente.</p> 	<p>Este concepto es ideal en escenarios en que el SI puede modularse en partes independientes.</p> <p>No es el concepto evolutivo, porque aquí si se conoce toda la funcionalidad. No es el concepto iterativo, porque aquí las iteraciones se corresponden con módulos.</p>



Ciclo de vida	Descripción	Características
<p><b>Ciclo de vida espiral</b></p>	<p>Este ciclo puede considerarse una conjunción de lo ciclos de vida prototipado e incremental. Se crea un prototipo que cubre la funcionalidad de ciertos módulos esenciales. El usuario valida el prototipo, se construye la solución, y se evoluciona el prototipo anterior para cubrir nueva funcionalidad.</p>	

Taula 1: Relación de ciclos de vida clásicos

## La aplicación de metodologías de desarrollo

La ingeniería de software se basa en la aplicación de metodologías de desarrollo que aseguren:

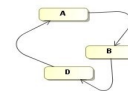
- La aplicación del ciclo de vida de software adecuado
- La reducción del alto coste y riesgo en el mantenimiento mediante la documentación y la reutilización
- Reduce la dependencia del individuo mediante la estandarización del desarrollo.
- Minimiza los incumplimientos en plazos de entrega mediante la planificación del proyecto.
- Facilita la estimación de nuevos proyectos a través de las métricas de proyectos anteriores.
- Asegura la calidad del software mediante la aplicación de métricas de calidad.



Dibuix 1: Situación de las metodologías en el proceso de diseño

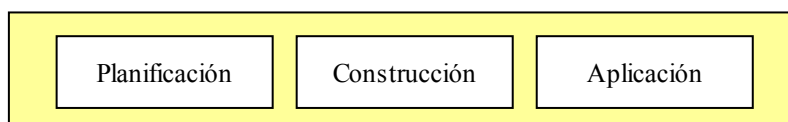
Podemos concebir la aplicación de metodologías como elemento integrador entre los elementos técnicos de generación y control de calidad de la solución de software, y la organización, que gestiona el proyecto y requiere de base documental para la reutilización, estimación y evaluación de la calidad.





## Fases generales del diseño

Cualquier proyecto de software consta de tres fases principales. Estas fases son ineludibles. Aunque puede haber mayor nivel de detalle en una o en otra dependiendo del proyecto.

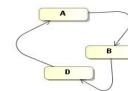


*Dibuix 2: Denominación: Greig Larman, (UML y patrones. Introducción al análisis y diseño OO)*

## Otras denominaciones

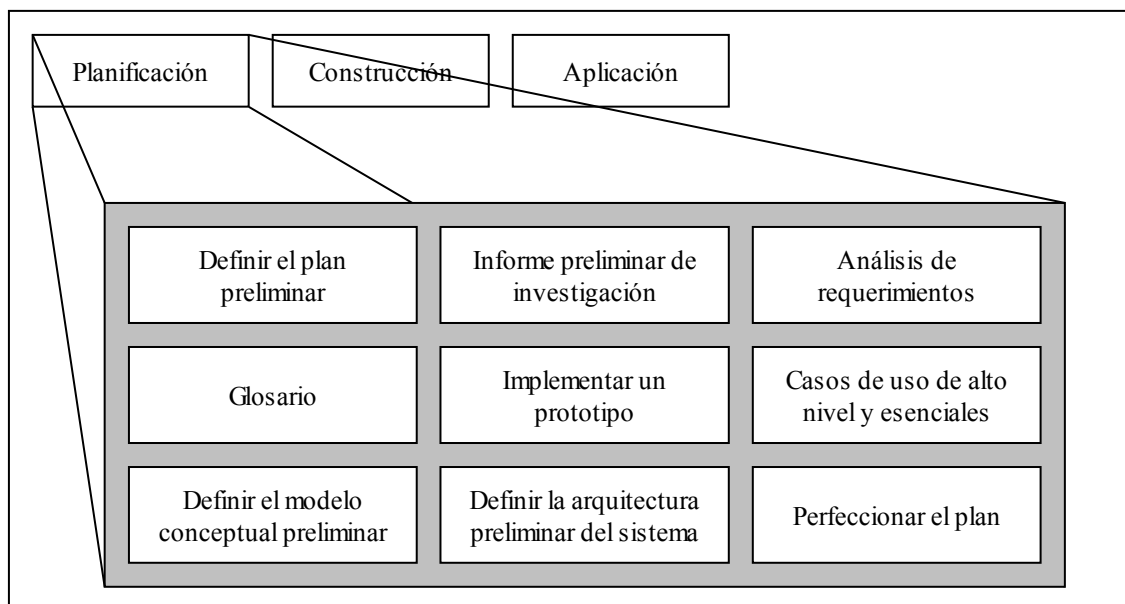
	Planificación	Construcción	Aplicación
<b>Métrica V3</b>	EVS, (Estudio de viabilidad)	ASI, (Análisis funcional del SI) DSI, (Diseño técnico del SI) CSI, (Construcción del SI)	IAS, (Implantación y aceptación del SI)
<b>Six Sigma</b>	Definir	Medir Analizar Consolidar	Implantar
<b>Extreme Programming, (XP)</b>	Exploración	Punto de fijación Planificación de la versión Planificación de la iterración Desarrollo	Desarrollo
<b>RUP</b>	Inception	Elaboration Construction	Transition

*Taula 2: Otras denominaciones comunes a las fases de diseño tradicionales*



## Fase de: Planificación

En esta fase se realizan una serie de tareas que se reflejan en el siguiente esquema:



Dibuix 3: Tareas de la fase de planificación

### Definir el plan preliminar.

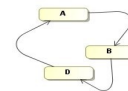
Definir la “idea” y recopilar toda la información disponible que nos permita establecer la idoneidad del proyecto.

- Describir la idea.
- Describir los recursos y el presupuesto.

### Informe preliminar de investigación.

Evaluación del mercado y alternativas. Explorar hasta que punto se resuelven las necesidades expresadas.

- Motivos y alternativas.
- Descripción de las necesidades de la empresa.



---

## Análisis de requerimientos.

El análisis de requerimientos consiste en una descripción de las necesidades o “deseos” de un producto. La finalidad es identificar y documentar lo que realmente se necesita. Definir las necesidades de la empresa **de forma inequívoca**.

El análisis de requerimientos es el documento mas importante y, a la vez, técnicamente mas informal de todos los que componen el desarrollo de software. Un documento de requerimientos ha de mantener un correcto equilibrio entre narración y diagramas. La proporción de diagramas y su tipología puede depender de las características a describir; pero un documento de requerimientos que sólo sea narrativo en la definición del problema y la solución, seguramente no será un documento correcto. Como diagramas suelen ser útiles: Diagramas de estados, casos de uso, diagramas de interacción, diseño de formularios, etc. Todo vale para que el cliente entienda que es lo que se va a hacer y que implicaciones tendrá en el trabajo cotidiano del usuario, y en la totalidad del sistema. Dichos diagramas no tienen porqué ser técnicamente perfectos, pero si han de ser ilustrativos, (han de aportar información).

## Fases del análisis de requerimientos.

El análisis de requerimientos puede dividirse en las siguientes fases:

### Panorama general.

Una presentación general del proyecto.

### Clientes.

Peticionario del proyecto.

### Metas.

Definir cual es el fin del proyecto. Que es lo que se pretende.

### Funciones del sistema.

Define los requerimientos del sistema. Determinar las labores que deberá acometer el nuevo aplicativo. Los requerimientos suelen agruparse por subsistemas. Por ello, puede ser conveniente definir los requerimientos junto a la definición de la arquitectura del sistema.

! Ver: [Definir la arquitectura preliminar del sistema](#)

Las funciones del sistema, (requerimientos), pueden catalogarse de dos formas:

– **Tipo**

El tipo cataloga la función principal del requerimiento. Pueden catalogarse, (por ejemplo), de la siguiente forma:

**Funcional:** Describe una función que debe proporcionar el sistema.

**Rendimiento:** Describe requerimientos relacionados con el rendimiento de datos, servicios o funcionalidades concretas.

**Seguridad:** Describe requerimientos de integridad de los datos y/o de seguridad de los datos, o del sistema.

**Implantación:** Requerimientos referentes a la instalación y puesta en marcha del sistema.

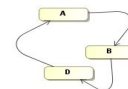
**Disponibilidad del sistema:** Requerimientos relacionados con la disponibilidad de la información o de funcionalidades concretas ante circunstancias especiales de funcionamiento.

– **Prioridad**

La prioridad cataloga los requerimientos por orden de importancia en su implementación. En este sentido pueden catalogarse, (por ejemplo), de la siguiente forma:

**Vital:** El requerimiento es crítico. Describe una funcionalidad necesaria.

**Importante:** El requerimiento es útil y necesario.



**Recomendable:** El requerimiento es útil, aunque puede omitirse.

**Opcional:** La no implementación del requerimiento no afecta las funcionalidades del sistema.

**Plantilla de descripción de funcionalidades:**

Ref #	Prioridad	Tipo	Caso de uso relacionado
#1	Vital	Funcional	Caso 1 caso 3
	Descripción de la funcionalidad		
#2	Recomendable	Funcional	Caso 2
	Descripción de la funcionalidad		

*Taula 3: Plantilla para la descripción de funcionalidades*

En esta plantilla las funciones pueden agruparse por subsistemas.

**Atributos del sistema.**

Todo lo relativo a la descripción de facilidad de uso del nuevo aplicativo. Pero nada referente a la funcionalidad.

**Plantilla de descripción de atributos:**

La plantilla de descripción de atributos debe contener:

- Tiempo de respuesta.
- Metáfora de interfaz.
- Tolerancia a fallos
- Plataforma del SO

El esquema de atributos suele representarse junto al de funcionalidades. Indicando, para cada funcionalidad, los atributos que le afectan.

**Requerimientos y equipos de enlace.**

Lista de las personas que deberían participar en la especificación de las funcionalidades y atributos del sistema. En la realización de entrevistas, pruebas, negociaciones y otras actividades.

**Grupos afectados.**

Aquellas personas o departamentos que reciben el impacto del desarrollo o aplicación del sistema.

**Suposiciones.**

Todo aquello relevante que se supone cierto.

**Riesgos.**

Cosas que puedan ocasionar fracaso o retraso del proyecto.

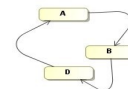
**Dependencias.**

Otras personas, sistemas y/o productos, de los cuales no puede prescindir el proyecto para su terminación.

---

**Glosario.**

El glosario es necesario para evitar malentendidos y uniformar los conocimientos del equipo. Su mantenimiento ha de ser constante durante todas las fases del desarrollo.



Ejemplo de glosario:

Concepto	Descripción	Ejemplos

Taula 4: Ejemplo de glosario

## Implementar un prototipo.

Opcional. Puede realizarse un prototipo. Este formará parte de la documentación y de la definición de la interficie de usuario. El usuario deberá validar el prototipo.

## Casos de uso de alto nivel y esenciales.

Es un documento narrativo que describe la secuencia de eventos de un actor, (usuario del sistema o agente externo), que utiliza el sistema para completar un proceso. No son los requerimientos, aunque los incluye tácitamente en las historias que narran. Dicho documento narrativo va acompañado de un diagrama, que describe gráficamente lo que en la narración se explica.

### Casos de uso de alto nivel.

Es el tipo mas utilizado en la fase de toma de requerimientos. Se basa en la siguiente plantilla:

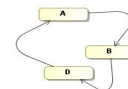
<b>Caso de uso:</b>	Nombre del caso de uso
<b>Actores:</b>	Lista de actores que intervienen en este caso de uso.
<b>Tipo:</b>	Primario, Secundario ...
<b>Descripción:</b>	.....

Taula 5: Plantilla para casos de uso de alto nivel

### Casos de uso expandidos.

Muestra mas información que el anterior. Y es mas utilizado en las fases de análisis funcional y de diseño. Se rige por la siguiente plantilla:

<b>Caso de uso:</b>	Nombre del caso de uso.
<b>Actores:</b>	Lista de actores que intervienen en este caso de uso.
<b>Propósito:</b>	Intención del caso de uso.
<b>Resumen:</b>	Equivale a “descripción” del caso de uso de alto nivel.
<b>Tipo:</b>	Primario / Secundario / Opcional ...
<b>Ámbito:</b>	Esencial / Real ...
<b>Referencias:</b>	Casos de uso relacionados con este.



**Curso normal de los eventos:**

Opcional

<b>Acción del actor:</b>	Respuesta del sistema.
<b>Acciones numeradas de los actores:</b>	Descripciones numeradas de las respuestas del sistema.

**Cursos alternos:**

Opcional

Para cada número de acción de un actor, o respuesta del sistema, explicar las alternativas y excepciones mas importantes.


**Contrato de operaciones:**

<b>Precondiciones:</b>	En que estado ha de estar el sistema para que el caso de uso sea factible.
<b>Postcondiciones:</b>	Como ha de quedar el sistema al finalizar el caso de uso

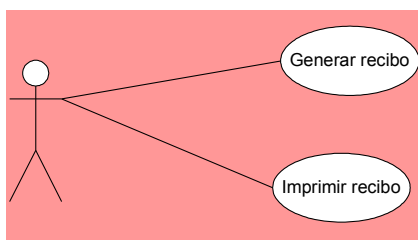
Los contratos de operaciones pueden incluirse en los casos de uso.

! Ver: [Definir contratos de operaciones.](#)

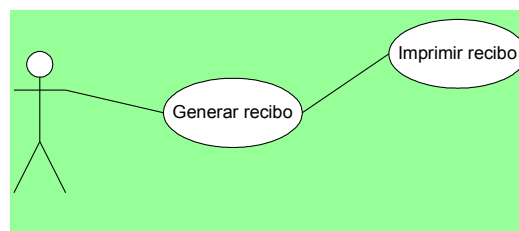
**Casos de uso: Actores.**

Representados por el símbolo: 

Representan entidades externas e internas del sistema, que participan en la historia del caso de uso. También puede ser la propia aplicación, o una máquina externa.

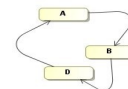


Caso de uso incorrecto en el caso de que “Imprimir recibo” sea una actividad que fuera dependiente de la generación.

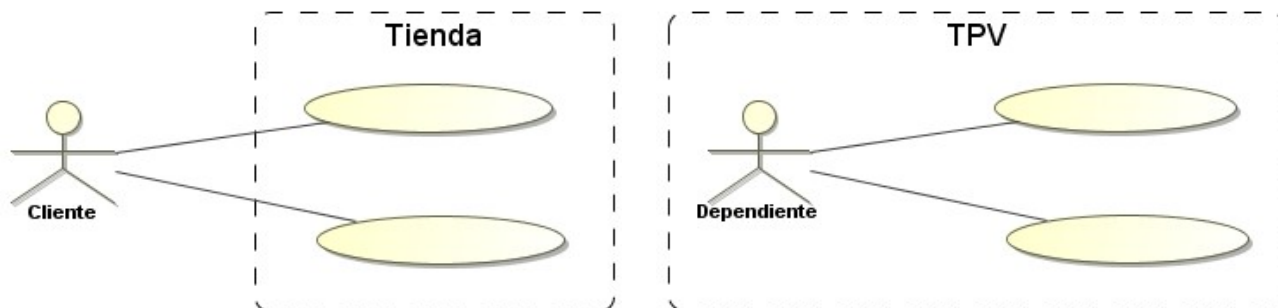


Caso de uso correcto

*Dibuj 4: Ejemplos de diagramas de casos de uso*



Las asociaciones de un caso de uso pueden estar englobados en un sistema :



Dibuix 5: Determinación del alcance del sistema en un caso de uso

Dependiendo del sistema, los casos de uso serían mas generales, (Tienda), o mas centrado en acciones, (TPV).

### Casos de uso: Tipos.

Se distinguen los siguientes tipos:

**Primario:** Procesos comunes mas importantes.

**Secundario:** Procesos menores o raros.

**Opcionales:** Procesos que podrían no definirse.

### Casos de uso: Ámbito.

Se distinguen los siguientes ámbitos:

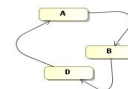
#### Esenciales

Casos de uso expandidos que se expresan de una forma teórica, y que contienen poca tecnología y pocos detalles de implementación. Las decisiones de diseño se posponen y se abstraen; especialmente las concernientes a la interfaz de usuario.

Por ejemplo:

Acciones de los actores	Respuestas del sistema
1 – El cliente se identifica.	1 – Se presentan las opciones

Taula 6: Ejemplo de caso de uso esencial



## Reales

Describe el proceso a partir de su diseño concreto actual. Al tratar de interfaces, a menudo ofrece presentaciones de pantallas. Es más propio del análisis funcional y del diseño.

Por ejemplo:

Acciones de los actores	Respuestas del sistema
1 – El cliente introduce su tarjeta.	1 – Pide el NIF
2 – Introduce el NIF	2 – Muestra menú de opciones

*Taula 7: Ejemplo de caso de uso real*

## Otros comentarios sobre los casos de uso.

Es importante que la primera acción descrita en un caso de uso sea, (o pueda transcribirse como):

**<Actor> inicia <Evento>**

Por ejemplo:

*Cliente llega a un TPV con productos que desea comprar*

Si en la descripción extendida del caso de uso hay una acción que permite x variantes; hay que realizar una descripción extendida para cada variante.

## Clasificación de los casos de uso.

Los casos de uso pueden clasificarse por:

- Tipo.
- Prioridad
- Subsistema

## Definir el modelo conceptual preliminar.

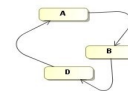
Esta subfase es aplazable; es decir, puede realizarse en otro momento del análisis en que se considere que se tiene más información.

Es el documento, junto a los casos de uso, más importante en la fase de análisis.

Se basa en el diagrama de entidades: Un diagrama de clases sin métodos, pero sí con atributos y asociaciones.

En la construcción del primer modelo conceptual, es mejor añadir entidades sobrantes, que descubrir en la fase de desarrollo entidades que no se definieron.





## Obtención de entidades.

Obtención de entidades a partir de una lista de categorías.

Antes de iniciar el diagrama de entidades, es conveniente crear una lista a partir de esta plantilla:

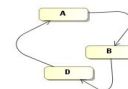
Categoría del concepto	Ejemplos
<b>Objetos físicos o tangibles</b>	TPV, Avión...
<b>Especificaciones, diseño o descripciones de cosas</b>	EspecificacionDeProducto, DescripcionVuelo ...
<b>Lugares</b>	Tienda, Aeropuerto...
<b>Transacciones</b>	Venta, Pago, Reserva ...
<b>Línea o renglón de elemento de transacciones</b>	VentasLineaProducto, LineaAlbaran ...
<b>Papel de las personas</b>	Cajero, Piloto ...
<b>Contenedores de otras cosas</b>	Tienda, Cesto, Avión...
<b>Cosas dentro de un contenedor</b>	Producto, Pasajero ...
<b>Otros sistemas de cómputo o electromecánicos externos al sistema</b>	SistemaAutorizacion, ControlTraficoAereo...
<b>Conceptos de nombres abstractos</b>	Hambre, Acrofobia ...
<b>Organizaciones</b>	DepartamentoDeVentas ...
<b>Eventos</b>	Venta, Robo, Junta, Vuelo, Accidente, Aterrizaje ...
<b>Procesos</b>	VentaUnProducto, ReservaAsiento ...
<b>Reglas y políticas</b>	PoliticaDeReembolso, PoliticaDeCancelaciones ...
<b>Catálogos</b>	CatalogoDeProductos ...
<b>Registros de finanzas, trabajo, contratos, etc.</b>	Recibo, ContratoEmpleo, Albaran ...
<b>Instrumentos y servicios financieros</b>	LineaDeCredito, Existencia ...
<b>Manuales, libros, etc.</b>	ManualDePersonal, ManualDeReparaciones ...

Taula 8: Plantilla para la obtención de entidades

Obtención de entidades a partir de frases nominales.

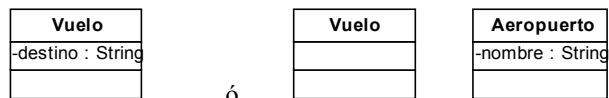
“Un **cliente** llega a una **caja de TPV** con **productos** que desea comprar”

Este método hay que utilizarlo con mucha prudencia, ya que la complejidad del lenguaje puede llevar a confusiones, (por ejemplo: Que el contexto oculte entidades implícitas. Que diversos elementos de la oración se refieran a la misma entidad, etc)



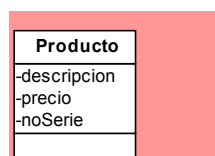
## Elección de entidades.

¿Entidad o atributo?

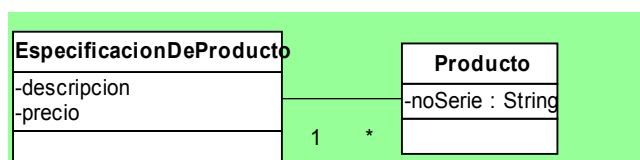


“destino” puede ser una cadena de texto, o algo mas complejo, (Aeropuerto). La elección de uno u otro dependerá de las necesidades. Pero en caso de duda es mejor convertir el atributo en una entidad independiente. Ya que en la fase de diseño es mucho mas sencillo “prescindir” de una entidad innecesaria, que “inventar” una entidad nueva.

Elección de entidades para evitar duplicidades.



Incorrecto en algunos escenarios



Correcto

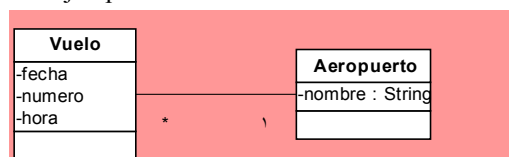
DibuiX 6: Ejemplo de elección de entidades

Un producto es un identificativo de su descripción. Con ello se evita que en los operativos de venta o de compra, estemos moviendo todos sus datos.

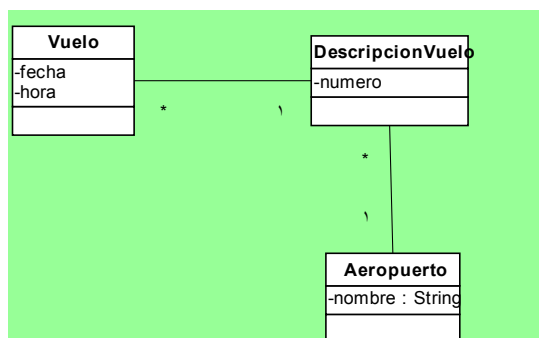
Este tipo de relaciones:

- Reduce información redundante o duplicada.
- La eliminación de un elemento, (Producto), no produce la perdida de información que ha de conservarse.

Otro ejemplo:

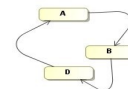


El “número de vuelo” puede conllevar la gestión de más información




Aquí, “número de vuelo” forma parte de una clas, (DescripcionVuelo), que puede encapsular más información

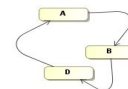
DibuiX 7: Mas ejemplos de elección de entidades



## Descarte de entidades.

Una vez escogidas todas las posibles entidades, es conveniente realizar un primer descarte. Se trata de encontrar aquellas entidades, que bien por duplicidad con otras, bien por falta de concreción, bien por otras causas, no son apropiadas en primera instancia para nuestro proyecto. Las causas posibles de descarte son:

- **Clases redundantes.**  
Clases que expresan la misma información que otras. Hay que dejar la que tenga el nombre mas informativo o adecuado al problema.
- **Clases irrelevantes.**  
Una clase que tenga que ver poco o nada con el problema, debe ser eliminada.
- **Clases vagas.**  
Una clase debe ser algo específico. Algunas entidades pueden no tener definidos sus límites de forma correcta. Otras pueden tener un ámbito excesivo. Estas entidades deben ser eliminadas o estudiadas con mayor profundidad.
- **Atributos.**  
Algunas entidades son en realidad atributos de otras entidades. Por ejemplo: DatosDeCuenta podría ser atributo de cuenta.  
 Véase: [Elección de entidades.](#)
- **Operaciones.**  
Si una entidad describe una operación que se aplica a objetos y que no es propiamente manipulada en si, entonces no es una entidad.
- **Roles.**  
El nombre de una entidad debería reflejar su naturaleza intrínseca, y no un rol o papel que desempeñe en una asociación. Por ejemplo: Si diversas clases llevan a cabo las mismas acciones, y se encuentran diferenciadas unicamente por el papel que ejercen en el sistema, probablemente puedan aglutinarse en una sola entidad. Por ejemplo: Vendedor y Dependiente.
- **Estructuras de implementación.**  
Las estructuras extrañas al mundo real deben ser eliminadas. Por ejemplo: Registro, Sistema, Línea de comunicaciones, etc. O bien substituidas por elementos reales, como por ejemplo: Cliente para Registro, y Teléfono o email para Línea de comunicaciones, etc.



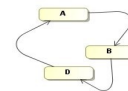
## Asociaciones entre entidades.

Categoría	Ejemplos
<b>A es una parte física de B</b>	Caja – TPV Ala – Avión
<b>A es una parte lógica de B</b>	VentasLineaProducto – Venta TramoDeVuelo – RutaDeVuelo
<b>A está físicamente contenido en B</b>	TPV – Tienda Producto – Estante Pasajero – Avión
<b>A está lógicamente contenido en B</b>	DescripcionDeProducto – Catalogo Vuelo – ProgramaVuelo
<b>A es una descripción de B</b>	DescripcionDeProducto – Producto DescripcionDeVuelo – Vuelo
<b>A es un elemento de línea de una transacción o reporte B</b>	VentasLineaProducto – Venta LineaAlbaran – Albaran
<b>A se conoce / introduce / registra / presenta / captura en B</b>	Venta – TPV Reserva – ListaPasajeros
<b>A es miembro de B</b>	Cajero – Tienda Piloto – Avión
<b>A es una subunidad organizacional de B</b>	Cajero – Tienda DtoContabilidad – Empresa
<b>A usa o dirige a B</b>	Cajero – TPV Piloto – Avión
<b>A se comunica con B</b>	Cliente – Cajero
<b>A se relaciona con una transacción B</b>	Pago – Venta
<b>A es una transacción relacionada con otra transacción B</b>	Pago – Venta Reserva – Cancelación
<b>A está contiguo a B</b>	TPV – TPV
<b>A es propiedad de B</b>	TPV – Tienda Avión – Línea aérea.

Taula 9: Plantilla para la asociación de entidades

### Asociaciones importantes.

- A es una parte física o lógica de B
- A está física o lógicamente contenido en B
- A está registrado en B

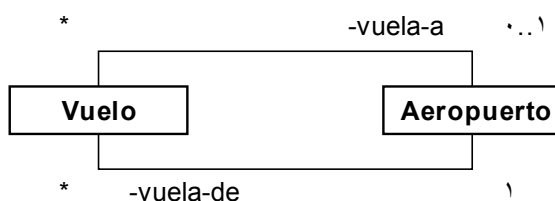


## Otros comentarios sobre entidades y asociaciones.

Es más importante descubrir los conceptos, (entidades), que las asociaciones. Es mejor no incluir las asociaciones redundantes ni las derivables.

Si es posible, las asociaciones deben tener nombre.

Si dos entidades están vinculadas por más de una asociación, estas han de dibujarse:



*Dibuj 8: Especificación de asociaciones en entidades*

Si durante la fase de desarrollo aparecen nuevas entidades y/o asociaciones, estas han de reflejarse en el diagrama de entidades del análisis funcional.

Las entidades y asociaciones eliminables del diagrama son aquellas que no aparecen reflejadas en el documento de requerimientos. El diagrama de entidades ha de ser sencillo y claro.

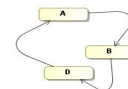
Enfatice las asociaciones que deban conocerse, pero incorpore también las opcionales que se requieran para comprender bien el dominio del problema que se plantea.

## Agregar atributos a las entidades.

El diagrama de entidades ha de reflejar los atributos, pero no métodos. Ni los atributos ni las entidades tienen por qué coincidir con los diagramas de clases de la fase de desarrollo.

Los atributos se extraen de los requerimientos y de los casos de uso, cualquier atributo no contenido en estos documentos está fuera de lugar.

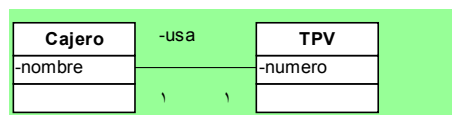
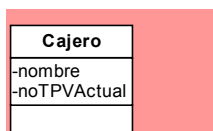
Pero pueden haber atributos no relacionados directamente con la clase en la que están. Por ejemplo: “fecha” puede ser atributo de la entidad “Venta”, porque de Venta se presupone la impresión de un recibo.



### Representación de los atributos.

- Los tipos primitivos se representan por un nombre en la caja que representa la entidad.
- Los tipos complejos se representan por asociaciones entre las entidades afectadas.

Los atributos no deben ser claves foráneas. Es decir: Nunca deben representar instancias concretas de otra entidad. Esto debe ser substituido por asociaciones. Por ejemplo:



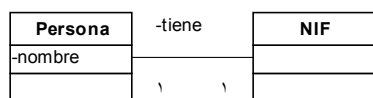
Correcto

El atributo “numeroTVPActual” está representando una instancia de TPV concreta. Se está reflejando una estructura de BBDD que no compete en este ámbito.

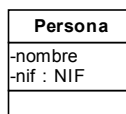
*Dibuix 9: Representación de atributos en entidades. Caso de equiparación incorrecta con estructuras de BBDD*

### Mostrar entidades desde atributos.

Para entidades secundarias o no representativas, podrían mostrarse en el diagrama de entidades como atributos de tipo complejo; en aquellas entidades que hagan uso.

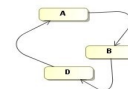


Complejo

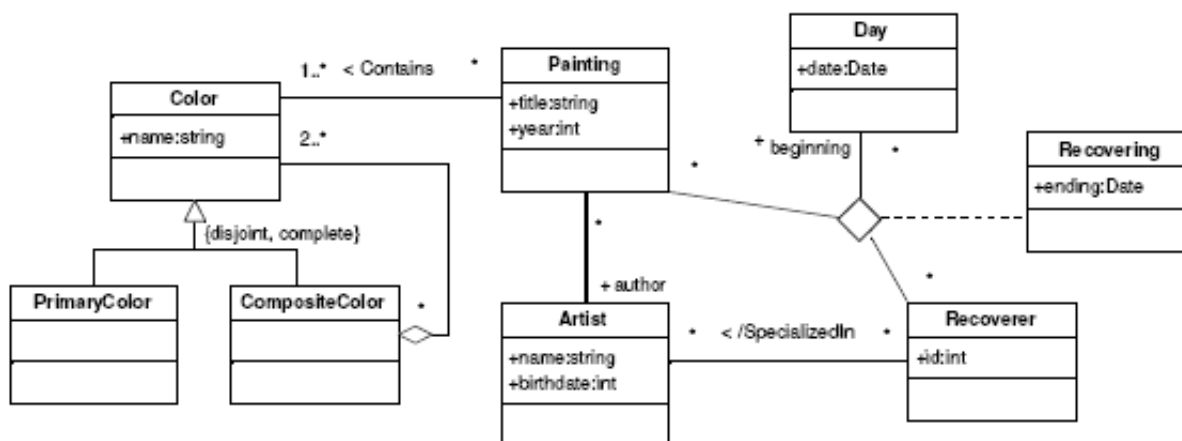


Simplificado

*Dibuix 10: Representación de atributos en entidades. Caso de simplificación de entidades*



## Ejemplo de diagrama de entidades



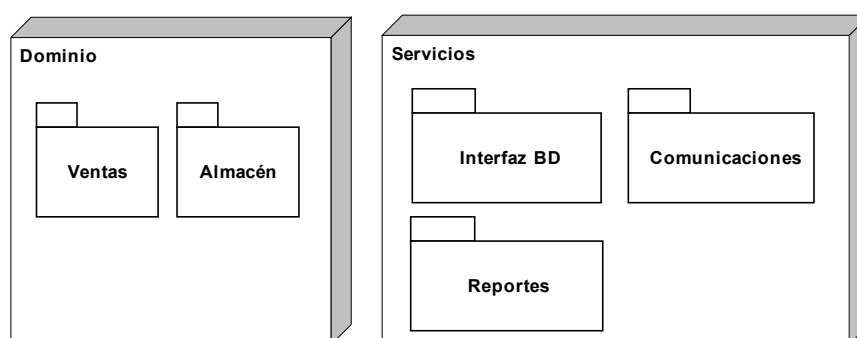
Dibuix 11: Ejemplo de diagrama de entidades

## Definir la arquitectura preliminar del sistema.

Aplazable. Puede realizarse junto a otras fases del análisis.

Un sistema bien diseñado puede agruparse en paquetes. Cada paquete realiza una acción, o un conjunto de acciones bien definido. Los paquetes pueden ubicarse en diferentes capas del sistema, creando así aplicaciones multicapa.

El diagrama de paquetes muestra el diseño de capas de la aplicación y las relaciones entre las diferentes capas y paquetes.

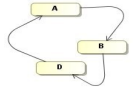


Dibuix 12: Ejemplo de diagrama de arquitectura

## Estratos y particiones.

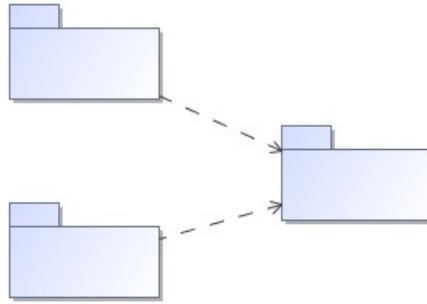
Un estrato es la agregación de diferentes paquetes, y que definen una capa del diseño.

Una partición se corresponde con uno o varios paquetes de un estrato, (o capa).



## Dependencia de Paquetes.

Si una clase de un paquete guarda relación con otra clase de otro paquete, es que hay dependencia entre paquetes. La dependencia se dibuja con una línea discontinua. La navegabilidad viene definida por la navegabilidad de las clases a las que hace referencia.



*Dibuj 13: Ejemplo de dependencia entre paquetes*

## Como partir el diagrama de entidades en paquetes.

Las entidades con alguna de las siguientes características podrían conformar un paquete:

- Las que se encuentran en la misma área o tema. Que estén estrechamente relacionados por un concepto o propósito.
- Las que se encuentren en la misma jerarquía de tipos.
- Las que participen en los mismos casos de uso.
- Las que presenten una asociación muy íntima.

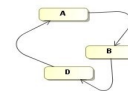
---

## Perfeccionar el plan.

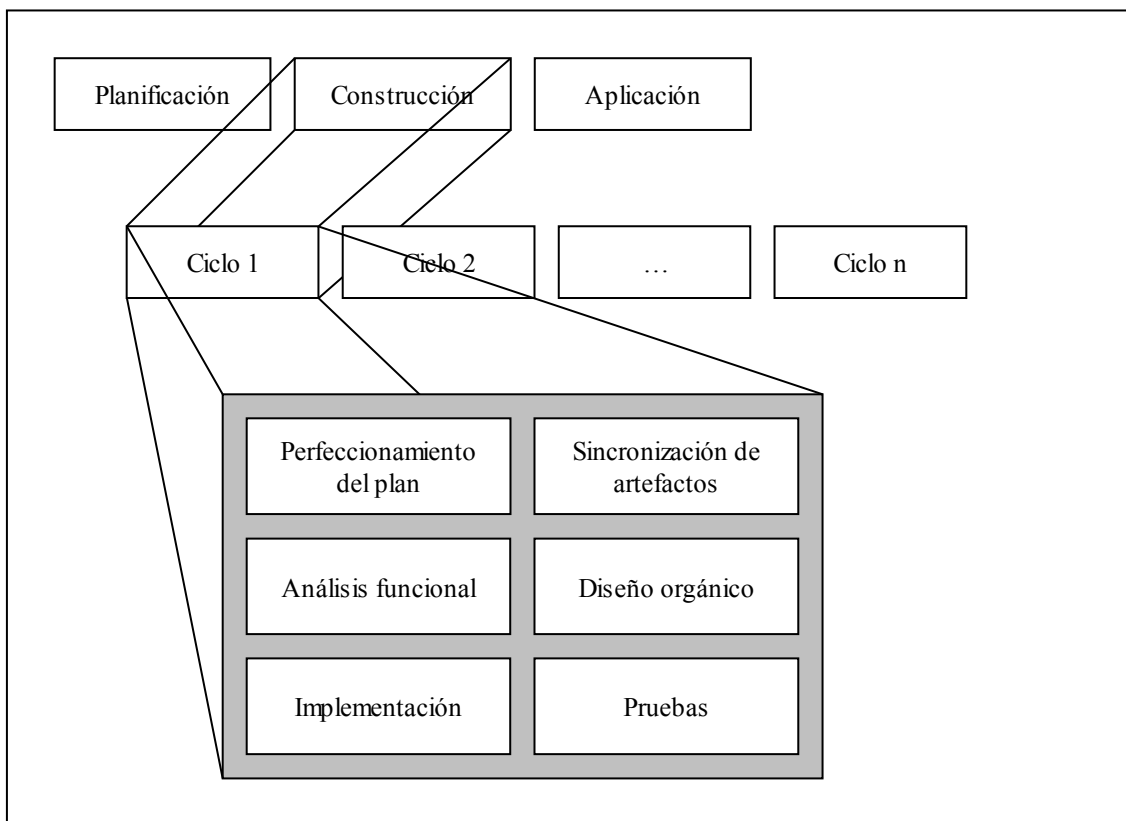


Véase: [Definir el plan preliminar.](#)





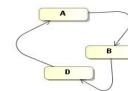
## Fase de: Construcción



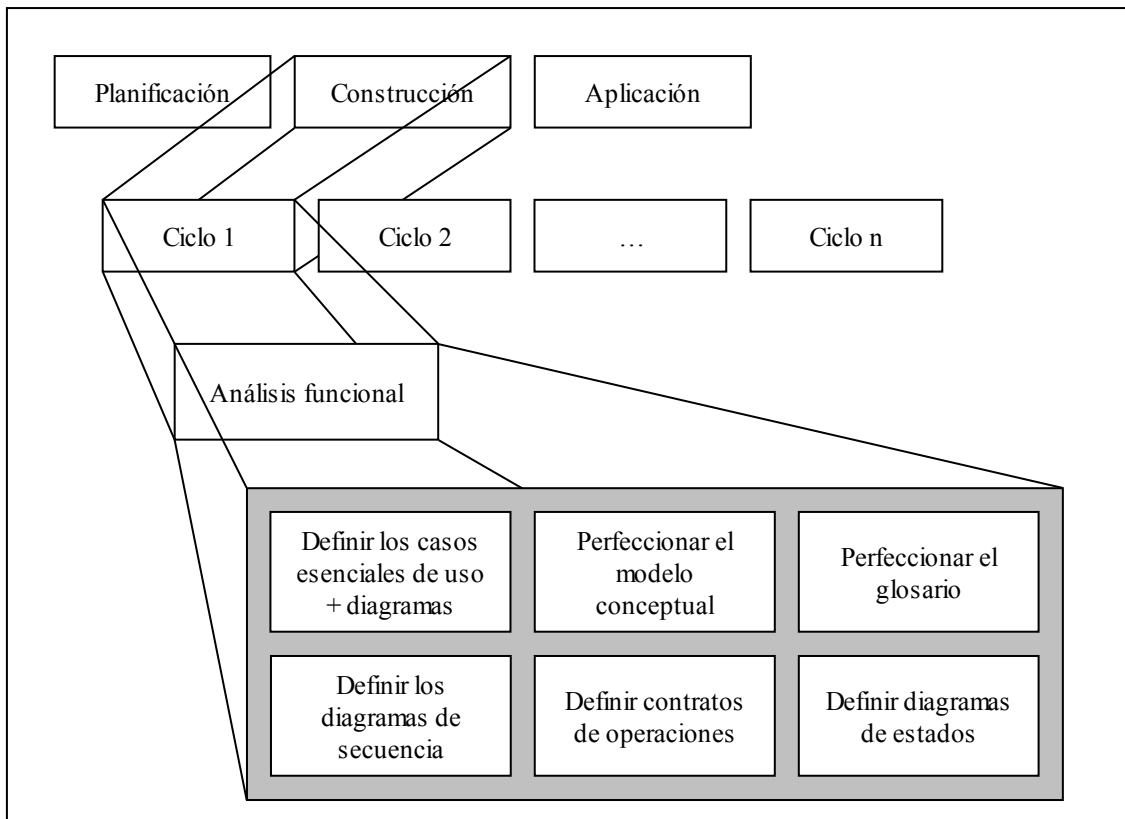
Dibuix 14: Tareas de la fase de construcción

### Perfeccionamiento del plan.

! Véase: [Definir el plan preliminar.](#)



## Análisis funcional.

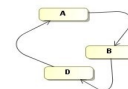


Dibuj 15: Subtareas de la tarea de análisis funcional

### Definir los casos esenciales de uso + diagramas.

Si no se hizo en una fase anterior, o en un ciclo anterior de esta fase.

! Véase: [Casos de uso de alto nivel y esenciales.](#)



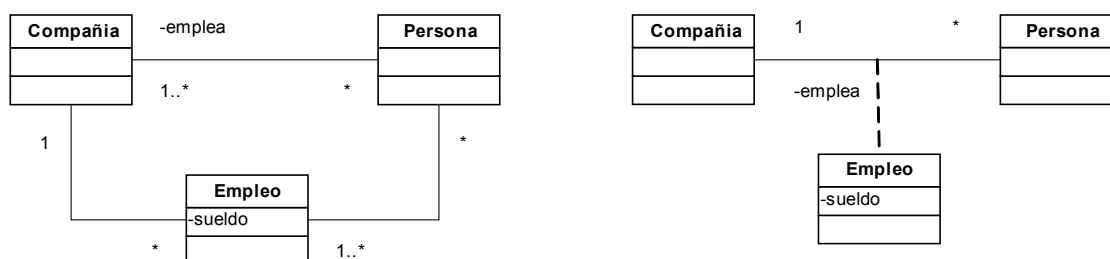
## Perfeccionar el modelo conceptual.

Perfeccionar las asociaciones y otros conceptos del diagrama de entidades obtenido en fases anteriores.

! Véase: [Definir el modelo conceptual preliminar.](#)

### Clases asociativas.

Opcional. Una clase ayuda y caracteriza la vinculación de otras dos. La asociación que une esas dos clases puede tener atributos.



Dibuix 16: Ejemplo de transformación en clase asociación

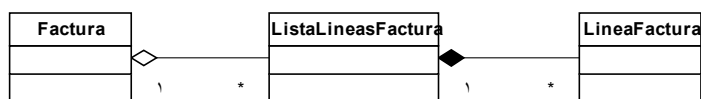
### ¿Cuándo un tipo asociativo?

- Un atributo está relacionado con una asociación.
- Hay una relación de muchos a muchos entre dos clases. Y la información se relaciona con la propia asociación.
- Sólo existe una instancia del tipo asociativo entre dos objetos que participan en la asociación.

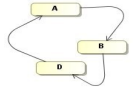
### Agregaciones.

#### ¿Cuándo?

- La duración de la parte es dependiente de la que tiene el compuesto. Es decir: Cuando desaparece el origen, desaparecen sus partes.
- Existe un evidente ensamblado físico o lógico de parte – todo.
- Algunas propiedades del compuesto se difunden hacia las partes, entre ellas su ubicación.
- Las operaciones aplicadas al compuesto se propagan hacia las partes: Destrucción, movimiento, registro, etc.



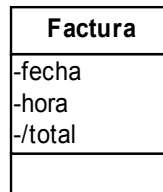
(La primera asociación corresponde a una agregación. La segunda a una composición).



### Atributos derivados.

Son aquellos que vienen definidos por el cálculo de otros atributos presentes en otras clases con las que hay alguna asociación. No muestre los atributos derivados a no ser que sea necesario.

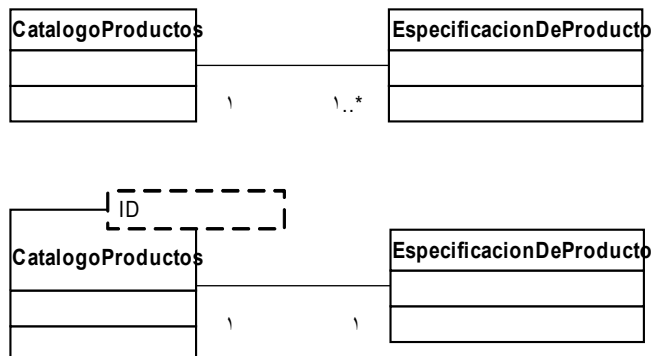
Un atributo derivado se indica en el diagrama de entidades, (y posteriormente en el de clases), precediéndolo de una “/”. Por ejemplo: El atributo “total” de la clase “Factura” es el resultado del total línea de cada línea de factura.



*Dibuj 17: Ejemplo de atributo derivado*

### Asociaciones cualificadas.

Opcional. Una clase A mantiene una asociación con una clase B dependiendo del valor de un atributo, (cualificador), de la clase A.

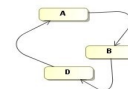


*Dibuj 18: Ejemplo de asociación cualificada*

Una asociación cualificada transforma una asociación general de 1 a muchos, a de 1 a 1.

### Perfeccionar el glosario.

! Véase: [Glosario](#).



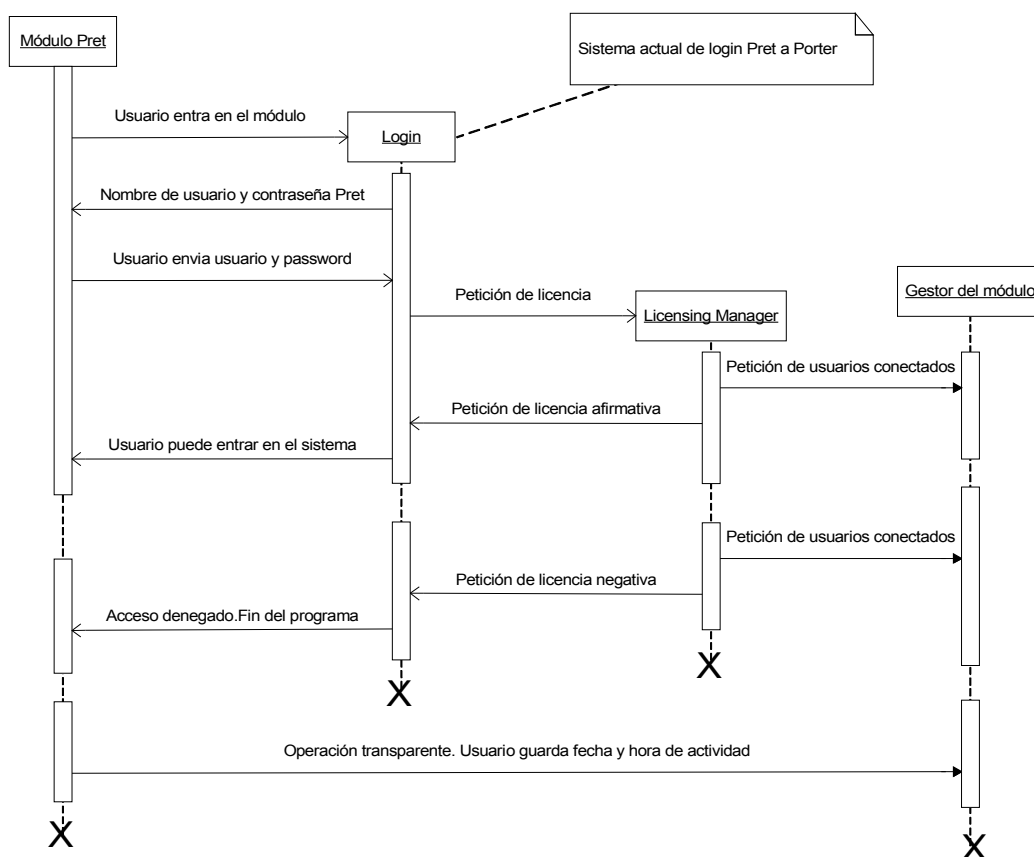
## Definir los diagramas de secuencia.

Muestra gráficamente los eventos que fluyen de los actores del sistema. Explica lo que hace, (el sistema), pero no como lo hace. El diagrama de secuencia se crea en relación al caso de uso. Ambos explicarán las interacciones de los actores y los eventos y operaciones que se desprenden de esas interacciones.

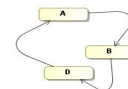
El diagrama de secuencia describe las operaciones del sistema. Estas operaciones, si lo necesitan, pueden llevar parámetros.

! Véase: [Diagramas de interacción, \(colaboración y secuencia\).](#)

### Ejemplo de: Diagramas de secuencia.



Dibuix 19: Ejemplo de diagrama de secuencia



## Definir contratos de operaciones.

Los contratos definen el efecto que tienen sobre el sistema las operaciones. Se basan en “Precondiciones” y “Postcondiciones”.

Cada operación del sistema, extraídos del diagrama de secuencia, se define mediante la siguiente plantilla:

<b>Nombre:</b>	Nombre de la operación y sus parámetros.
<b>Responsabilidades:</b>	Descripción de la operación.
<b>Tipo:</b>	Concepto, Entidad, Interfaz ...
<b>Referencias cruzadas:</b>	Relación con otras operaciones.
<b>Notas:</b>	Información a tener en cuenta para esta operación.
<b>Excepciones:</b>	Casos y circunstancias en que se produce una excepción en esta operación. Descripción.
<b>Salida:</b>	Transacciones a otras clases.
<b>Precondiciones:</b>	Que ha de ocurrir antes que se inicie la operación.
<b>Postcondiciones:</b>	Como debe quedar el sistema al acabar la operación.

Taula 10: Plantilla para la definición de contrato de operacioens

Las postcondiciones deben indicar:

- Creación y eliminación de instancias.
- Modificación de atributos.
- Asociaciones formadas y canceladas.
- Deben reflejar los cambios de estado y los resultados, (pero no las acciones).
- Han de reflejar lo que hacen, pero no como lo hacen. No han de adentrarse en el diseño.
- Las postcondiciones hablan siempre en pasado. “Se creó”, “se produjo”, “se eliminó”, etc.

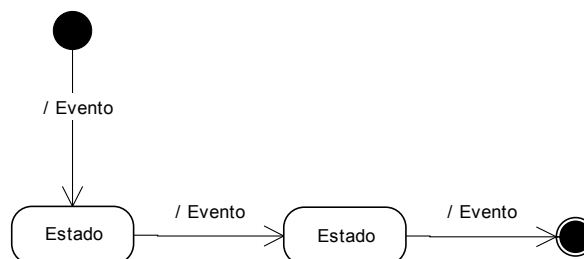
**Generalmente, los contratos de operaciones se vinculan con los casos de uso. Para cada caso de uso definido, se indican las precondiciones y postcondiciones que le afectan.**

## Definir diagramas de estado.

Opcional

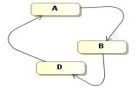
Descripción gráfica de los eventos y estados por los que pasa un objeto.

Un evento es un acontecimiento importante o digno de remarcar. Un estado es una condición de un objeto en un momento determinado.



Un diagrama de estados puede aplicarse sobre:

- Casos de uso.
- Clases de software.
- Entidades de un modelo conceptual.



### Diagramas de estado del sistema.

Es un resumen de todos los diagramas de estado. Y muestran las transiciones de eventos y estados que ocurren en el sistema de forma general.

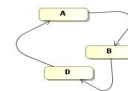
### Diagramas de estado de clases.

Generalmente hay pocas clases que sean merecedoras de su propio diagrama de estados:

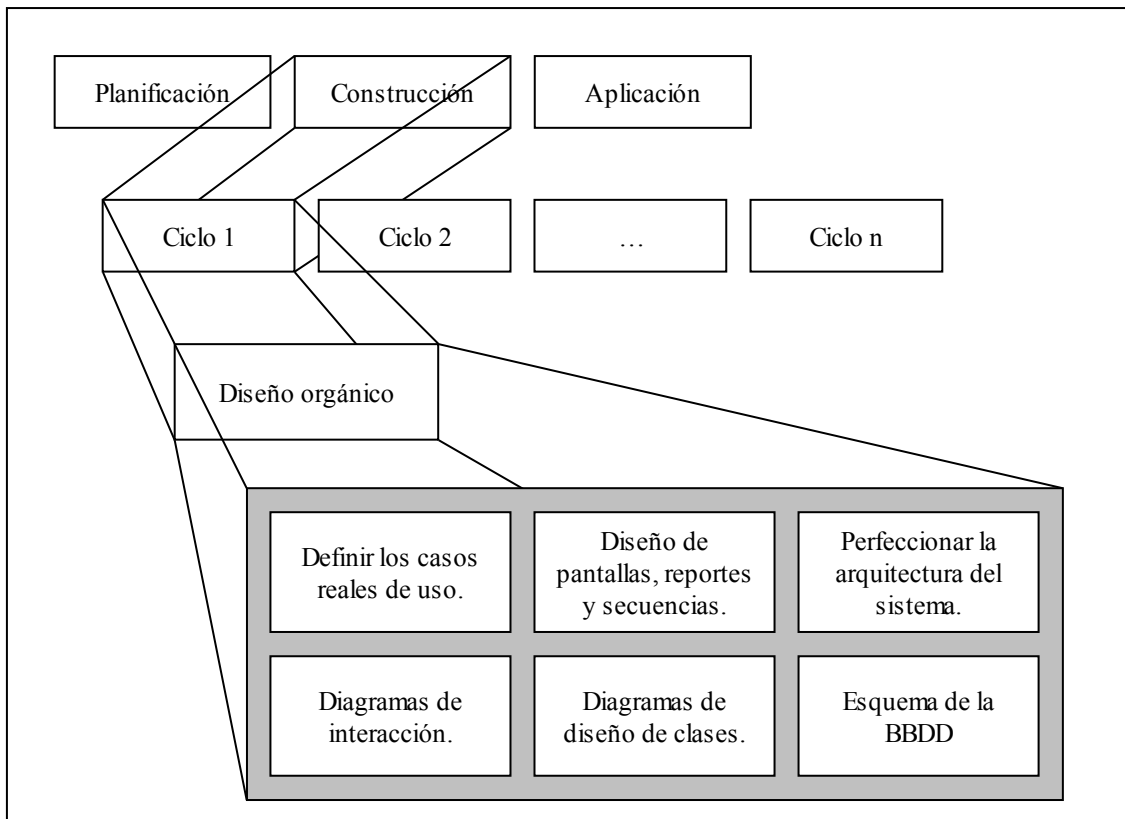
- Controladores de eventos.  
Dependiendo del evento realizan una función u otra.
- Transacciones.  
Una clase inicia una transacción al recibir cierto evento. Durante la transacción recibe otros eventos a los que debe reaccionar.
- Coordinadores.  
Por ejemplo: Applets de Java.
- Ventanas.  
Ciertas acciones en ventanas sólo son posibles dependiendo del estado de otros objetos. Por ejemplo: Cortar y pegar sólo es posible si el portapapeles tiene algún contenido.
- Clases de dispositivo.  
Reaccionan de una forma u otra dependiendo de su estado actual.

### Herencia: Modelo de estados cambiantes.

No modele los estados de una clase X como subclases de esta. Defina una jerarquía de estados y asóciela a X. Los estados de una clase no se definen en el diagrama de entidades



## Diseño orgánico.



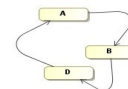
Dibuj 20: Subtareas de la tarea de diseño orgánico

### Definir los casos reales de uso.

Esta vez, la definición debe hacerse en profundidad. Y no sólo sobre lo esencial.

De los casos de uso de la fase de análisis, se derivan los casos reales de uso. A los que puede añadirse el diseño de pantallas.





## Diseño de pantallas, reportes y secuencias.

El diseño de pantallas, reportes, etc. ha de ser tomado como un diseño de orientación al cliente sobre el aspecto y la secuencia de funcionamiento de las partes de interacción Usuario – Máquina.

Caja	Estado
1	OK
2	
3	OK
4	

Dibuj 21: Ejemplo de definición de interfaz gráfica

El diseño de pantallas nunca puede representar un contrato estricto de diseño para el cliente. El diseño final dependerá del lenguaje, la plataforma y las decisiones finales del diseño. El diseño de pantallas es de carácter orientativo y sirve para que el cliente tome conciencia del resultado final de las interfaces del proyecto.

El diseño de pantallas también puede ser una competencia del análisis de requerimientos. Puesto que ayuda al entendimiento del problema por parte de los usuarios.

Si se definió prototipo, este puede substituir esta definición de interfaz gráfica.

## Perfeccionar la arquitectura del sistema.

! Véase: [Definir la arquitectura preliminar del sistema.](#)

### Patrón: FACHADA.

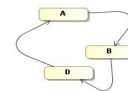
Una fachada es una clase que sirve de interfaz pública para todas las clases de un paquete. Por ejemplo. El paquete BDR contiene multitud de clases para gestionar una BBDD. Pero sólo una clase, (la fachada), contiene los métodos públicos que ofrecen los servicios de gestión de BBDD.

El resto de clases del paquete deben ser privados.

### Patrón: MODELO – VISTA o DOMINIO – PRESENTACIÓN

Las clases del dominio nunca deben conocer directamente los objetos vista, ni estar directamente acoplados a ellos.

Es correcto que los formularios envíen mensajes a objetos del dominio solicitando información. Pero no es correcto que objetos del dominio se comuniquen con los formularios para mostrar información.



**Patrón: PUBLICAR – SUBSCRIBIR.**

Una clase del dominio puede comunicarse con formularios de forma indirecta, mediante eventos.

El formulario se “suscribe” a los eventos que le interesan de la clase del dominio. La clase del dominio hace saltar sus eventos a medida que se produce la información. La suscripción a eventos por parte de los formularios acostumbra a hacerse mediante interfaces.

Las funciones de suscripción pueden implementarse en las propias clases del dominio afectadas. O crear una clase de administración de eventos general.

Con el patrón Publicar – Suscribir pueden realizarse aplicaciones multi-hilo.

- ! Véase: [Gestores y administración de eventos.](#)
- Véase: [De diagramas de colaboración a métodos en clases. Patrones.](#), (Controlador)

**Patrón: COORDINADOR – GESTOR.**

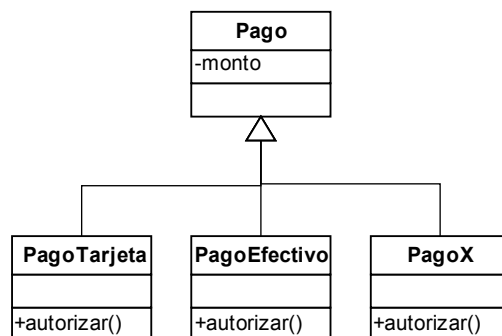
Un gestor es una clase que intercede entre los formularios y las clases del dominio.

Los formularios hacen peticiones al gestor, y este les da respuesta después de consultar con las clases del dominio. Un gestor a de:

- Consolidar y transformar la información proveniente de los objetos del dominio.
- Responder a eventos provenientes de los formularios.
- Abrir ventanas que muestren la información proveniente de los objetos del dominio.
- Administrar las transacciones, (Commit y Rollback).

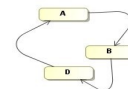
**Patrón: POLIMORFISMO.**

Diversas clases relacionadas por herencia realizan de forma diferente el mismo método.



*Dibuix 22: Ejemplo de polimorfismo*

Para nada es aconsejable el uso de “If” o “Case” para diferenciar tipos. Generalmente la clase base en un diagrama de herencia de este tipo suele ser abstracta.



**Patrón: FABRICACION PURA.**

Una clase precisa de un método. Ese método puede ser necesario en otras muchas clases. O bien, aunque la clase precisa de ese método, se sale del planteamiento general de la clase. Es decir: La implantación de ese método en la clase provocaría:

- Baja cohesión.
- Alto acoplamiento.
- Poca reutilización.
- Mucho código duplicado.

Ante alguno de esos síntomas se aconseja crear una nueva clase que realice esas funciones. A esa clase se le denomina “Fabricación Pura”.

Por ejemplo:  
Métodos para guardar objetos en una BD.

**Patrón: INDIRECCIÓN.**

Cuando una clase necesita servicios de otra que le es completamente ajena, y a la que no se quiere acoplamiento.

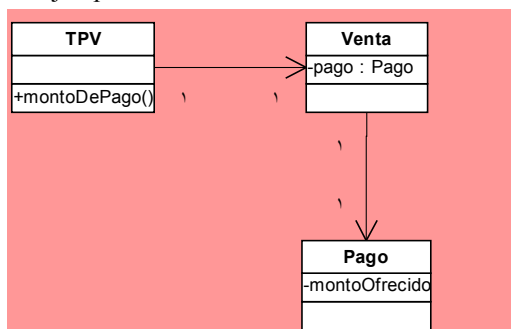
Por ejemplo:  
Un servicio de envío de archivos accede a la clase “Modem”, que es la clase que se encarga de acceder a los servicios API de comunicaciones. “MODEM” cumple el patrón de Indirección.

Los patrones de [Patrón: PUBLICAR – SUBSCRIBIR.](#), [Patrón: AGENTE y AGENTE REMOTO.](#), [Patrón: FACHADA.](#) y [Patrón: DISPOSITIVO.](#) son también un caso de indirección.

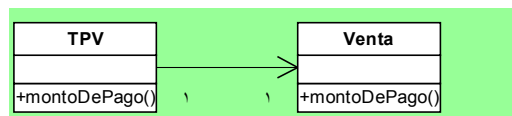
**Patrón: NO HABLES CON EXTRAÑOS.**

Una clase tiene conexión con otra. Y usa una conexión de esta con una tercera clase. Esta situación no cumple el patrón de “No hables con extraños”. Para solucionar esto es necesario crear un atributo en la segunda clase que acceda a la información deseada de la tercera.

Por ejemplo:

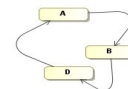


Incorrecto.  
No cumple el patrón “No hables con extraños”  
TPV.montoPago = Venta.Pago.montoOfrecido



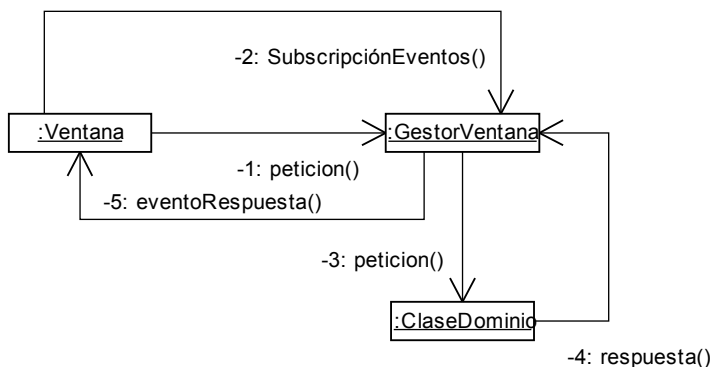
Correcto.  
TPV.montoDePago = Venta.montoDePago()  
Venta.montoDePago = Pago.montoOfrecido.

*Dibuix 23: Ilustración del patrón "No hables con extraños"*



## Gestores y administración de eventos.

### Ejemplo de gestor:



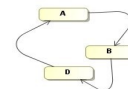
Dibuix 24: Ilustración del uso de gestores

Un formulario requiere de una información de un objeto del dominio. En lugar de realizar la petición directamente a dicho objeto, utiliza una clase intermedia que funciona como “Gestor del formulario”. Es el gestor quien recibe la petición del formulario. Seguidamente el gestor realiza una petición semejante al objeto del dominio. Al recibir la respuesta este se la envía finalmente al formulario.

Es en la forma de dar la respuesta es donde pueden o no entrar en juego los eventos. Si la petición que realiza el formulario se hace en forma de función que espera un valor o, como es el caso del diagrama, el formulario recibe un evento por parte del gestor, (`eventoRespuesta()`), con la respuesta a la petición.

El gestor del formulario y el formulario pueden estar unidos. Y generalmente existe una jerarquía de gestores para controlar todos los formularios de la aplicación.

! Véase: [De diagramas de colaboración a métodos en clases. Patrones.](#) (Controlador)



## Diagramas de interacción, (colaboración y secuencia).

Con el diagrama de entidades + los contratos de operaciones; ambos obtenidos en la fase de análisis. + los casos reales de uso; podemos generar los diagramas de interacción.

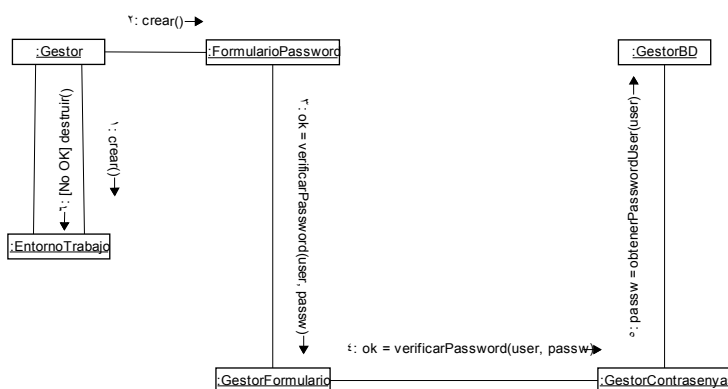
Los diagramas de interacción explican gráficamente como los objetos y el usuario interactúan para realizar las tareas. Los diagramas de interacción son:

- Diagramas de colaboración.
- Diagramas de secuencia.

! Véase: [Definir los diagramas de secuencia.](#)

Es en los diagramas de interacción donde se aplican los patrones de diseño de software.

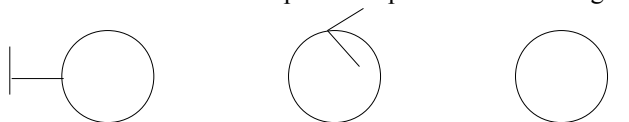
### Ejemplo de: Diagramas de colaboración.



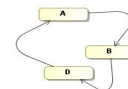
Dibuj 25: Ejemplo de diagrama de colaboración

### Otra representación de los diagramas de colaboración

Los diagramas de colaboración también pueden representarse de la siguiente forma:



Los símbolos corresponden a Pantalla, Control y Dominio respectivamente



### Como diseñar diagramas de colaboración.

- 1- Elabore un diagrama para cada operación del sistema del ciclo actual de desarrollo.  
En cada mensaje del sistema dibuje un diagrama incluyendolo como mensaje inicial.
- 2- Si el diagrama se torna complejo, divídalos.
- 3- Use las precondiciones y postcondiciones del contrato de la operación sobre la que realiza el diagrama de colaboración.  
Use también la descripción del caso de uso y aplique patrones si es necesario.

De los diagramas de colaboración se derivan los métodos que finalmente se aplicarán a las clases. La definición de esos métodos se basa en los siguiente principios:

- **Hacer**
  - Hacer algo en uno mismo.
  - Iniciar una acción en otros objetos.
  - Controlar y coordinar actividades en otros objetos.
- **Conocer**
  - Estar enterado en los datos privados encapsulados.
  - Estar enterado de la existencia de objetos conexos.
  - Estar enterado de cosas que se pueden derivar o calcular.

### De diagramas de colaboración a métodos en clases. Patrones.

#### Patrón: EXPERTO.



Dibuix 26: Ilustración del patrón experto

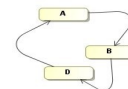
Aquella instancia que reciba un mensaje, deberá tener acceso a todo lo necesario para poder realizar el cálculo que le requieren. Y será su clase quien implemente el método derivado.

#### Patrón: CREADOR.

Desde un objeto A se crean y se mantienen objetos B.

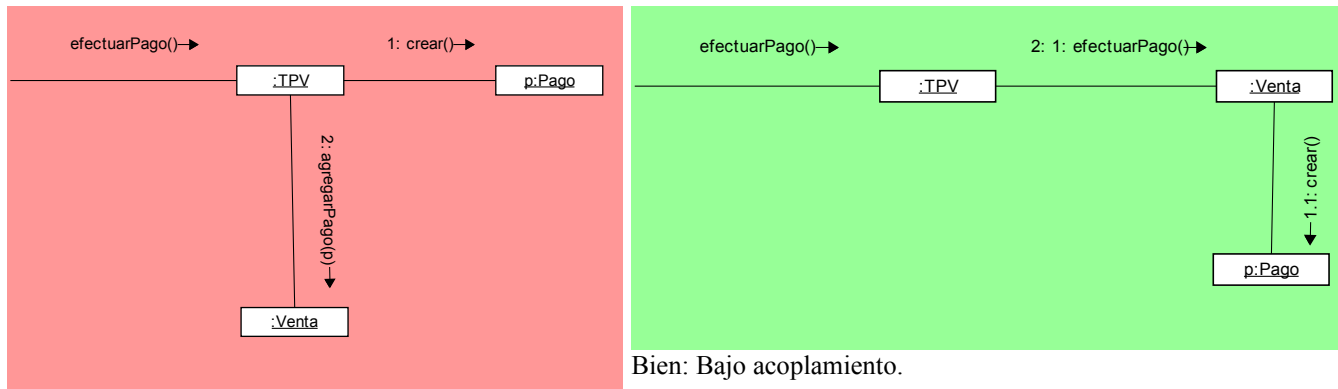


Dibuix 27: Ilustración del patrón Creador



**Patrón: BAJO ACOPLAMIENTO.**

Cuando un método puede vincularse a dos clases, ha de elegirse la clase con menor acoplamiento.



Mal: Alto acoplamiento.

Bien: Bajo acoplamiento.

*Dibuix 28: Ilustración del patrón Bajo acoplamiento*

El bajo acoplamiento favorece la reutilización. La herencia es la forma mas alta de acoplamiento; por lo que ha de tratarse con cuidado.

El acoplamiento bajo llevado al extremo produce diseños donde una clase hace todo el trabajo, y el resto de clases son meros contenedores de datos. Esto corresponde a un mal diseño.

**Patrón: ALTA COHESIÓN.**

Las clases han de diseñarse de forma que hagan un trabajo concreto y cohesionado. Una clase con baja cohesión hace muchas cosas, muy variadas y muy diferentes. Una clase así estará mal diseñada.

El mismo ejemplo que en el patrón de bajo acoplamiento es aplicable aquí.

**Patrón: CONTROLADOR.**

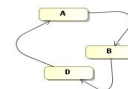
Un controlador es una clase con un método que responde a un evento del sistema o de otra clase. Un usuario pulsa el botón “terminar venta”. Esto produce un evento que es capturado por el método `terminarVenta()`. La ubicación de este método dependerá del diseño, pero la clase que lo contenga será “Controladora”

La clase que contenga este método deberá guardar relación con las funciones que realiza; siguiendo el patrón de “Alta cohesión”. Una clase que “controle” todos los eventos del sistema, estará mal diseñada.

Los controladores de tipo humano, (cajero, vendedor, etc), que manejan eventos, generalmente no son eficientes, porque generalmente suelen ser poco cohesionados.

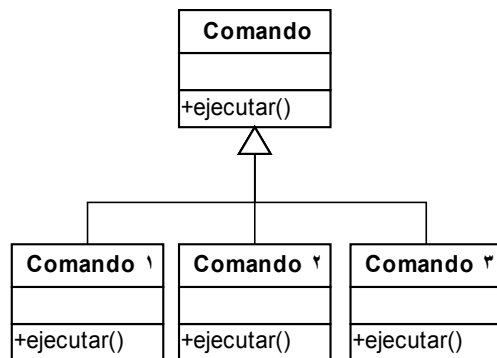
La capa de presentación nunca debe manejar los eventos. Sólo provocarlos y actuar al recibirlos.

- ! Véase: [Gestores y administración de eventos.](#)
- Véase: [Patrón: PUBLICAR – SUBSCRIBIR.](#)



**Patrón: COMANDO.**

En aplicaciones sin interfaz de usuario, en la que deben ejecutarse diversos comandos, se crea una clase “Comando” con el método `ejecutar()`. Y se hereda de ella para cada comando que la aplicación implemente.



*Dibuix 29: Ilustración del patrón Comando*

Una clase llamada “Manejador” ejecutará una clase u otra según necesite

**Conectar la capa de presentación y la del dominio.**

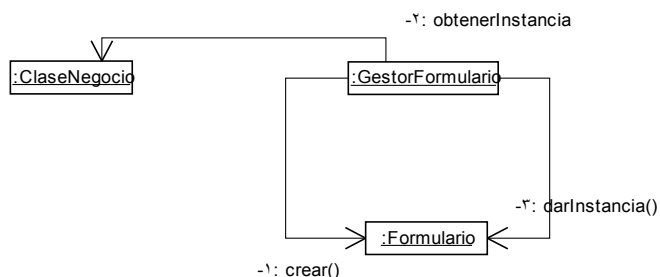
**Opción 1:**



*Dibuix 30: Presentación y Dominio conectados directamente*

El formulario se crea, y este pide a la clase inicial que le asigne una instancia de la clase Informativa. A partir de ese momento, el formulario se comunicará con la clase. Con esta opción, la clase de negocio es creada por el formulario.

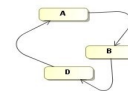
**Opción 2:**



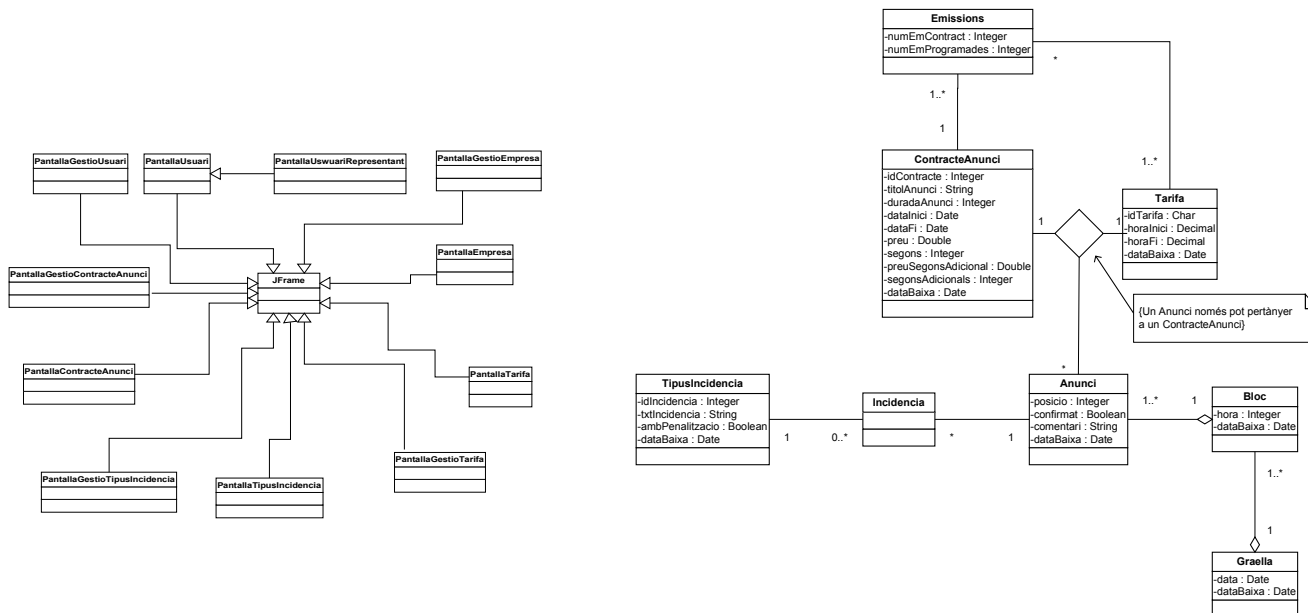
*Dibuix 31: Presentación y Dominio conectados mediante gestores y subscripción a eventos*

La clase de negocio nunca debe comunicarse directamente con la clase de presentación. Es el gestor del formulario el que enlaza una con otra.





## Diagramas de diseño de clases.



Dibuix 32: Ejemplos de diagramas de clases

### El análisis y el diseño son una estimación.

Los diagramas de colaboración son una aproximación, pero nunca serán perfectos. Los diagramas de colaboración se basan en el diagramas de entidades, que tampoco nunca será perfecto. Es posible que surgan conceptos nuevos no previstos; o que conceptos previstos no sean necesarios.

En los casos de uso reales, los contratos que inicialmente se establecieron, (precondiciones y postcondiciones), no son inmutables.

### Elección de la clase inicial.

La clase inicial debe ser:

- Una clase que represente todo el sistema de información lógico.
- y/o
- Una clase que represente integralmente el negocio u organización.

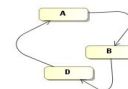
### Diagramas del diseño de clases.

Los diagramas del diseño de clases se crean a partir de:

- Diagramas de interacción.
- Diagrama de entidades.

Para crear un diagrama de clases:

- Identifique las clases a partir del diagrama de entidades.



- Dibuje el diagrama de clases.
- Represente los atributos a partir del diagrama de entidades.
- Represente los métodos a partir de los diagramas de interacción.
- De tipos a atributos, a métodos y parámetros.
- Dibuje las asociaciones a partir del diagrama de entidades, (y otros).
- Asigne navegabilidad a las asociaciones.
- Asigne las relaciones de dependencia no relacionadas con atributos, (líneas punteadas).

### Partición del diagrama de clases.

Generalmente existe un diagrama de paquetes. Para cada paquete se realiza un diagrama de clases.

Cada diagrama de clases suele vincularse con un diagrama de jerarquías, que contiene:

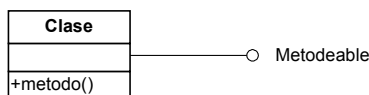
- Un diagrama de clases del dominio.
- Un diagrama de excepciones.
- Un diagrama de gestores.
- Un diagrama de pantallas.

### Otras consideraciones.

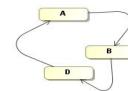
- Los constructores suelen omitirse del diagrama de clases. No así de las fichas CRC.
- Los atributos suelen ser siempre privados, y disponer de métodos get y set. Estos métodos no suelen figurar en el diagrama de clases. No así en las fichas CRC.
- Los atributos derivados de las asociaciones no se muestran en el diagrama de clases. No así en las fichas CRC.
- Los multiobjetos pueden estar basados en una clase de apoyo específica que contenga todas las operaciones para navegar en la colección que conforman. Pero dichas operaciones no pueden implementarse en la clase objeto, gestora del multiobjeto.

! Véase: [Fichas CRC](#).

### Interfaces.

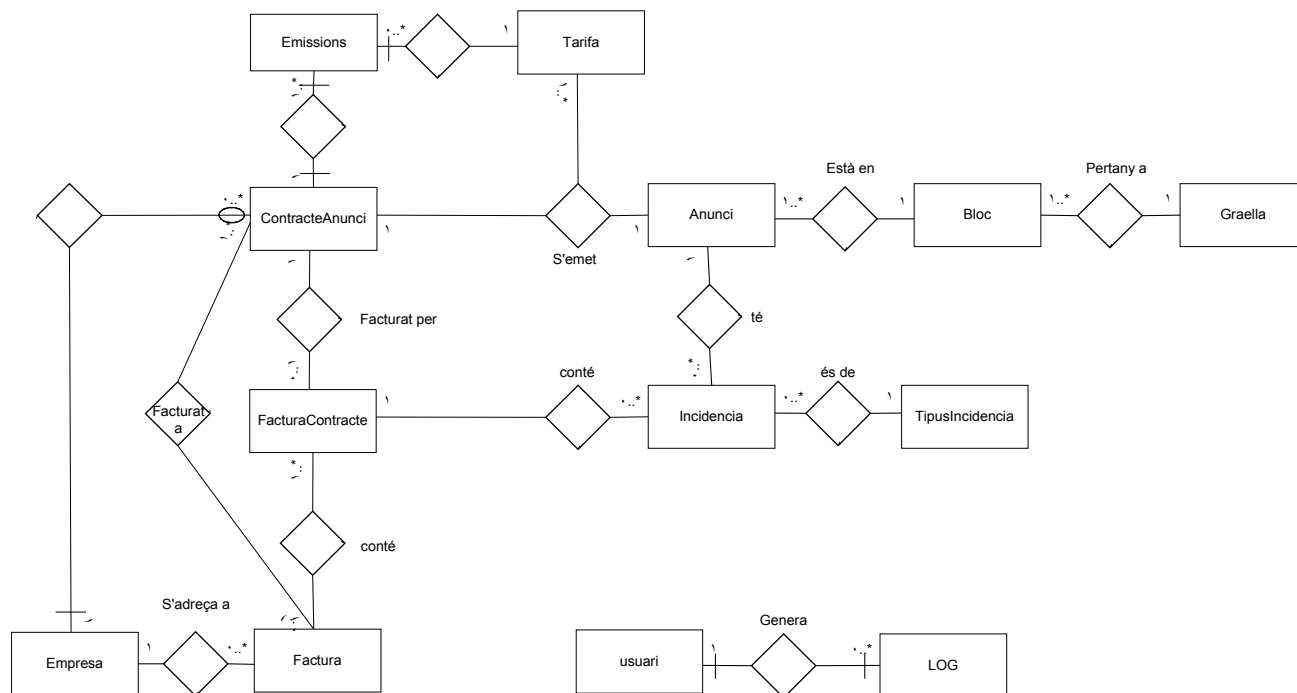


Dibuj 33: Ejemplo de definición de interfaces



## Esquema de la BBDD.

Definir las tablas con sus atributos y claves. Describir cada una de las tablas.  
 Generar un diagrama de entidad/relación.



Dibuix 34: Ejemplo de diagrama ER

## Patrones GOF.

### Patrón: ESTADO.

El comportamiento de un objeto depende de su estado. La lógica condicional no es adecuada a causa de su complejidad, escalabilidad o duplicación. Por lo tanto:

- Crear una clase para cada estado que influye en el comportamiento del objeto.
- Con base en el polimorfismo, asignar métodos para manejar el comportamiento del objeto en cada estado.
- Cuando el objeto reciba un mensaje dependiendo del estado, el mensaje será enviado al objeto raíz de la jerarquía de estados.

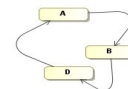
### Patrón: SINGLETON.

Una clase puede acceder a los servicios de otra a la que no está conectada, mediante un método estático que devuelve la instancia actual de la clase.

### Patrón: DISPOSITIVO.

Es un tipo de patrón “Fachada”, pero especializado para dispositivos hardware. Por ejemplo: Una clase llamada “Modem”.

! Véase: [Patrón: FACHADA.](#)



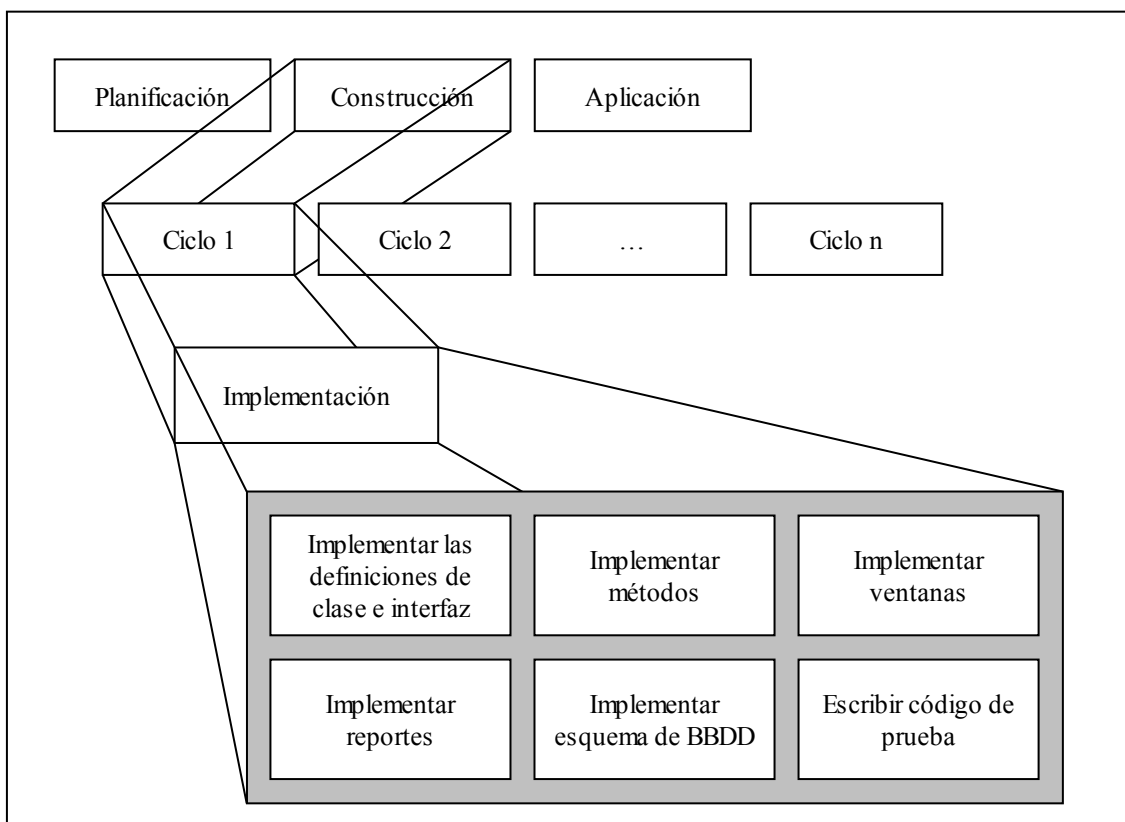
**Patrón: AGENTE y AGENTE REMOTO.**

Algunos servicios que son competencia de una clase, no se pueden o no se quieren acceder directamente. Se crea una clase y se le asigna la responsabilidad de realizar esos servicios.

**Patrón: AGENTE VIRTUAL.**

Una clase interactua con otra, que es una representante, (agente virtual), de una tercera. El objeto real puede estar guardado en una BD. El agente virtual hace de intermediario; y posee métodos para materializar el objeto real, bajo demanda de la primera clase.

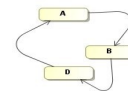
**Implementación.**



*Dibuix 35: Subtareas de la tarea de implementación*

**Orden de implementación.**

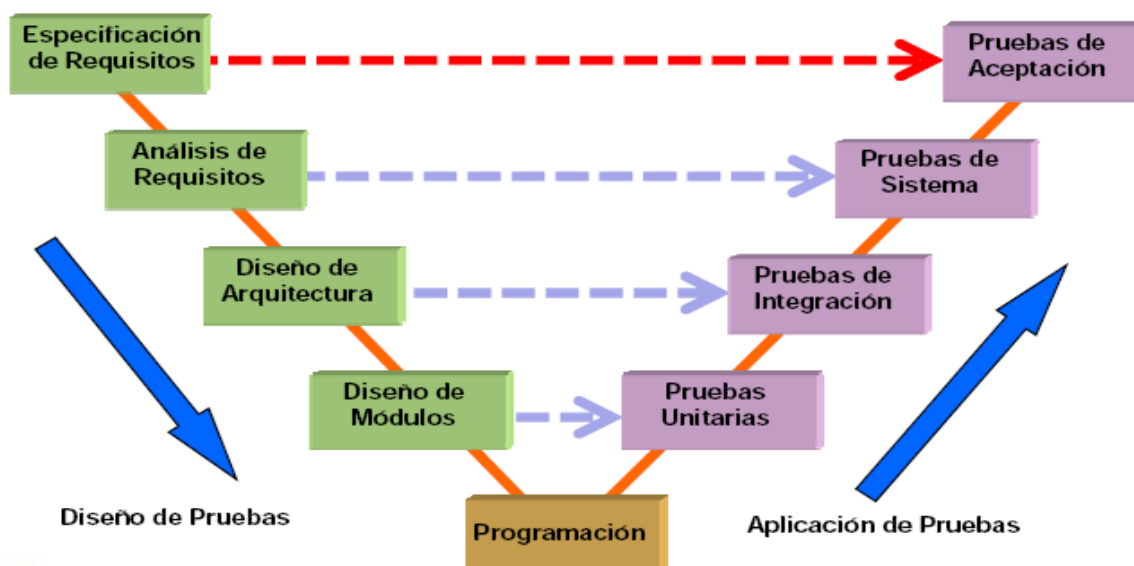
Es conveniente implementar primero las clases menos acopladas o periféricas. Y luego implementar las clases que dependen de estas.



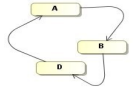
## Pruebas.

Existen diferentes modalidades de pruebas, entre las que podemos distinguir:

- **Pruebas unitarias, (Pruebas de unidad)**  
Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.
- **Pruebas de integración.**  
Posteriormente a las pruebas unitarias, las Pruebas de Integración verifican el correcto funcionamiento del sistema o subsistema sometido a prueba.
- **Pruebas del sistema.**  
Estas pruebas son previas a las pruebas de aceptación. El equipo de desarrollo comprueba el correcto funcionamiento de todo el sistema, y verifica que todos los requerimientos son coherentes.
- **Pruebas de desempeño.**  
Como parte de pruebas del sistema, las pruebas de desempeño se concentran en la forma en que los sistemas realizan el trabajo, realizando diversos test que comprueban:
  - El rendimiento del hardware
  - El rendimiento de la base de datos en condiciones de alto rendimiento
  - Pruebas de recuperación ante escenarios de mal funcionamiento de partes de hardware o software necesarios
- **Pruebas de aceptación.**  
Estas pruebas las realiza el cliente. Son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos y del manual del usuario. Son similares, (o pueden incluirse en parte), a las pruebas de sistema.
- **Pruebas de regresión**  
Estas pruebas verifican el correcto funcionamiento de todo el sistema después de un proceso de corrección o evolución del mismo.

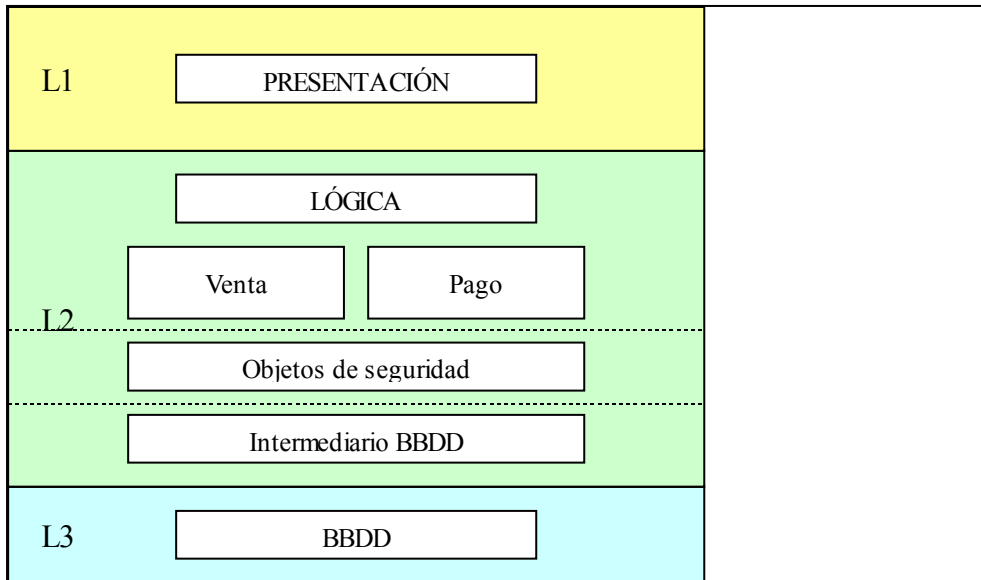


Dibuix 36: Modelo de pruebas



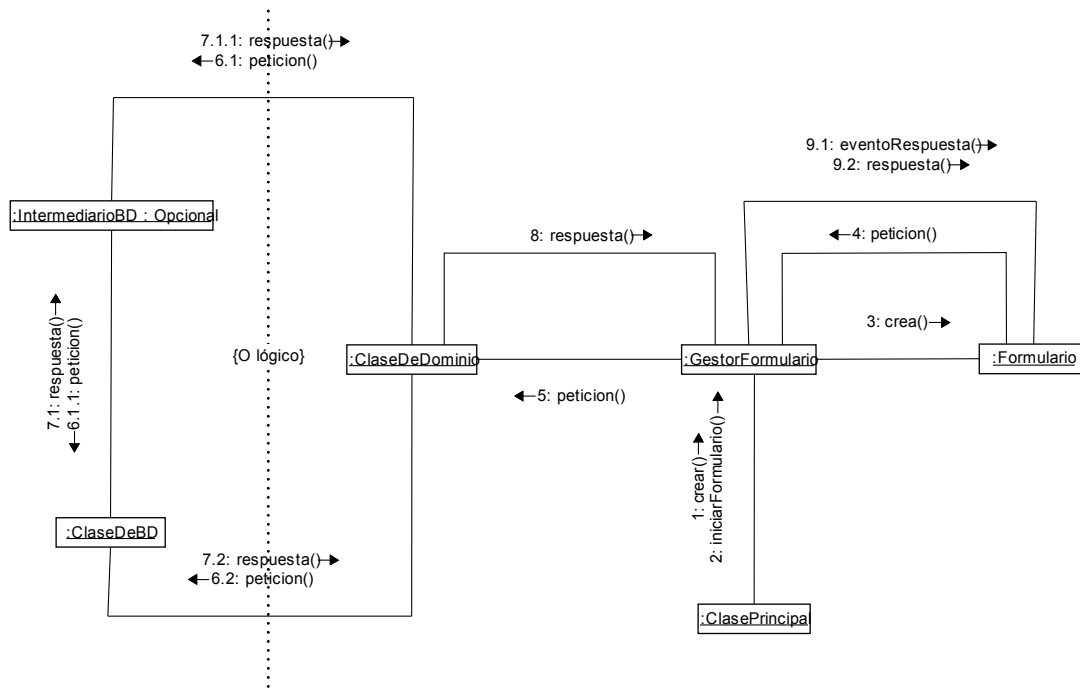
## Apéndices

### Capas de un sistema ordinario de información orientado a objetos.

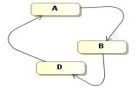


Dibuix 37: Ilustración de arquitectura multicapa

### Implementación de la comunicación entre capas.



Dibuix 38: Modelo de comunicaciones multicapa



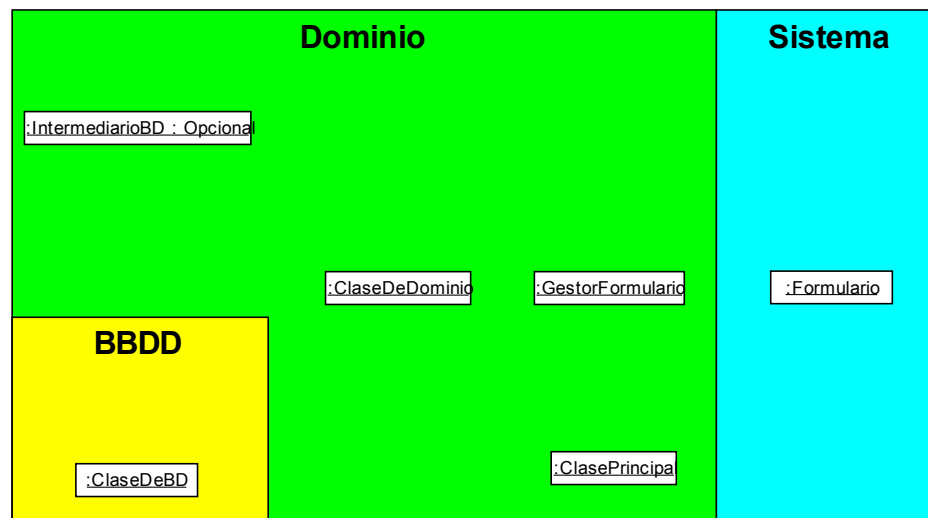
La clase `Principal` crea una instancia de `GestorFormulario`. Este gestor hace de puente entre las clases de dominio y las de presentación. `GestorFormulario` crea y abre el formulario al cual gestiona. `GestorFormulario` recibe peticiones del formulario. Las gestiona con las clases del dominio que precise.

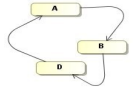
Las clases del dominio contactan con las clases de gestión de BD directamente o a través de un intermediario.

### Comentarios.

1. La capa de presentación y la de dominio nunca deben conectarse de forma directa. El formulario puede solicitar información a una clase del dominio. Pero la clase del dominio nunca debe conectarse con el formulario para darle la información solicitada por este. Para esto existen los eventos. El formulario puede subscribirse a los eventos de la capa de dominio que precise. Al ocurrir un evento, el formulario recibe la información solicitada.
2. El diagrama de relaciones arriba expuesto es independiente a otras estructuras que puedan haber presentes en la aplicación, (por ejemplo, excepciones).

### Disposición de capas del modelo anterior.



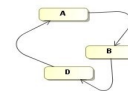


## Fichas CRC.

Las fichas CRC definen, para cada clase sus responsabilidades; extraídas de los diagramas de colaboración, de los casos de uso y del diagrama de entidades.

<Nombre de la clase>, (hereda de <Nombre de la clase>)	
Descripción :	Descripción de la clase. Breve explicación de su funcionalidad
Tipo :	Principal, Auxiliar, Interacción...
Características :	Concreta, composta
Responsabilidades :	Gestiona errors del gestor
Colaboraciones :	Con que otras clases mantiene relación
Constructores :	
+nombre (pText: String)	
Atributos :	
Lista de atributos.	
Métodos :	
Lista de métodos	





## Plantilla simplificada del diseño de software.

