Customer Journey Map:

Es una herramienta para trackear las emociones del usuario y mejorar asi sus experiencias.

Descubrimiento (Awareness): Cuando un **consumidor descubre el producto**. No se le incita a comprar, solo se le informa del producto que satisface una necesidad que puede tener.

Consideración (Consideration): Cuando el **consumidor quiere realizar una compra**. Elige entre varias opciones, hay que informarle sobre las características del producto, sus ventajas.

Compra (Purchase): Cuando el **usuario ya ha decidido su compra y la quiere llevar a cabo**. Tener un canal rápido, y un personal adecuado para mejorar la experiencia.

Retención (Retention): **Mantener la satisfacción del cliente**, y propiciar que repita compras y se fidelice.

Recomendación (Advocacy): Los **clientes ayudan a mejorar nuestra imagen**. Puede darse por las redes sociales, las valoraciones, y el boca a boca.

Construcción:

Eje X: Fases por las que pasa el cliente
Eje Y: Como siente las experiencias

Hay que indicar también 3 cosas:

- Puntos positivos
- Puntos negativos
- Puntos críticos

Construcción 2da parte:

Se definen las interacciones visibles y no visibles para los clientes.

Diseño:

Es el **proceso de definición** de:

- Arquitectura
- Componentes
- Interfaces
- Características del sistema / componente

Y el **resultado de ese proceso**.

Sistema: **Entidad lógica** con un conjunto de responsabilidades u objetivos. Consiste de software y/o hardware.

Subsistema: **Sistema parte de un sistema mayor**. Tiene una interfaz bien definida.

Componente: **Cualquier pieza de software o hardware con un rol claro**. Generalmente diseñados para ser reutilizables. Pueden ser reemplazados.

Módulo: **Componente definido a nivel de lenguaje de programación**. Package / Namespace.

Proceso de diseño de software:

- **Diseño arquitectónico**: Se define la arquitectura (En capas, MVC, monolítico, etc)
- **Especificación abstracta**: Se especifican los subsistemas. Cada subsistema realiza un servicio importante.
    - Contiene objetos altamente acoplados
    - Relativamente independiente de otros sistemas
    - Generalmente se descompone en módulos, pero también puede ser en subsistemas más pequeños.
- **Diseño de la interfaz**: Se describen las interfaces de los subsistemas. Se usa como caja negra.
- **Diseño de los componentes**: Cada subsistema se descompone en componentes, y se diseñan sus interfaces.

Principios del diseño:

- **Dividir y conquistar**: Dividir problemas grandes en más pequeños.
- **Incrementar Cohesión**: Mantener juntas las cosas relacionadas, y fuera el resto.
- **Reducir el acoplamiento**: Minimizar la interdependencia entre módulos.
- **Mantener la abstracción alta**: Ocultar o diferir los detalles. Reduce la complejidad.
- **Incrementar la reusabilidad**: Diseñar pensando en la reusabilidad en otros contextos. Reutilizar códigos y diseños.
- **Diseñe para ser flexible**: Anticipar y prepararse para cambios a futuro.
- **Anticipe la obsolescencia**: Planificar los cambios de tecnología y entorno para que el software siga ejecutándose cuando ocurran.

Cohesión: (Mantener alta)

Mantener unidas cosas relacionadas, y fuera el resto.

Objetivo: Diseñar servicios robustos, cuyos elementos esten fuertemente relacionados.

Ventajas: Favorece la comprensión y el cambio de los sistemas.

Acoplamiento: (Mantener bajo)

Es una medida de la interconexión entre módulos de una estructura de software.

Depende de:
- Complejidad de la conexión
- Punto de entrada / referencia a modulo
- Datos que se pasan a través de la interfaz

Se debe reducir al **eliminar o reducir relaciones innecesarias**. Hay que hacer que los **componentes sean lo más independientes** posible.

Dependencia entre componentes:

Ocurre cuando:
- **Referencias** de un componente a otro (Invocaciones)
- Datos pasados de un componente a otro (**parámetros**)
- **Grado de control** de un componente sobre otro (ej: composición)

2 Clases de Acoplamiento:

Fuertemente acoplados:
- Módulos usan variables compartidas
- Intercambian información de control.

Débilmente acoplados:
- Hacer que los detalles de datos estén dentro de un componente.
- Interfaz con otros componentes mediante lista de parámetros.
- Compartir información sólo aquellos componentes que la necesiten.
- No compartir información global.

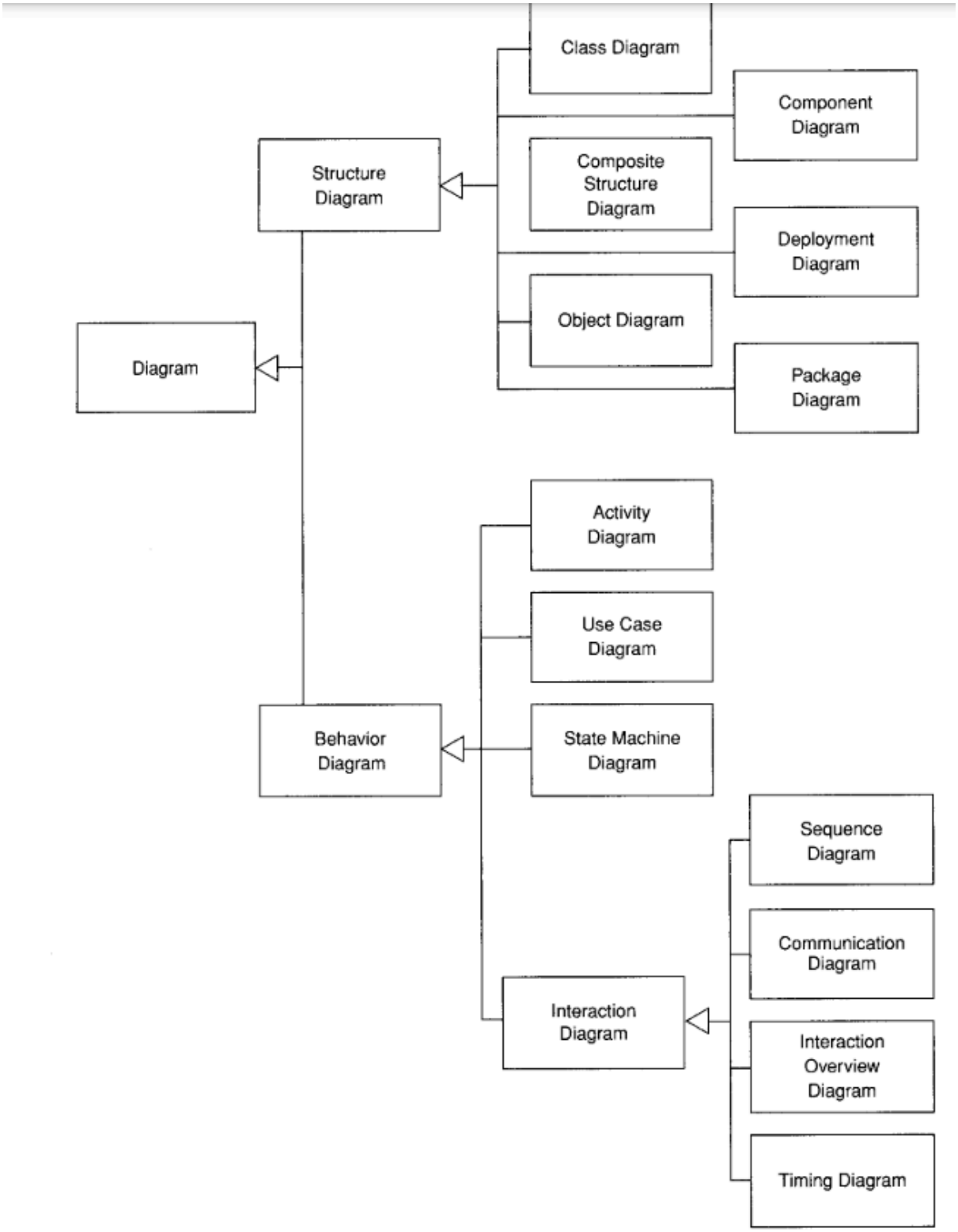
UML (Lenguaje Unificado de Modelado):

Familia de notaciones gráficas para describir o diseñar sistemas de software,en particular **orientado a objetos**.

Está **respaldado por la OMG**, un consorcio abierto de empresas que se formo para generar el estándar. **Aparicio en 1977 y unifico** muchos lenguajes de modelado grafico.

Diagramas UML:

Tipos de diagramas:



Diagramas de Caso de Uso:

Proporciona una **narrativa de como los actores usan el sistema**.

Representa:

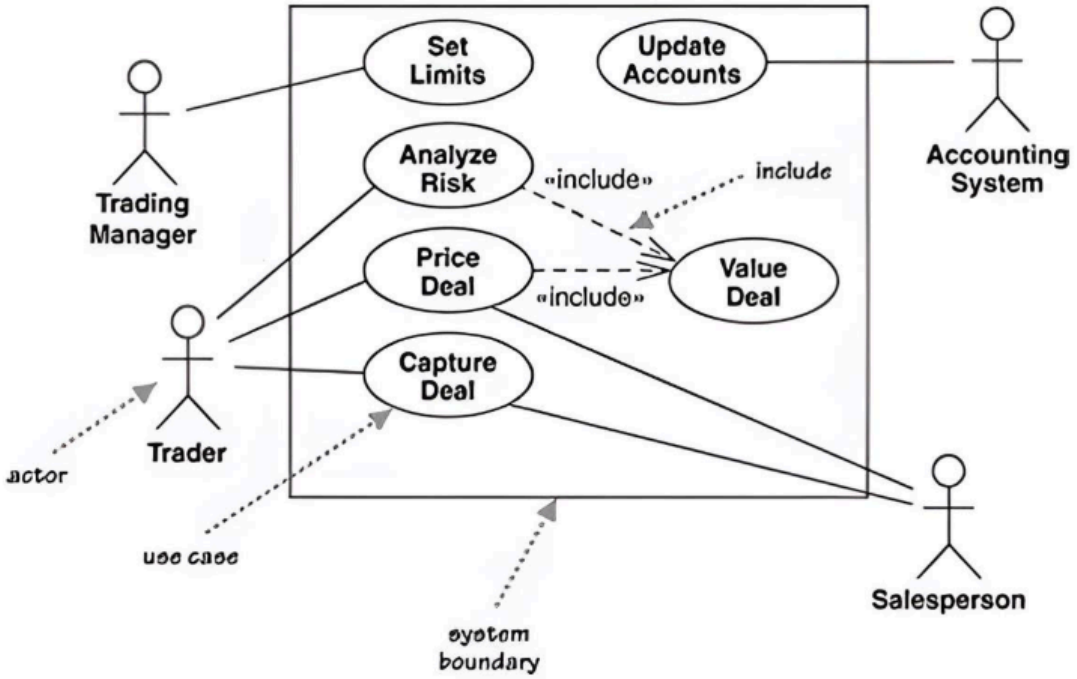
- **Requisitos funcionales**
- **Actores** que se comunican con el sistema. Usuarios y/o sistemas externos.
- **Relaciones entre requisitos funcionales y actores**.

Los diagramas de caso de uso **se pueden hacer a partir de un escenario**, es decir un conjunto de pasos que hable sobre como los actores interactuan con el sistema.

Hay que **concentrarse en la descripción textual, allí yace el valor** de los diagramas de caso de uso.

Tipos de relaciones entre subcasos de uso:

Includes: Si ocurre el caso de uso entonces también **ocurría el caso de uso relacionado**
Extends: Si ocurre el caso de uso entonces **PUEDE ocurrir en algún caso el caso de uso relacionado**.



Diagramas de Clase:

Describen los tipos de objetos así como sus relaciones estáticas. Incluyen **propiedades**, **operaciones** y **restricciones** aplicables a las conexiones entre objetos.

- **Pueden volverse incoherentes al expandirse y crecer**. Es mejor dividirlos en más pequeños.
- Crean una **visión general de alto nivel del sistema**
- Disminuir las líneas superpuestas
- Usar **colores** para agrupar módulos comunes.

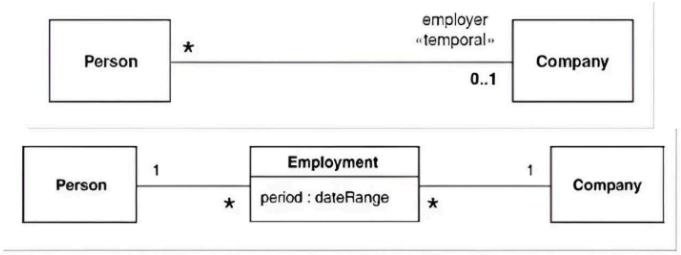
UML utiliza el termino **"feature" que abarca estado y comportamiento**.

Tipos de relaciones:

Generalización:
Similitudes entre clases se colocan en una clase general (super-tipo), **la relación entre esta clase y sus subtipos es la generalización**.

Asociaciones:
Es la más común y se utiliza **para representar dependencia semántica**. Se representa con una simple línea continua que une las clases que están incluidas en la asociación.

Clase de asociación:
Durante el proceso de diseño, puede surgir comportamiento u otros atributos que no tienen un responsable claro en la asociación.
En esos casos, surge la necesidad de tener **una nueva clase con la información de la asociación**.



Agregación:
Agregación se representa con rombo blanco. Refiere semánticamente a **"es parte de"**.
• La duda es; cuando es una agregación y cuando es una asociación. Martin nos responde:
– As Jim Rumbaugh says, "Think of it as a modeling placebo" [Rumbaugh, UML Reference]

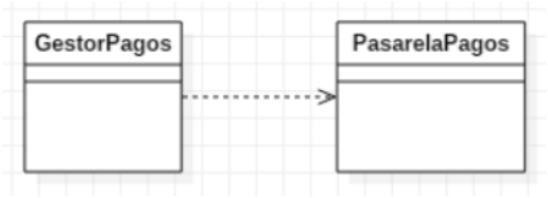
La agregación implica una **relación donde el hijo puede existir independientemente del padre**.

Composición:
Similar a la agregación, pero con una relación jerárquica más fuerte entre el objeto y las partes que lo componen.

Los elementos que forman parte no tienen sentido de existencia si no es dentro del elemento que los compone.

Tienen los mismos tiempos de vida, cuando el objeto padre muere todos los compuestos también.

Dependencia:
Se utiliza para reflejar relaciones donde **una clase requiere de otra para funcionar**, pero esta no forma parte del Estado



Atributos

• Visibilidad nombre: tipo multiplicidad = valor predeterminado {cadena de propiedades}
• La visibilidad indica si el atributo es **público (+) o privado (-), o restringido (#)**
• El nombre es el nombre del atributo.
• El tipo indica qué tipo de objeto puede almacenar el atributo,
• La **multiplicidad indica cuántos objetos se pueden almacenar en el atributo**.
• El valor predeterminado es el valor del atributo si no se especifica en la creación del objeto.
• Las **propiedades permiten indicar propiedades adicionales del atributo**. como por ejemplo si es de solo lectura. La omisión de la cadena de propiedades indica que el atributo es modificable.

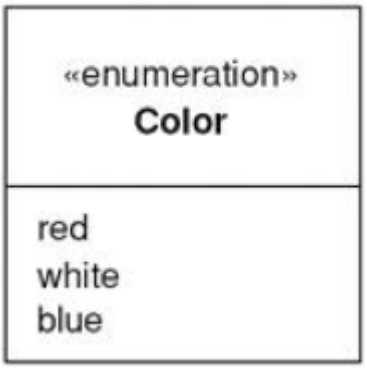
Multiplicidad

• La multiplicidad de una propiedad **indica cuántos objetos pueden ocupar esa propiedad**.
• Las multiplicidades más comunes son:
– **1**
– **0...1**
– ***
• En general, las **multiplicidades se definen con un límite inferior y un límite superior**.

• Los elementos en una multiplicidad forman un conjunto por defecto, pero s**e puede agregar {ordered} para indicar que el orden es importante y {non unique} para permitir duplicados**.

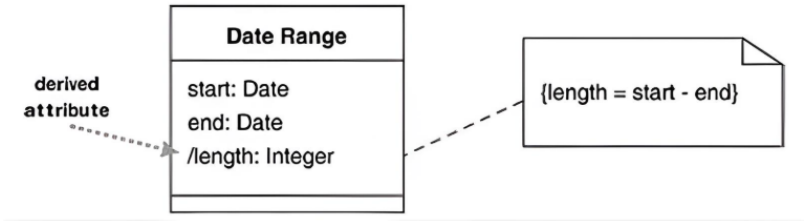
Enumeradores

• Set fijo de valores que no tienen comportamiento.
• En la práctica… Prefiero dejarlos “dinámicos"…



Propiedades derivadas

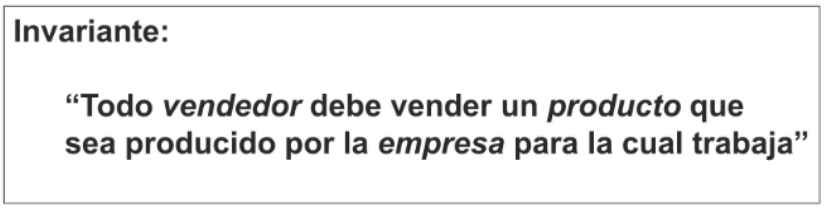
• Se calculan en base a otros atributos



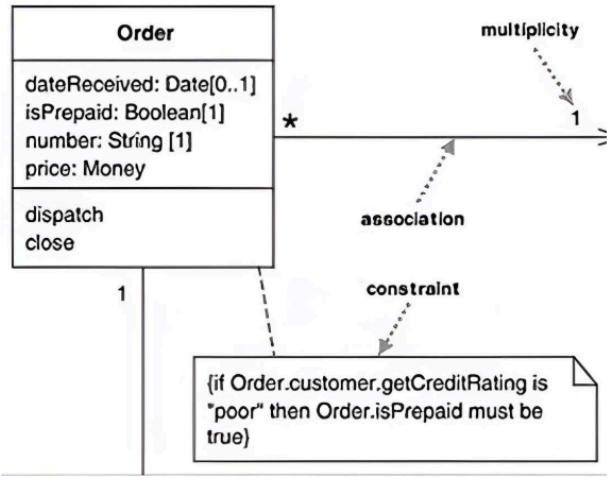
Restricciones

• Es muy común el hecho de que **un Modelo de Dominio no alcance a representar exactamente la realidad** planteada
• Existen casos donde un modelo representa fielmente la mayoría de los aspectos de la realidad sin embargo permite otros que no son deseables

Para aclarar las múltiples interpretaciones, podemos usar lo que llamamos invariantes



Se permiten expresar restricciones, utilizando los corchetes {}



Diagramas de Secuencia:

Representar el **intercambio de mensajes entre los distintos XXX del sistema** para cumplir con una funcionalidad.

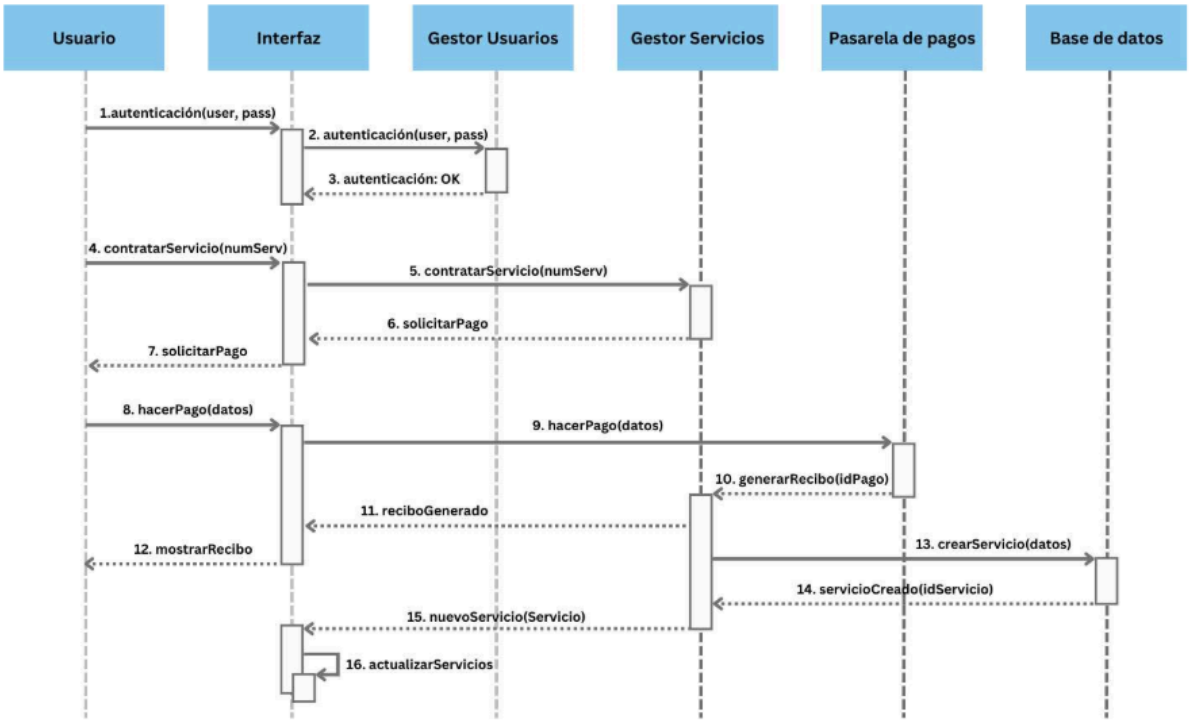
Pueden mejorar la comprensión del diagrama de clases, al representar cómo interactúan distintos objetos entre si.

Describen cómo colaboran grupos de objetos, normalmente los diagramas de secuencia **capturan el comportamiento de un solo escenario**.

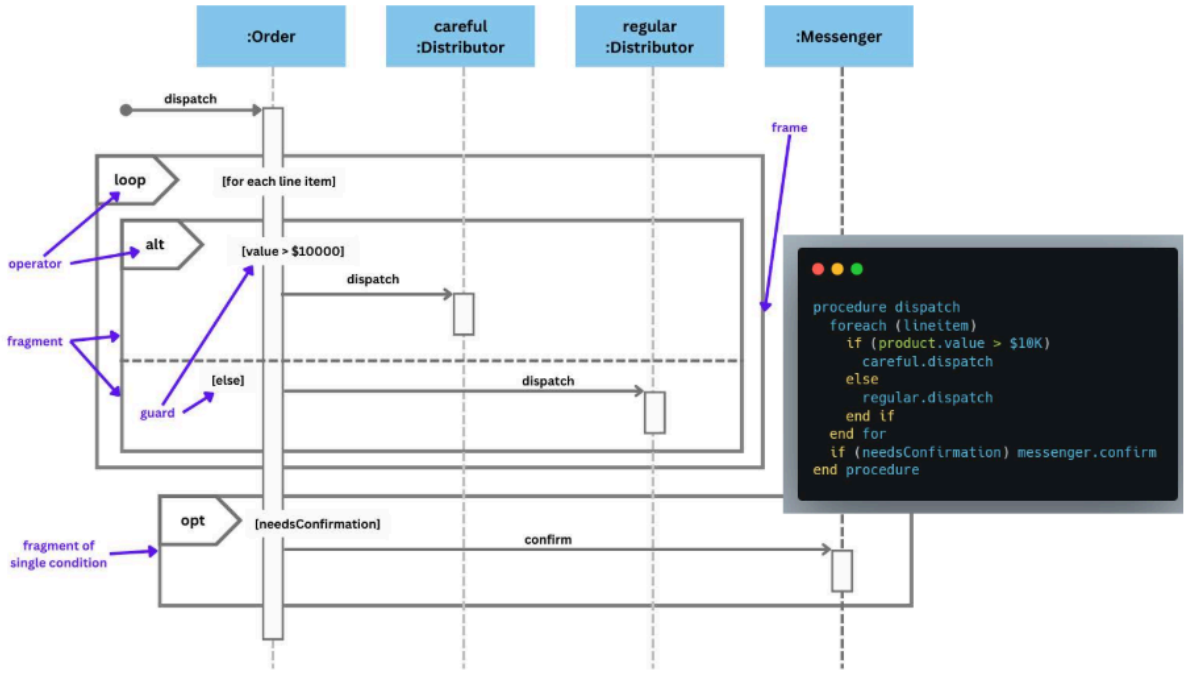
Dos dimensiones:
Horizontal: Representa los objetos que participan en la secuencia. (Componentes, sistemas, etc)
Vertical: Representa la linea de tiempo, mas arriba es menor tiempo, mas abajo es mayor.

NO se suele “reglar” la dimensión vertical para tener medidas especificas.

Las flechas entre los objetos son los mensajes.
Los mensajes pueden o no tener respuestas, y si estan son punteadas.



También se pueden representar:
LOOP: Bucles
ALT: Condición para que ocurra una cosa o la otra
OPT: Opcional



Diagramas de paquetes:

- Representan la **estructura más básica de un sistema**.
- Al crecer hay que buscar otras formas de visualizar las relaciones.
- Nos permiten **agrupar por algún concepto con un mayor nivel de abstracción**.

Diagrama de componentes:

Representa agrupaciones de módulos según algún criterio.

Documenta las relaciones entre el sistema a través de las interfaces.

Diagrama de Deploy:

Se usa junto a el diagrama de componentes (a veces incluso el de paquetes). Juntos dan una **visión general de cómo está desplegado el sistema**.

Se utiliza para representar cómo se **sitúan los componentes lógicos en los distintos nodos físicos**.

Un **nodo** es algo que puede alojar software, y puede ser de dos tipos:
- Un **dispositivo**, hardware (computadores, etc)
- Un **entorno de ejecución**, software (SO, etc)

Los nodos tienen **artefactos** que son **las manifestaciones físicas del software**, generalmente archivos.

Los **artefactos se pueden representar con cajas de clase, o listando su nombre en un nodo**.
Si se representa como caja de clase se le añade un nombre <<artefacto>>

Las **rutas de comunicación** entre los nodos representan cómo se comunican las cosas.
Pueden indicarse **protocolos**.

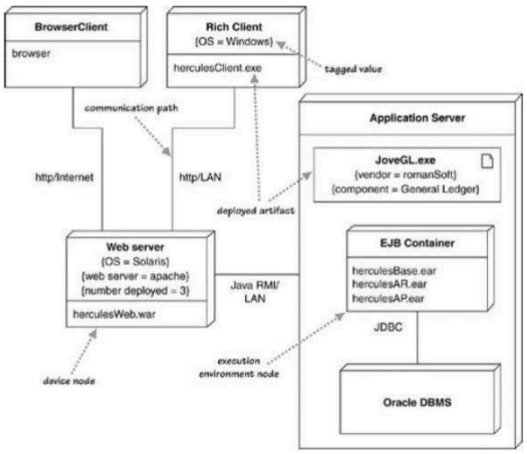


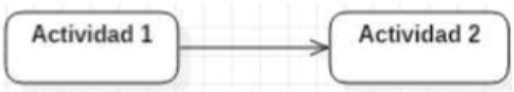
Diagrama de Actividad:

Modela el flujo de trabajo de un sistema. Representa procesos, procedimientos y algoritmos complejos visualmente sencillos.

No se tiene en cuenta los mensajes enviados.

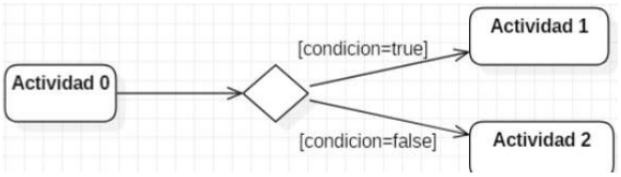
Ofrecen una visión a alto nivel. Se modelan las actividades, que se puede asemejar a los requisitos funcionales de negocio.

Elementos:
Flechas: Representar el flujo, de donde a donde se pasa.

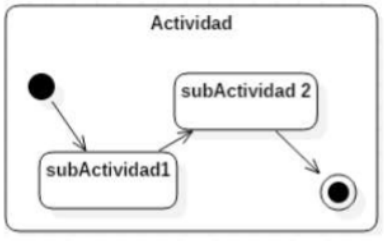


Notación de un flujo de actividad

Rombo: Representan decisiones, lleva a dos caminos (true or false)

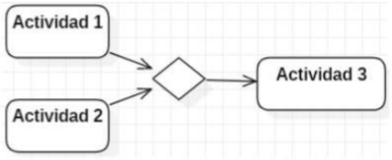


Circulo cerrado: Inicio del flujo
Dos circulos: Fin del flujo

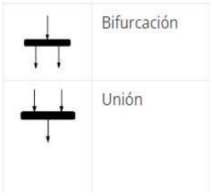


Notación de una actividad compuesta

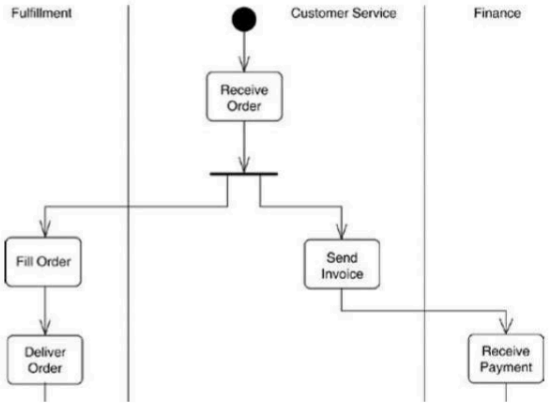
Nodo de fusión: Es un rombo que permite llegar a la misma actividad por dos o mas flujos.



Bifurcación: Indica el inicio de flujos en paralelo
Union: Representa el fin de flujos en paralelo (Todos los flujos deben terminar para darse)



Particiones: Para conocer “quien hace que”



Tiempo: Representa que una vez pase el tiempo indicado se desencadena un flujo

Recepción: Se indica que se recibe un evento externo
Envio: Se indica que se envía un mensaje / evento.