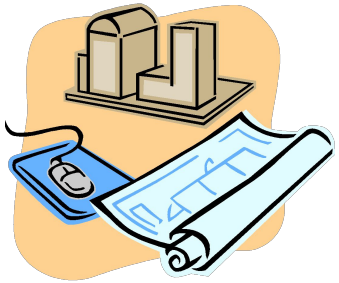
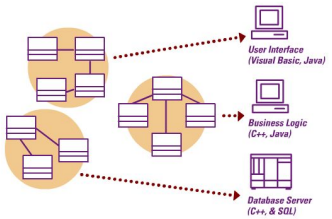
Vision General de UML

¿Qué es un Modelo?

Un modelo es una simplificación de la realidad

Es una **representación** a bajo costo de la **realidad.**





Construimos modelos para poder comprender mejor el sistema que estamos desarrollando...

Que es un modelo…

- Es una representación explícita y externa de parte de la realidad como la ven las personas que desean usar el modelo para entender, cambiar, gestionar y controlar dicha parte de la realidad. Pidd (2003)
- Lo más que se puede esperar de cualquier modelo es que puede suministrar una aproximación útil a la realidad. Box et al. (2005)

Que es un modelo…

- Un modelo es la abstraccion de un Sistema
- La abstraccion permite ocuparnos de detalles relevantes para un proposito
- Utilidad de modelado: abordar sistemas complejos

Que lenguaje emplear para modelar software?

- Codigo Fuente? X
- Lenguaje Natural? X

- Lenguaje Visual? OK
  - Facil de interpretar y procesar!!

Que lenguaje emplear para modelar software

```
package codemodel;
public class Guitarist extends Person implements MusicPlayer {
    Guitar favoriteGuitar;
     public Guitarist (String name) {super(name);}
            // A couple of local methods for accessing the class's properties
    public void setInstrument(Instrument instrument) {
           if (instrument instanceof Guitar) {
                        this.favoriteGuitar = (Guitar) instrument;
           } else {
                       System.out.println("I'm not playing that thing!");
            }
    }
    public Instrument getInstrument( ) {return this.favoriteGuitar;}
}
```

- Representa sólo la lógica e ignora el resto
- El ser humano lo interpreta muy lentamente
- No facilita la reutilización ni la comunicación

Que lenguaje emplear para modelar software

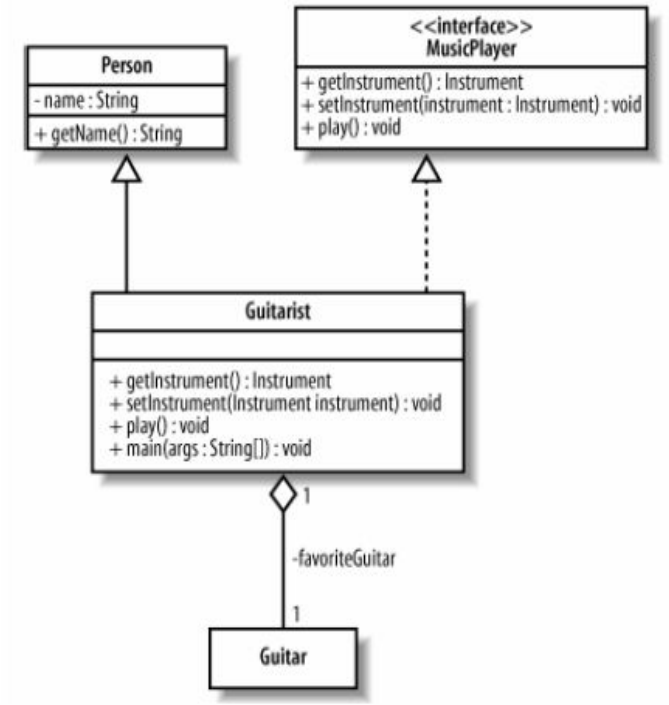
Guitarist is a class that contains six members: one static and five non-static. Guitarist uses, and so needs an instance of, Guitar; however, since this might be shared with other classes in its package, the Guitar instance variable, called favoriteGuitar, is declared as default.

Five of the members within Guitarist are methods. Four are not static. One of these methods is a constructor that takes one argument, and instances of String are called name, which removes the default constructor.

Three regular methods are then provided. The first is called setInstrument, and it takes one parameter, an instance of Instrument called instrument, and has no return type. The second is called getInstrument and it has no parameters, but its return type is Instrument. The final method is called play. The play method is actually enforced by the MusicPlayer interface that the Guitarist class implements. The play method takes no parameters, and its return type is void.

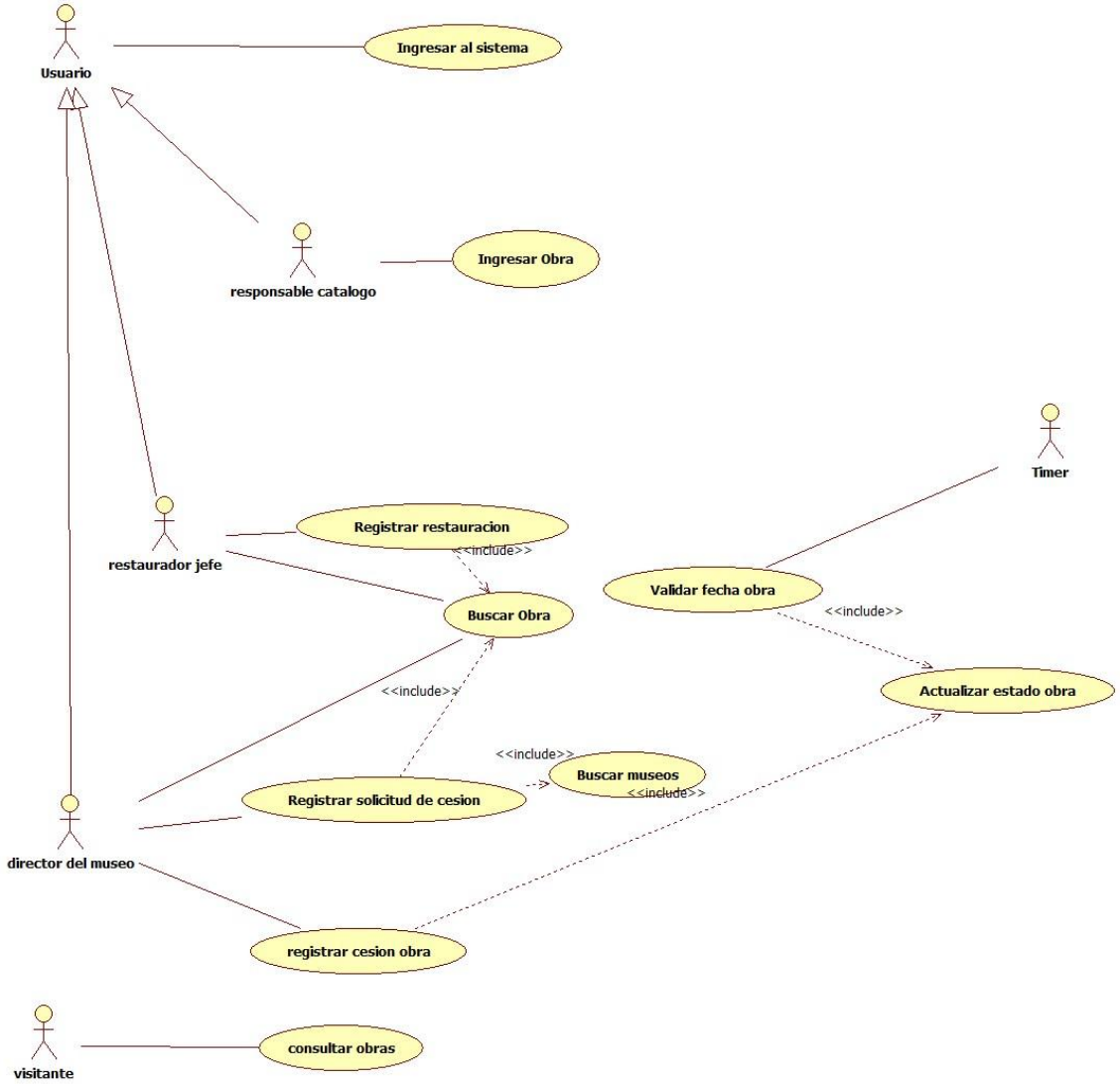
- Es ambigua y confusa
- Es lenta de interpretar
- Difícil de procesar

Que lenguaje emplear para modelar software



- No es ambigua ni confusa (una vez conocemos la semántica de cada elemento de modelado)
- Es fácil y rápida de interpretar
- Es fácil de procesar por herramientas

Ejemplo de un modelo



¿ Cuál es el objetivo de Modelar?

- Nos ayudan a visualizar un sistema como es, o como queremos que sea.
- Nos permiten especificar la estructura o el comportamiento de un sistema.
- Nos dan una plantilla que nos guía en la construcción del sistema.
- Documentan las decisiones que hemos tomado.
- Nos facilitan la comunicación con el cliente.

Principios de Modelado

- **La elección de qué modelos crear tiene una profunda influencia en cómo atacamos un problema y cómo delineamos una solución.**
  - Diferentes enfoques (analista estructurado, analista orientado a objeto o desarrollador de base de datos)
- **Cada modelo debe poder expresarse a diferentes niveles de precisión.**
- **Los mejores modelos tiene una conexión con la realidad.**
- **Un único modelo no es suficiente. Cualquier sistema es enfocado mejor con un conjunto de modelos independientes, pero relacionados.**

Aportes del Modelado

- Se facilita la comunicación entre el equipo al existir un lenguaje común.
- Se dispone de documentación que trasciende al proyecto.
- Hay estructuras que trascienden lo representable en un lenguaje de programación.

Conceptos del modelado

- Sistema software
  - Descripto por un conjunto de modelos
- Modelo
  - Simpllificacion para comprender major un sistema
- Diagrama
  - Representacion grafica de un modelo (UML usa grafos)
- Vista
  - Subconjunto de diagramas de un modelo que analiza un aspect concreto

Vistas

- En el contexto del software hay cinco vistas complementarias que son las mas importantes para visualizar, especificar, construir y documentar una arquitectura software
  - Vista de casos de uso
  - Vista de diseño
  - Vista de interacción
  - Vista de implementación
  - Vista de despliegue
- Cada una de estas vistas involucra modelado estructural y modelado de comportamiento.

Clases, Interfaces, Req. funcionales

Funcionalidades del sistema

Artefactos del sistema, archivo, código fuente

Vista de Diseño

Vista de Implementacion

Vista de casos de uso

Vista de interaccion

Vista de despliegue

Lógico

Fisico

Flujo de datos, control, req. no funcionales

Instalación y ejecución del sistema

Modelado Orientado a Objetos

- La visión tradicional toma una perspectiva algorítmica.
- La vision actual del desarrollo de software toma una perspectiva orientada a objetos.
  - El principal bloque es el objeto
    - Objeto es una cosa, del vocabulario del problema o de la solucion
    - Clase es un conjunto de objetos que son suficientemente similares para compartir una especificacion.
    - Todo objeto tiene una entidad puede nombrarse (y de esta forma distinguirse de otros objetos), un estado y comportamiento
- Visualizar, especificar, construir y documentar sistemas orientados a objetos es el proposito de UML.

Ingeniería Dirigida por Modelos

- Ingeniería dirigida por modelos (MDE) es un enfoque para el desarrollo de software donde los modelos en lugar de los programas son los principales resultados del proceso de desarrollo.
- Los programas que se ejecutan en una plataforma de hardware / software se generan automáticamente a partir de los modelos.
- Los defensores de la MDE sostienen que esto eleva el nivel de abstracción en la ingeniería de software para que los ingenieros ya no tienen que preocuparse por los detalles del lenguaje de programación o las características específicas de plataformas de ejecución.

¿Qué es UML?

Es un lenguaje de modelado, de propósito general, usado para la visualización, especificación, construcción y documentación de sistemas Orientados a Objetos

¿Qué es UML?

- UML = Unified Modeling Language
- Un lenguaje de propósito general para el modelado orientado a objetos. Impulsado por el Object Management Group (OMG, Object Management Group, www.omg.org)
- Documento “OMG Unified Modeling Language Specification”
- UML combina notaciones provenientes desde:
  - Modelado Orientado a Objetos
  - Modelado de Datos
  - Modelado de Componentes
  - Modelado de Flujos de Trabajo (Workflows)

¿Por qué UML es un lenguaje?

Lenguaje = Notación + Reglas

- Sintáctica
- Semántica
- Pragmática

¿Por qué UML es un lenguaje?

- Un lenguaje provee un vocabulario y reglas para combinar los elementos del vocabulario con el propósito de comunicar.
- En un lenguaje de modelado ese vocabulario y reglas se focalizan en representaciones conceptuales y físicas de un sistema.

¿Qué NO ES UML?

UML **NO** es:

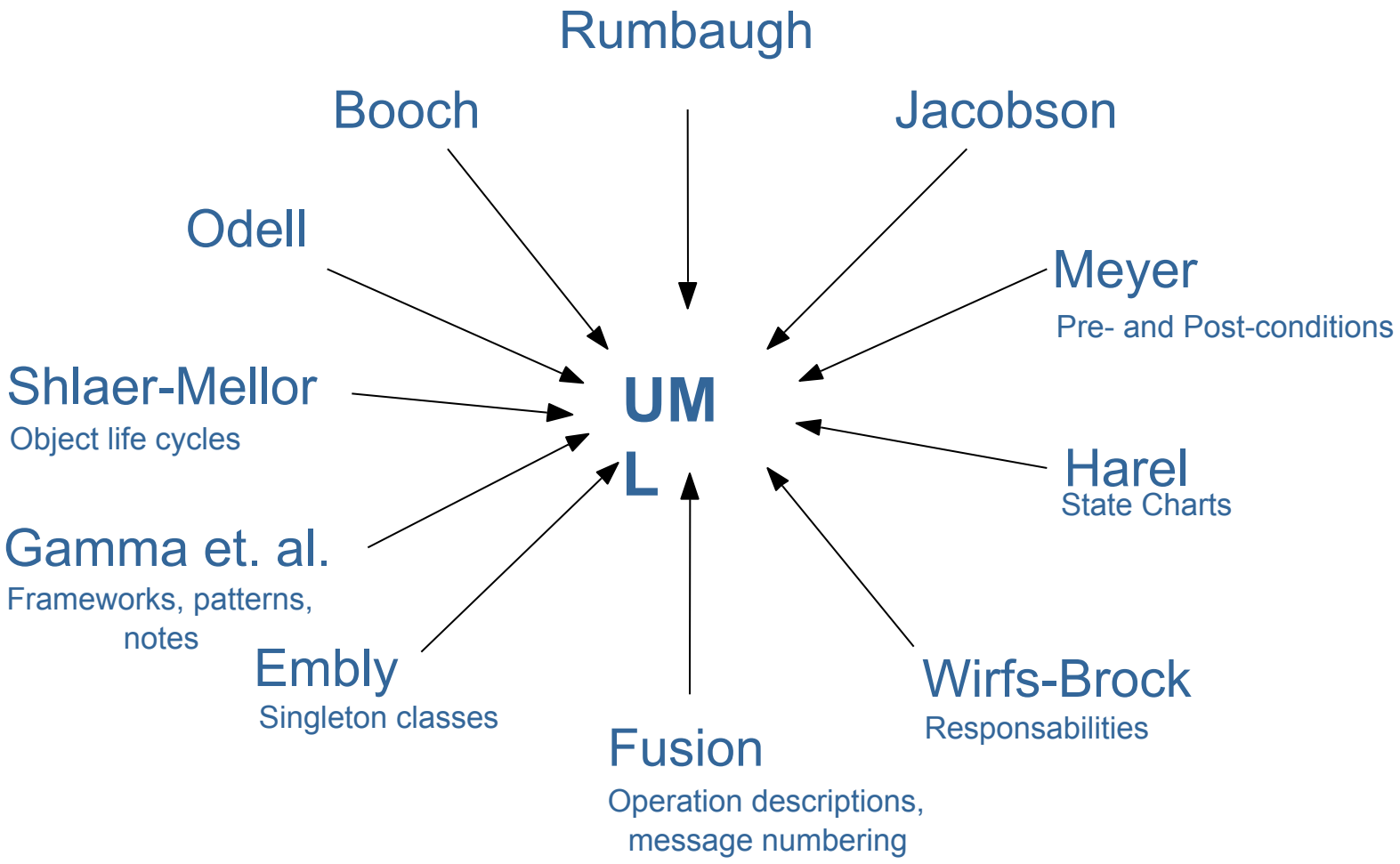
- Un lenguaje de programación visual.
- La especificación de una herramienta o un repositorio.
- **UNA METODOLOGÍA, MÉTODO O PROCESO.**

Situación de Partida

- **Diversos métodos y técnicas OO, con muchos aspectos en común pero utilizando distintas notaciones**
- **Inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas, etc.**
- **Pugna entre distintos enfoques (y sus correspondientes gurús)**

Establecer una notación estándar

UML "aglutina" enfoques OO



Un lenguaje para documentación...

Contempla a todos los “artefactos” que se producen en el desarrollo de Software, entre ellos:

- Requerimientos
- Arquitectura
- Diseño
- Código
- Planes de Proyecto
- Pruebas
- Prototipos

Principales Elementos del Lenguaje

- **Bloques de construcción:**
  - Elementos: componente de un modelo, es una abstracción hecha del sistema que se esta modelando.
  - Relaciones: Ligan los elementos entre si
  - Diagramas: Agrupan colecciones interesantes de elementos y relaciones.
- **Reglas:**
  - Relacionan los bloques de construcción
- **Mecanismo comunes:**
  - Permiten aplicar estas reglas a través de UML

Bloques de Construcción UML

Bloques de Construcción UML	Elementos	Caso de Uso	Estructurales
		Clase	
		Clase Activa	
		Interfaz	
		Componente	
		Colaboración	
		Nodo	
		Interacción	De Comportamiento
		Máquina de Estados	
		Paquete	De Agrupación
		Modelo	De Anotación
		SubSistema	
		Nota	
	Relaciones	Dependencia	
		Asociación	
		Generalización	
	Diagramas	Clases	Estructurales
		Objetos	
		Componentes	
		Despliegue	
		Caso de Usos	De Comportamiento
		Secuencia	
		Colaboración	
		Estados	
		Actividades	

Beneficios del Modelado con Casos de Uso

Cliente	• Ayuda a determinar el alcance general del sistema. • Provee una validación de los requerimientos del cliente. • Ayuda en la estimación de la programación y la presupuestación del sistema. • Actúa como base para las pruebas de aceptación.
Usuario	• Provee los requerimientos del usuario para su validación. • Modela la interacción del usuario con el sistema.
Líder de Proyecto	• Ayuda en la estimación de programación y presupuestación. • Ayuda en la evaluación de la factibilidad y el riesgo del proyecto. • Ayuda en la rastreabilidad de los requerimientos. • Ayuda en la registración del progreso del sistema.
Arquitecto	• Delinea la arquitectura del sistema • Ayuda a rastrear requerimientos arquitectónicos. • Ayuda a evaluar la completitud, consistencia y coherencia de la arquitectura.
Desarrollador	• Provee modelos de los requerimientos para el diseño del sistema. • Es un medio para documentar el sistema.

Modelado de Requerimientos Funcionales con Casos de Uso

Bloques de Construcción de UML

Diagrama de Casos de Uso

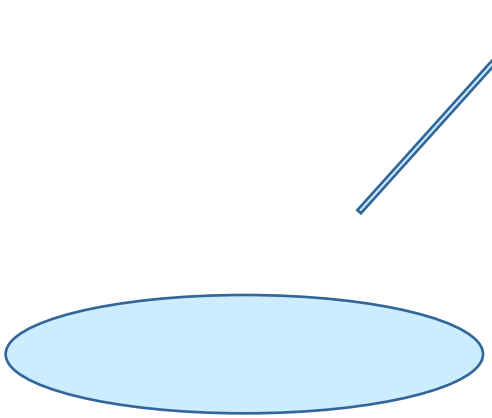
Muestra un conjunto de Casos de Uso, actores y sus relaciones.

Permite establecer el comportamiento deseado del sistema. Especifica los requerimientos funcionales del Software.

Contiene comúnmente:

- Casos de Uso
- Actores
- Relaciones de extensión, inclusión, generalización y/o dependencia.

Elementos que intervienen: Casos de Uso



El conjunto de todos los **Casos de Uso**, debe cubrir los requerimientos del Sistema en su totalidad.

Se pueden definir Caso de Uso en diferentes niveles:

- **A nivel de sistema de Negocio**
- **A nivel de sistema de Software**

Las descripciones de los Casos de Uso son cruciales para la comprensión del sistema

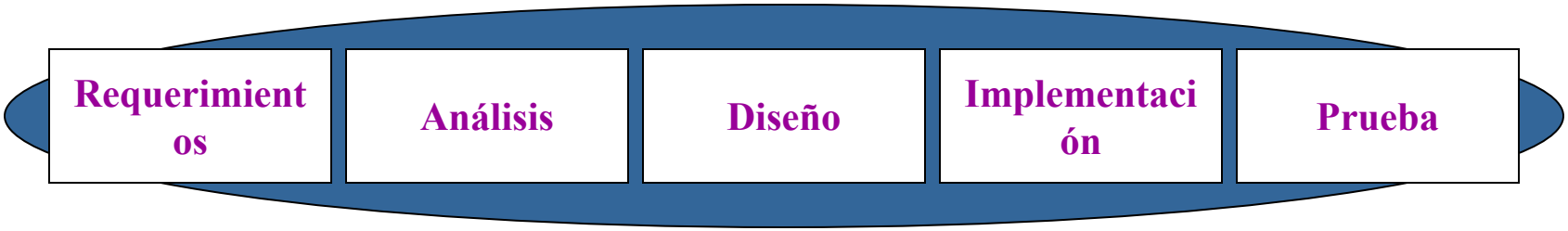
Propiedades:

- Captura alguna función visible para el usuario.
- Puede ser grande o pequeño.
- Debe alcanzar un objetivo específico para el actor.

Elementos que intervienen: Casos de Uso

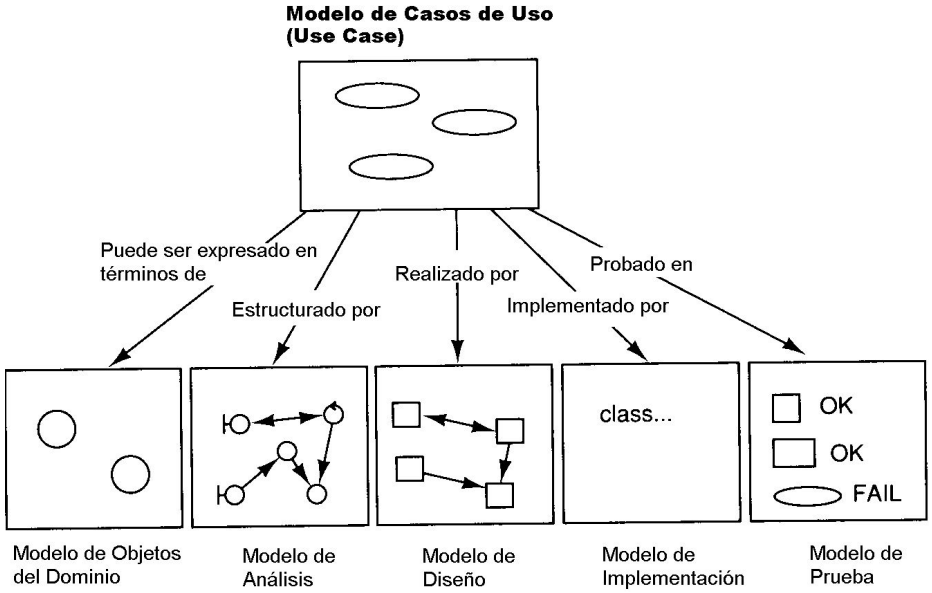
- Los Casos de Uso se determinan observando y precisando, actor por actor, las secuencias de interacción, los escenarios, desde el punto de vista del usuario
- Un escenario es una instancia de un caso de uso
- Los casos de uso intervienen durante todo el ciclo de vida. El proceso de desarrollo estará dirigido por los casos de uso

Proceso Conducido por Casos de Uso . . .



- Captura el Valor de los Requerimientos
- Conduce el Proceso
- Delinea la arquitectura

El Modelo de Caso de Uso controla la formación de todos los otros modelos



Trabajando así el modelo del sistema será conducido por Casos de Uso:

- Si se desea cambiar el comportamiento del Sistema, remodelamos el actor y el Caso de Uso apropiado.
- Como tenemos **rastreabilidad** se podrá modificar el sistema desde nuevos requerimientos.

¿Cómo encontrar Casos de Uso?

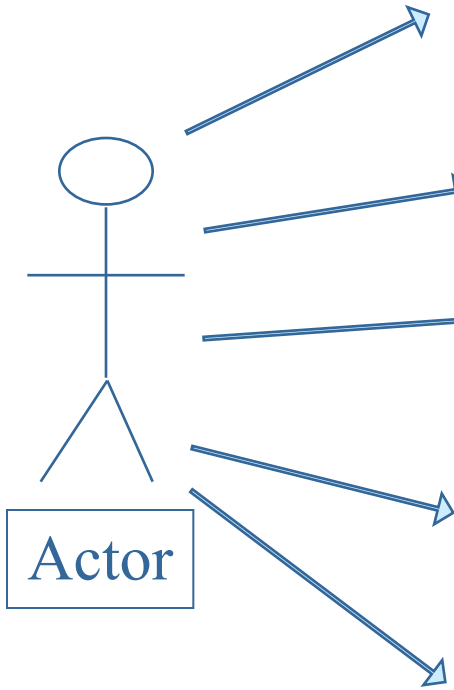
- Para cada actor identificado: ¿cuáles son las tareas en las cuales el sistema debería estar involucrado?
- ¿Necesita el actor ser informado a cerca de ciertas ocurrencias en el sistema?
- ¿Necesita el actor informar a cerca de cambios externos, repentinos?
- ¿Provee el sistema al negocio con el comportamiento correcto?
- ¿Pueden ejecutarse todos los aspectos por los Caso de Uso que se han identificado?
- ¿Qué Casos de Uso soportarán y mantendrán el sistema?
- ¿Qué información debe ser modificada o creada en el sistema?

¿Cómo encontrar Casos de Uso?

Los Casos de Uso de soporte, que no representan lo que comúnmente son las funciones principales del sistema, pueden ser de las siguientes clases:

- Inicio y finalización del sistema
- Mantenimiento del Sistema. Por ejemplo: agregar nuevo usuarios, definir perfiles de usuarios.
- Mantenimiento de los datos almacenados en el sistema, ejemplo: el sistema debe trabajar en paralelo con un sistema legado y los datos necesitan sincronizarse entre los dos.
- Funcionalidad necesaria para modificar el comportamiento del sistema.

Elementos que intervienen: Actores



Representa un rol que interactúa con el sistema, puede ser un usuario humano u otro sistema o dispositivo de hardware. El nombre del actor debe describir el papel desempeñado.

Como simboliza el ambiente del sistema no lo describimos en forma detallada.

Una persona puede ejecutar distintos **roles** en el sistema

Hay actores principales: son los que usan el sistema **directamente;** para quienes desarrollamos el sistema.

Hay actores secundarios: son aquellos de los que el sistema necesita ayuda para poder cumplir con el objetivo del Caso de Uso. Son aquellos que mantienen o administran el sistema.

¿Cómo encontrar Actores?

- ¿Quién o qué inicia eventos con el sistema?
- ¿Quién proveerá, usará o quitará información?
- ¿Quién usará esta funcionalidad?
- ¿Quién está interesado en cierto requerimiento?
- ¿En que parte de la organización será usado el sistema?
- ¿Quién dará soporte y mantendrá el sistema?
- ¿Cuales son los recursos externos del sistema?
- ¿Qué otros sistemas necesitarán interactuar con este sistema?

Utilidad de identificar actores de esta forma

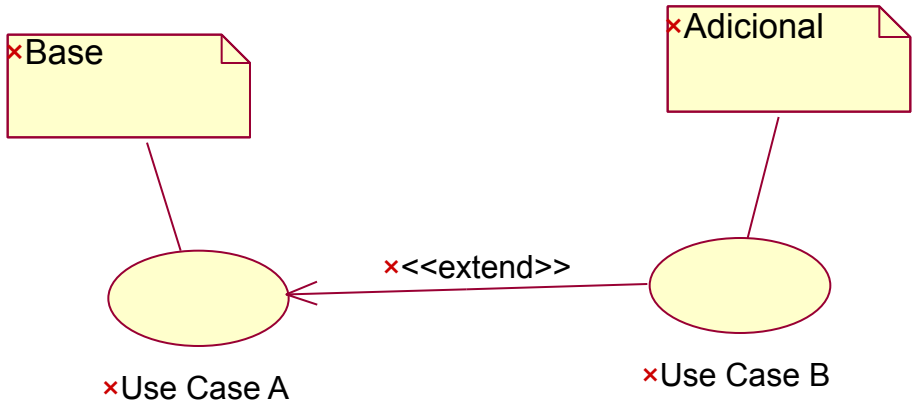
- Porque la estructura del sistema debe decidirse desde el punto de vista de la funcionalidad principal. Es decir, los usuarios deciden la funcionalidad del sistema.

- Porque se garantiza que el sistema se adaptará a los actores más importantes.

- Los actores facilitan la identificación de funciones, determinando por ejemplo:
  - ¿Cuáles son las tareas principales de cada usuario?
  - ¿Tendrá el actor que informar al sistema sobre cambios exteriores?
  - ¿Desea el usuario ser informado sobre cambios inesperados?

Elementos que intervienen: Relaciones

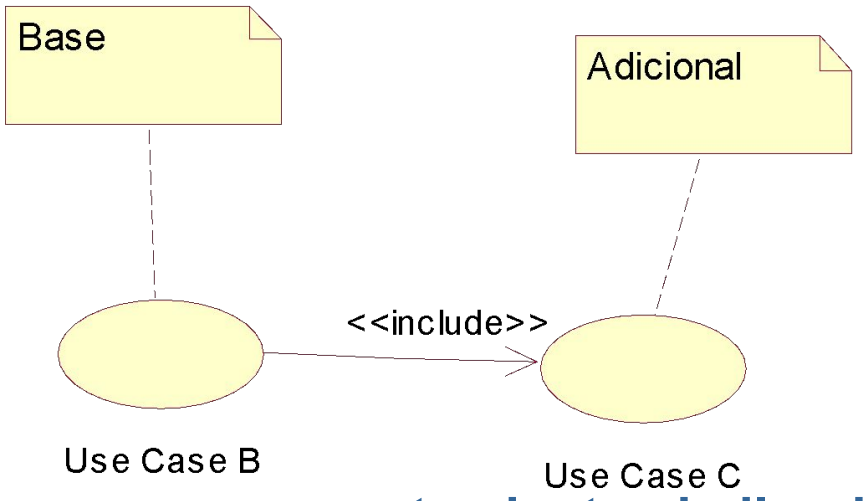
Asociaciones de Extensión



- Especifica como un Caso de Uso puede insertarse y así extender la funcionalidad de otro.
- El Caso de Uso donde se insertará la extensión debe ser un curso completo en sí mismo.
- Se usan para modelar partes optativas, alternativas, etc.
- Se dibuja con una flecha cuya dirección va desde el Caso de Uso de extensión (adicional) al Caso de Uso base.

Elementos que intervienen: Relaciones

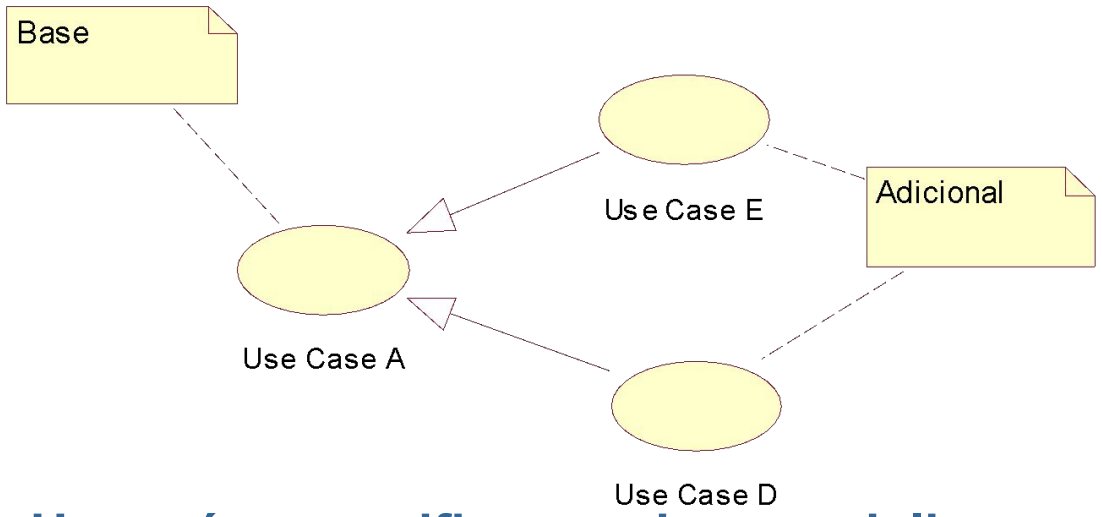
Asociaciones de Inclusión



- Especifica y agrupa comportamiento similar de varios Casos de Uso, es un Caso de Uso abstracto, que otros podrán usar.
- Se usan cuando su intervención es necesaria para completar un curso completo de eventos.
- Una instancia del Caso de Uso origen incluye también el comportamiento descrito por el Caso de Uso destino.
- Se dibuja con una flecha desde el Caso de Uso concreto o base al use case abstracto (adicional).

Elementos que intervienen: Relaciones

Asociaciones de Generalización



- Un Caso de Uso más especifico puede especializar a un Caso de Uso más general.
- Una relación de generalización entre Casos de Uso implica que el caso de uso hijo contiene todos los atributos y secuencias de comportamiento y puntos de extensión definidos para el padre.
- Se dibuja con una flecha desde el Caso de Uso hijo al padre.
- Los Casos de Uso hijos pueden redefinir el comportamiento heredado del padre.

Un ejemplo...

¿Cuál es la utilidad de la técnica?

- **Identificar requerimientos:** organizar la forma en la que se lleva a cabo el relevamiento con los usuarios (qué preguntar y cuando)
- **Analizar requerimientos** a partir de las primeras funciones se puede organizar la información y buscar formas de profundizarla.
- **Especificar requerimientos** complementados con descripciones en lenguaje natural o con diagramas de actividad.

Especificación de Casos de Uso

- Plantilla de Descripción de Casos de Uso a Trazo Fino
- Plantilla de Descripción de Casos de Uso a Trazo Grueso

Especificación de Casos de Uso

Lineamientos para el contenido del flujo de eventos

- Describir como inicia y termina el Caso de Uso
- Describir que datos se intercambian entre el actor y el Caso de Uso.
- No describir detalles de la interfaz del usuario, a menos que sea necesario para entender el comportamiento del sistema.
- Describir el flujo de eventos, no solo la funcionalidad, para reforzar esto comenzar cada acción con: “Cuando el actor...”
- Describir solo los eventos que pertenecen a ese Caso de Uso, y no lo que pasa en otros Casos de Uso o fuera del sistema.
- Evitar terminología vaga tal como “por ejemplo” “etc” “información”.
- Detalle en el flujo de eventos todos los “que” que deberían responderse, recuerde que los diseñadores de pruebas usarán ese texto para identificar casos de prueba.

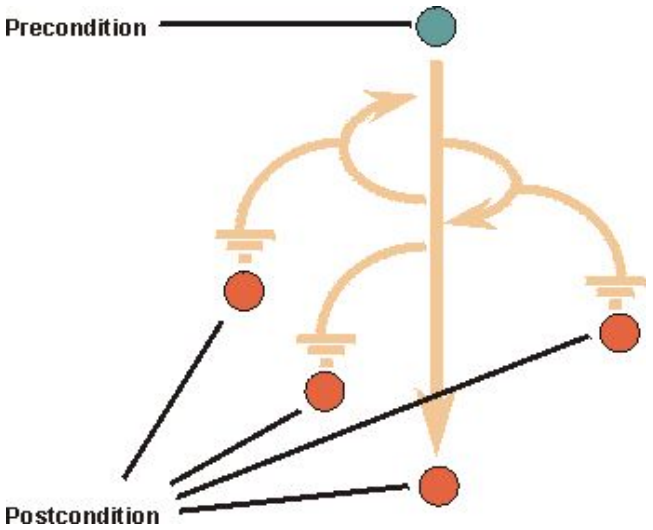
Especificación de Casos de Uso

Pre y Post Condiciones

- Una pre-condición es una **restricción sobre cuando un Caso de Uso puede empezar**. **No es el evento que inicia** el Caso de Uso.

- Una pre-condición de un Caso de Uso, no es una pre-condición para un único subflujo, aunque se pueda definir pre y post condiciones a nivel de subflujo.

- Una post-condición para un Caso de Uso debe ser verdadera, independientemente de cual flujo sea ejecutado. Si algo puede fallar, debería cubrirse en la post condición diciendo: " La acción se ha completado o si algo ha fallado, la acción no se ha realizado", en lugar de decir "La acción se ha completado".



¿Qué nivel de detalle adoptar?

- Si el desarrollo es incremental:
  - Identificar todos los requerimientos que se pueda.
  - Definir Prioridades.
  - Seleccionar cuáles requerimientos se implementarán en cada versión.
- Como plantear las descripciones
  - Se identifican las funciones en forma general, “de trazo grueso”
    - Se incluyen escenarios operacionales más relevantes, no entrando en detalles sobre acciones que realiza el sistema.
  - Luego se especifican las funciones en forma detallada, “de trazo fino”
    - Se completan los detalles.
    - Se incluyen las alternativas, especificando en particular errores o especificaciones que provienen de requerimientos de los usuarios.