3.1 Análisis orientado a objetos

Indice

- Introducción
- Modelos de ciclo de vida
- Análisis orientado a objetos
- Modelado de software
- Modelado visual
- UML
- Elementos de modelado
- Diagramas de UML
- Vistas arquitecturales

Introducción

- Un proyecto de desarrollo de software no consiste sólo en programar
- Se necesita conocer las necesidades del cliente
  - Obtener requisitos, analizarlos, validarlos…
- Es necesario diseñar una solución y hacer los "planos del software"
- Se debe asegurar que el software funciona
  - Pruebas unitarias, pruebas de integración…
- Se debe mantener el software
  - Documentación de cada fase

Introducción

- ¿Qué estrategia seguimos para organizar las fases de un proyecto?
- Un modelo de ciclo de vida nos guía en las fases que hay que realizar, sus entradas y salidas, los criterios de transición, las actividades a realizar…
- La elección de uno u otro depende del tipo de proyecto
- Con tecnologías orientadas a objetos se tiende a ciclos de vida iterativos e incrementales

Modelos de ciclo de vida

Existen distintos modelos de ciclo de vida:

- Modelos lineales
- Modelos iterativos
- Modelos incrementales
- Modelos basados en prototipos
- …

Modelos lineales o secuenciales

- Características:
  - Facilitan el control del progreso a los gestores
  - Enfoque sistemático y secuencial
  - Fases separadas en la especificación y desarrollo
- Filosofía poco realista:
  - No se ajusta al proceso de desarrollo de software
  - Generalmente el flujo no es secuencial, sino que aparecen diversas iteraciones
  - Para técnicas de desarrollo basadas en objetos y componentes no es el soporte más adecuado

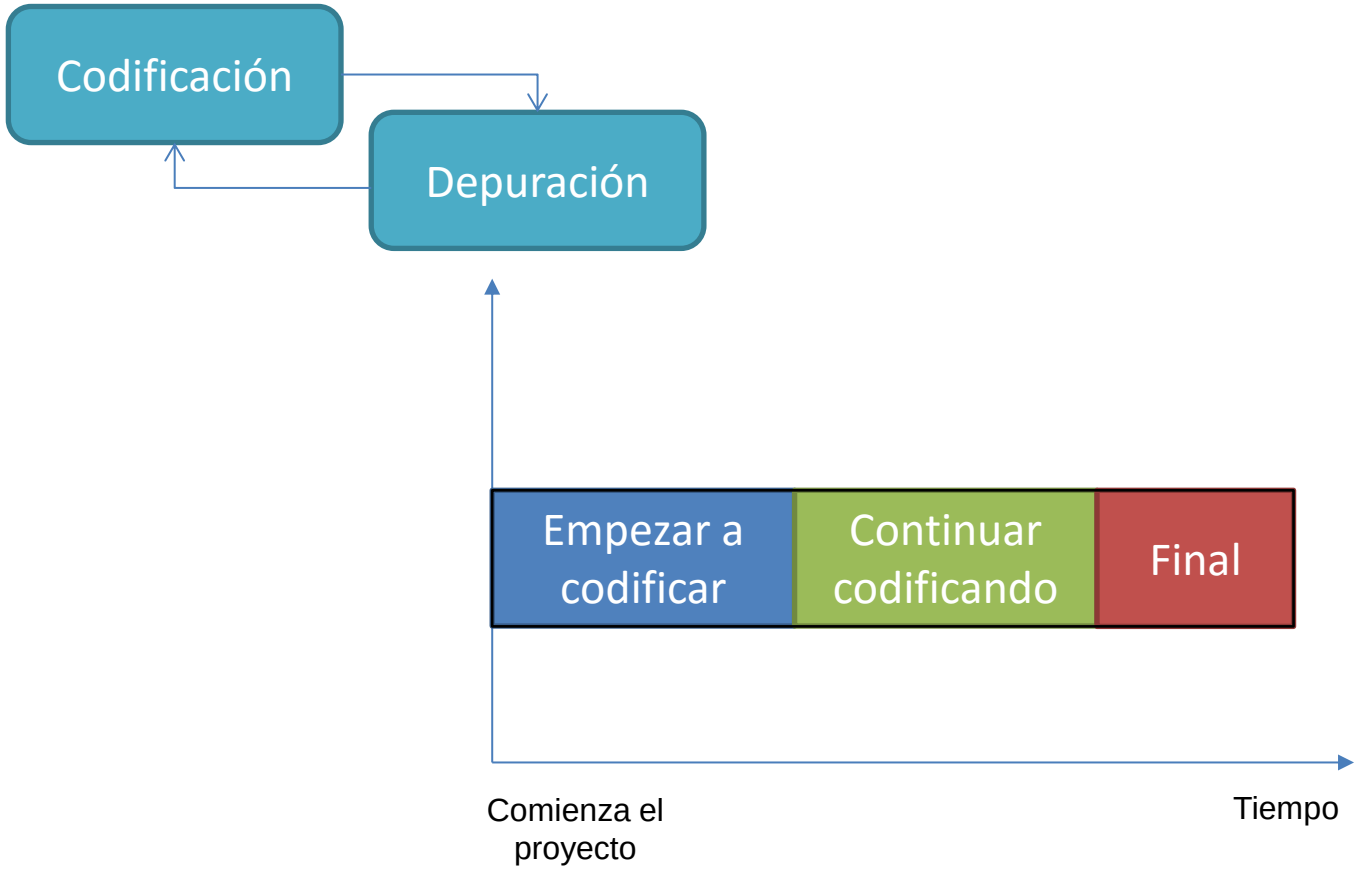
Modelos lineales o secuenciales

Modelo primitivo

- Conocido también como Modelo Prueba y Error o Modelo Codifica y Mejora
- Modelo aplicado en las primeras experiencias de programación
- No existe planificación ni diseños previos
- Se realizan una serie de iteraciones formadas por fases de codificación-depuración

Modelos lineales o secuenciales

Modelo primitivo



Modelos lineales o secuenciales

Modelo primitivo

- Inconvenientes
  - Código mal estructurado tras varias iteraciones
    - Código espagueti
  - Costes elevados debido a las numerosas recodificaciones
  - Posible rechazo en la entrega, al no existir análisis de requisitos previo
  - Costes elevados de depuración por la falta de planificación
  - Costes elevados de mantenimiento por la falta de estructura y documentación

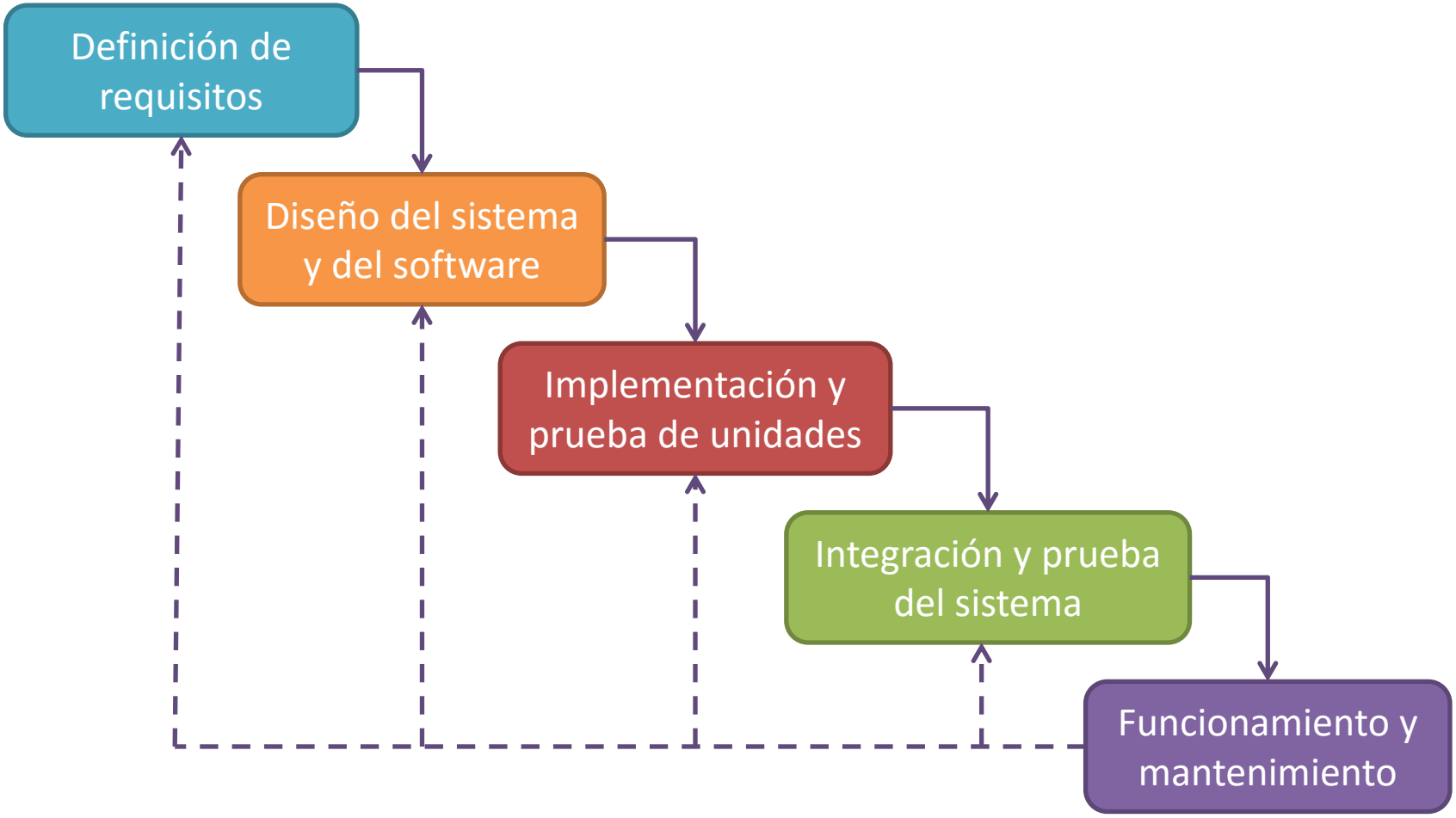
Modelos lineales o secuenciales

Modelo clásico o en cascada

- Enunciado por Royce (1970)
- Se denomina en cascada porque las etapas se suceden de forma lineal como cayendo en cascada de una a otra
- Una etapa no puede empezar hasta que no ha acabado la anterior
- En cada etapa se genera todo un conjunto de documentos (Es un modelo dirigido por documentos)

Modelos lineales o secuenciales

Modelo clásico o en cascada



[Sommerville, 2005]

Modelos lineales o secuenciales

Modelo clásico o en cascada

Análisis y definición de requisitos:

- Los servicios, restricciones y metas del sistema se definen a partir de las consultas con los usuarios.
- Se definen con mucho detalle y sirven como una especificación del sistema

Diseño del sistema y del software

- Permite describir cómo se van a satisfacer los requisitos
- Se dividen los requisitos en sistemas hardware o software.
- Se establece la arquitectura completa del sistema.

Modelos lineales o secuenciales

Modelo clásico o en cascada

Implementación y prueba de unidades:

- El diseño del software se lleva a cabo como un conjunto o unidades de programas
- La prueba de unidades implica que cada una cumpla su especificación

Integración y prueba del sistema

- Los programas o unidades se integran y prueban como un sistema completo
- Tras las pruebas el sistema se entrega al cliente

Funcionamiento y mantenimiento:

- Por lo general, es la fase más larga
- Instalación del sistema y corrección de errores no descubiertos en etapas anteriores
- Mejoras de las unidades del sistema

Modelos lineales o secuenciales

Modelo clásico o en cascada

- El modelo en cascada original preveía “ciclos de retroalimentación”
- Sin embargo, la mayoría de organizaciones que aplica este modelo de proceso lo trata como si fuera estrictamente lineal
- Problemas
  - Es difícil para el cliente establecer todos los requisitos de manera explícita al principio del proyecto
  - El cliente debe tener paciencia. Una versión funcional sólo estará disponible cuando el proyecto esté muy avanzado
  - Poca o nula flexibilidad a cambios

Modelos lineales o secuenciales

Modelo clásico o en cascada

- ¿Cuándo es útil?
  - Para proyectos complejos pero que se entienden y quedan bien definidos desde el comienzo
  - Cuando realizamos una migración de software desde un entorno tecnológico obsoleto

Modelos evolutivos

Modelo incremental

- El sistema definido se divide en subsistemas de acuerdo a su funcionalidad
- Las versiones se definen comenzando con un subsistema funcional pequeño y agregando funcionalidad con cada nueva versión
  - Cada nueva parte entregada se denomina incremento
- Combina elementos del modelo en cascada con la construcción de prototipos
- Se entrega un producto operativo en cada incremento

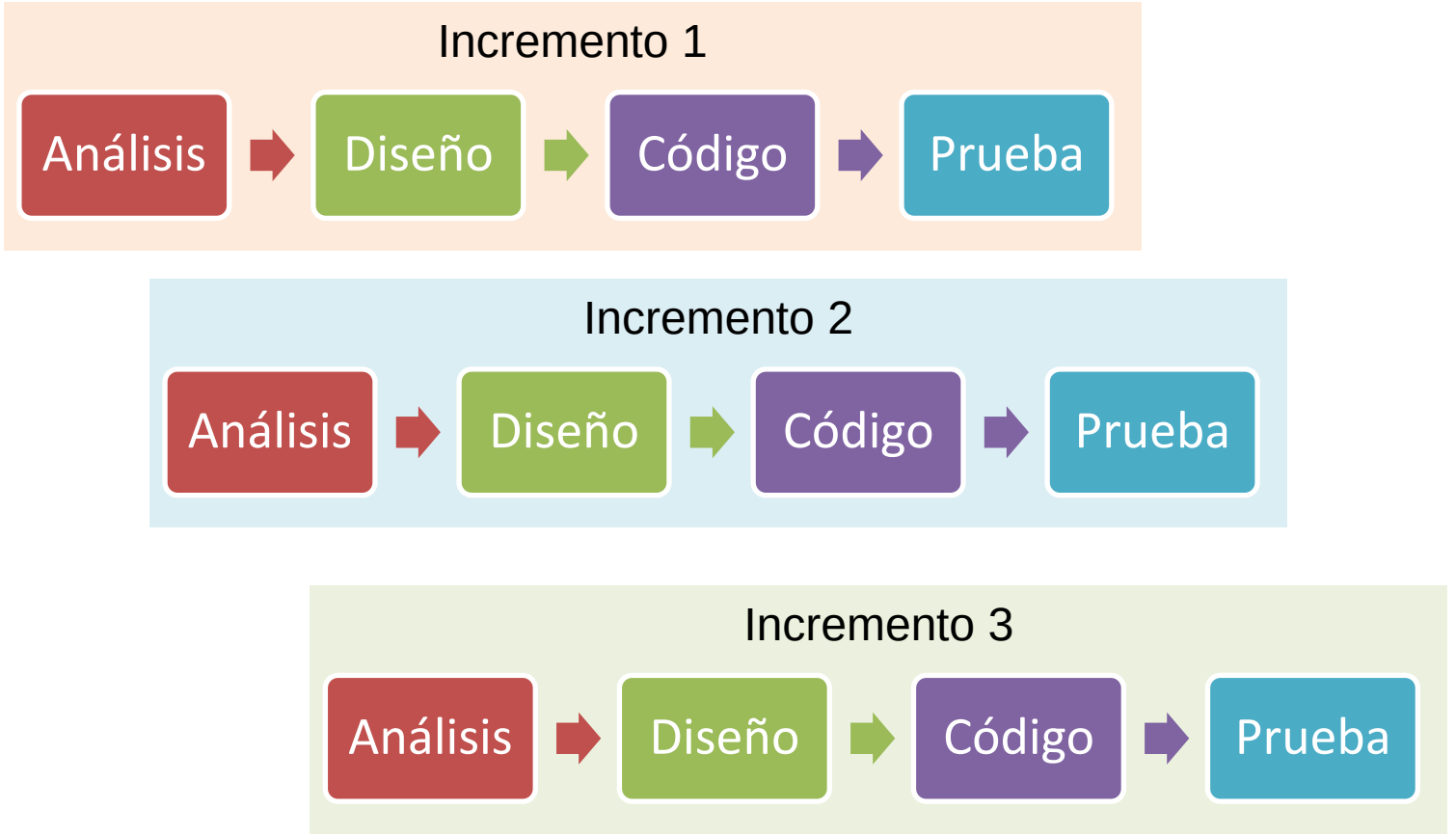
Modelos evolutivos

Modelo incremental

- Los clientes identifican a grandes rasgos los servicios que proporcionará el sistema
- Identifican qué servicios son más importantes y cuáles menos
- Se definen varios incrementos donde cada uno proporciona un subconjunto de la funcionalidad del sistema
- Una vez definidos los incrementos, se especifican con detalle los requisitos del primer incremento
- No se aceptan cambios en los requisitos del incremento actual

Modelos evolutivos

Modelo incremental



Modelos evolutivos

Modelo incremental

- Los primeros incrementos son versiones incompletas del producto final
- Se requiere mucha experiencia para definir los incrementos y distribuir en ellos las tareas de forma proporcionada
- Cada fase de una iteración es rígida y no se superpone con otras
- Todos los requisitos se deben definir al inicio

Modelos evolutivos

Modelo iterativo

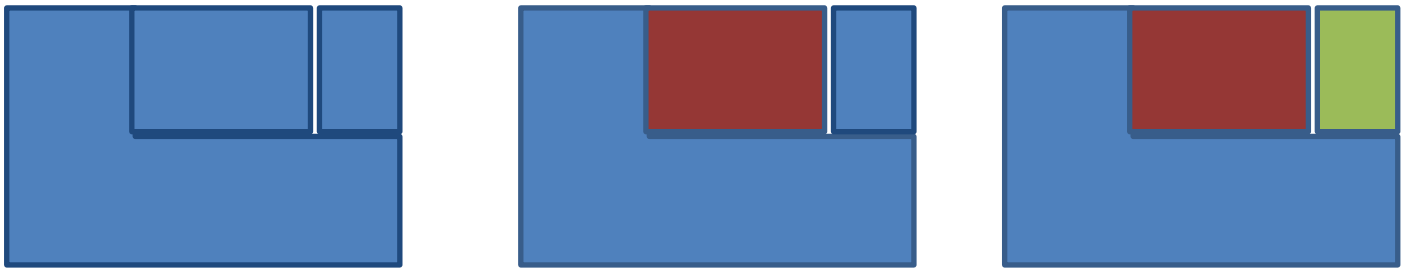
- Entrega un sistema completo desde el principio
- Posteriormente va cambiando la funcionalidad de cada subsistema con cada versión
- Pretende reducir el riesgo entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de recogida de requisitos
- Consiste en la iteración de varios ciclos de vida en cascada
- Al final de cada iteración se entrega al cliente una versión mejorada o con mayores funcionalidades

Modelos evolutivos

Modelo iterativo vs Modelo incremental

Desarrollo incremental → sistema parcial, funcionalidad completa

Desarrollo iterativo → sistema completo, funcionalidad parcial



Modelos evolutivos

Modelo basado en prototipos

- Un prototipo es un modelo experimental de un sistema o de un componente de un sistema que tiene los suficientes elementos que permiten su uso
  - Es una solución parcial que describe la interacción entre el hombre y la máquina, mostrando parte de su funcionalidad no optimizada
- Gracias al prototipo el cliente puede hacerse una idea de cómo está evolucionando el producto y esto ayuda a refinar los requisitos del sistema

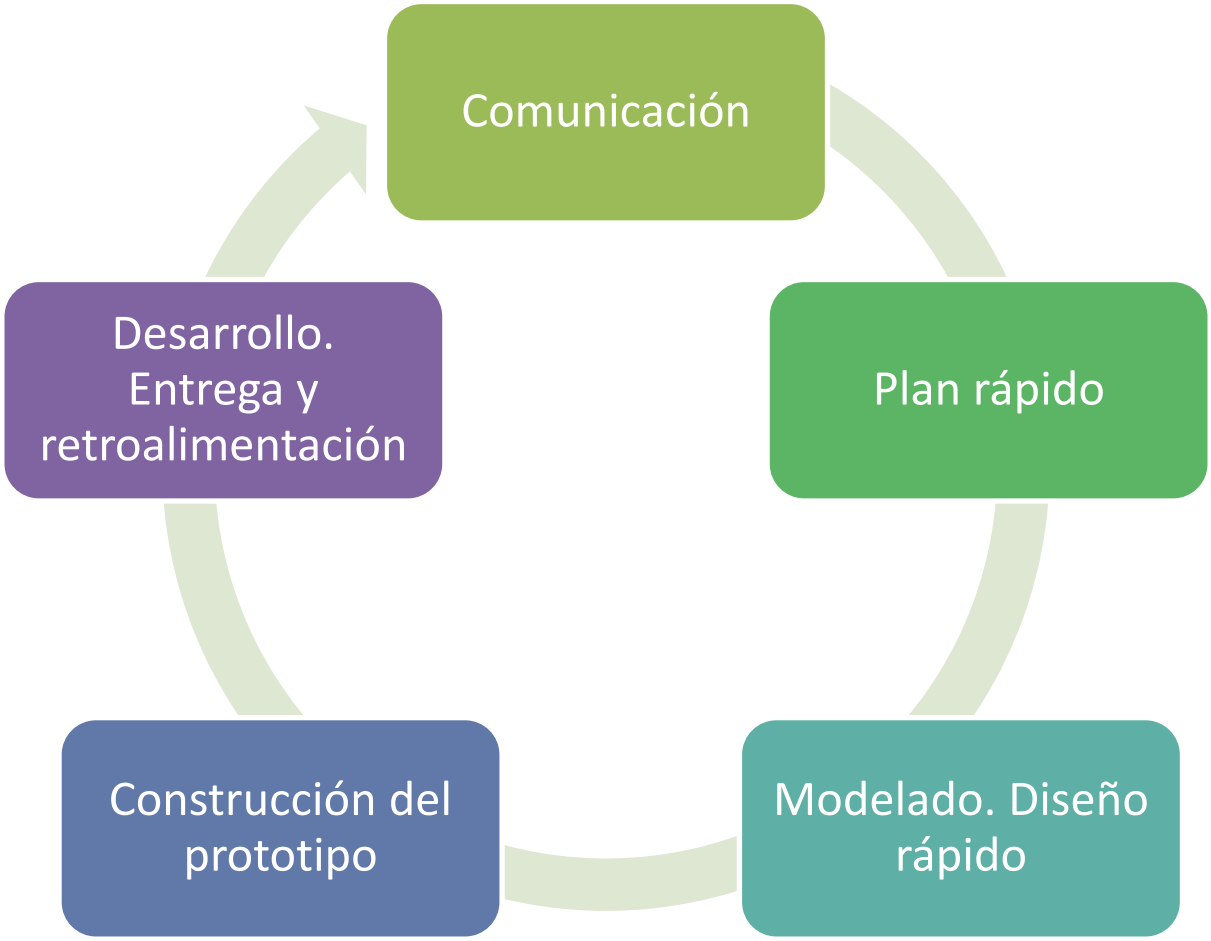
Modelos evolutivos

Modelo basado en prototipos

- Los requisitos de los negocios y productos cambian a lo largo del tiempo
- Los modelos secuenciales no son los más apropiados
- La entrega de una versión limitada posibilita la visión de la parte esencial de un sistema y además permite definir más detalles y extensiones
- Los ingenieros necesitan un modelo que permita trabajar con un producto que evoluciona con el tiempo

Modelos evolutivos

Modelo basado en prototipos



Modelos evolutivos

Modelo basado en prototipos

- El proceso comienza con la comunicación
- El ingeniero y el cliente definen los objetivos globales
- Se identifican los requisitos conocidos
- Se plantea con rapidez una iteración de construcción de prototipos y se presenta un diseño rápido
- El diseño rápido se centra en aspectos visibles para el cliente o el usuario final

Modelos evolutivos

Modelo basado en prototipos

- El diseño rápido lleva a la construcción de un prototipo
- El prototipo lo evalúa el cliente/usuario y con la retroalimentación se refinan los requisitos
- La iteración ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente
- El desarrollador entiende mejor qué debe hacer
- De manera ideal el prototipado debería servir como un mecanismo para identificar los requisitos

Modelos evolutivos

Modelo basado en prototipos

- El prototipo final que cumple las expectativas del cliente rara vez se convierte en un producto entregable
  - Suele ser lento
  - Está unido con “chicle”
  - Prisas para hacerlo funcionar
  - Alto coste de mantenimiento
- Debe entenderse como un mecanismo para determinar requisitos que sirva para desarrollar posteriormente el producto real

Modelos evolutivos

Modelo basado en prototipos

- ¿Cuándo es útil?
  - Se recomienda para clientes que quieren ver resultados a corto plazo
  - Cuando el cliente no sabe lo que quiere y los requisitos no están bien definidos desde el principio
  - Cuando los requisitos evolucionan muy rápidamente

Modelos evolutivos

Modelo en espiral

- Propuesto por Boehm en 1988
- Modelo híbrido:
  - Combina la naturaleza iterativa de la construcción de prototipos con los aspectos controlados y sistemáticos del modelo en cascada
- Es un modelo evolutivo del desarrollo, formado por un conjunto de vueltas de espiral:
  - En las primeras vueltas el Sw es un modelo en papel, la especificación de un producto. Todavía no funciona.
  - En las sucesivas vueltas, se desarrolla un prototipo.
  - En las últimas iteraciones se obtienen versiones completas del producto.

Modelos evolutivos

Modelo en espiral

- El número de actividades a realizar se incrementa notablemente a medida que nos alejamos del centro de la espiral. Las primeras son menos costosas
- La evaluación después de cada fase permite cambios
- Con este modelo obtenemos el producto final a partir de piezas más pequeñas

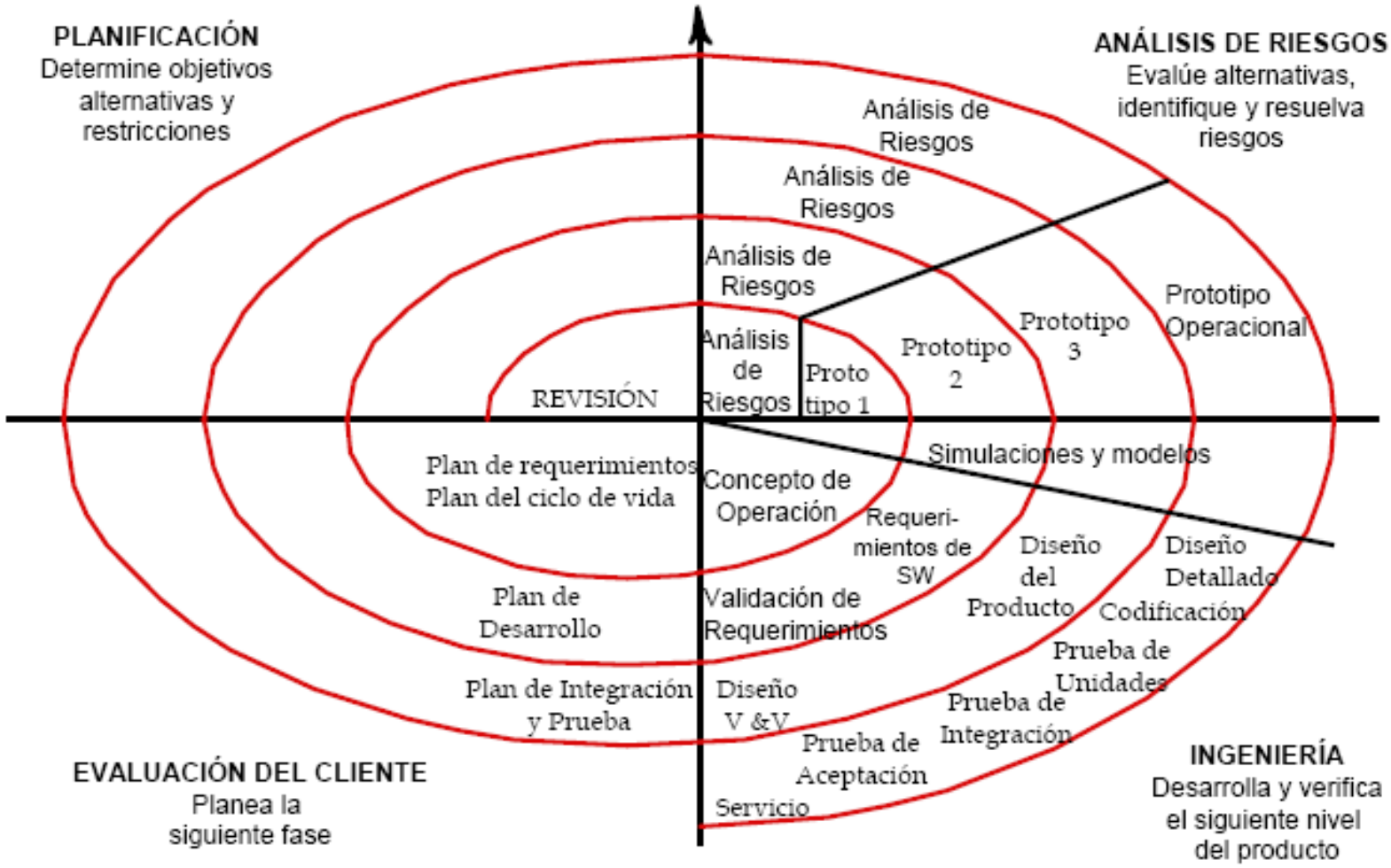
Modelos evolutivos

Modelo en espiral

- A diferencia de los demás, el modelo en espiral incorpora el factor **Riesgo -> es un modelo** orientado a riesgos:
  - Tiene como objetivo vital pensar en las cosas que pueden ir mal en el desarrollo del software y saber cómo resolverlas.
- La gestión de riesgos es una actividad muy importante en la gestión de un proyecto

Modelos evolutivos

Modelo en espiral



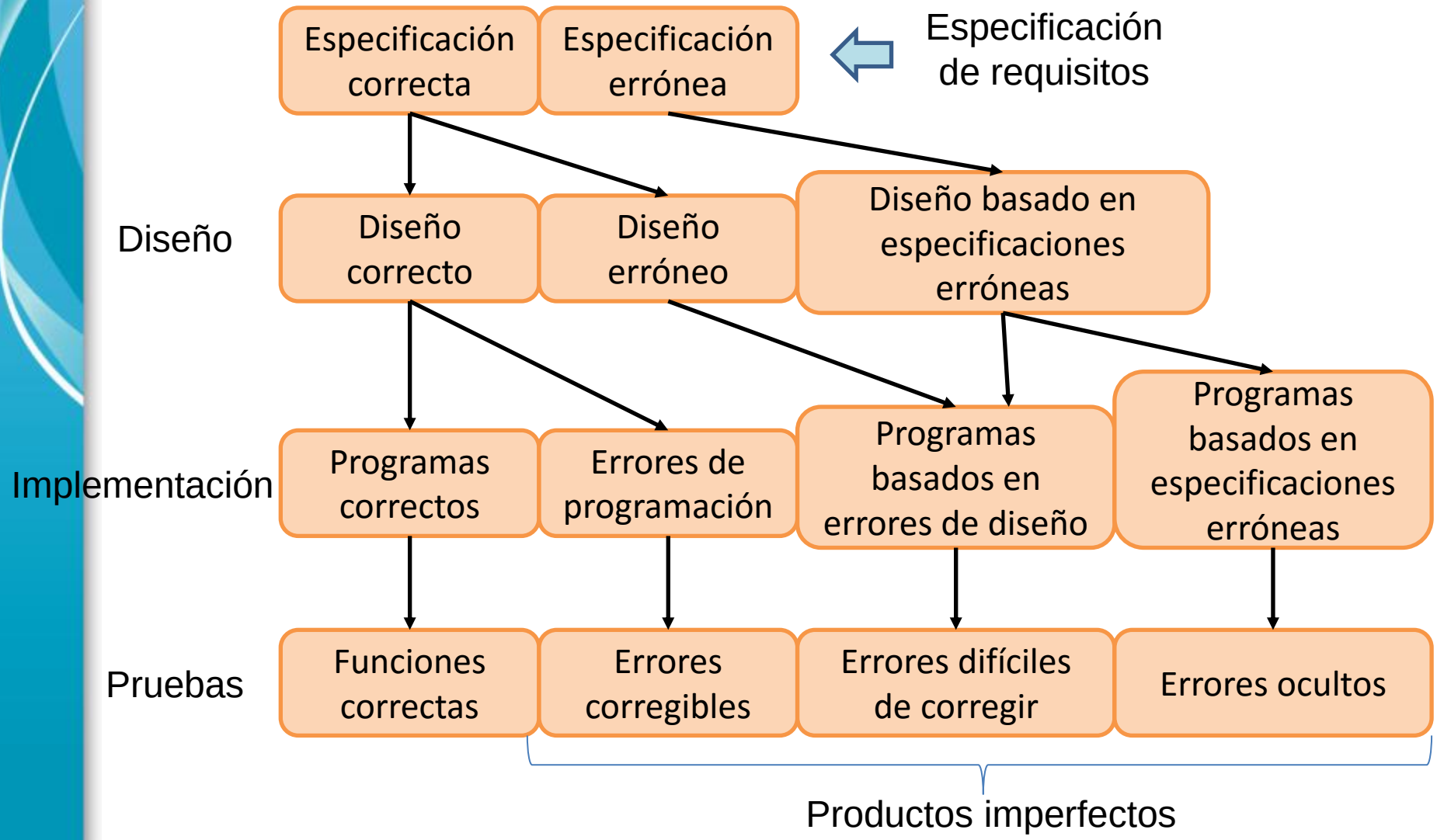
[IEEE, 1988]

Modelos evolutivos

Modelo en espiral

- Cada ciclo se divide en 4 sectores:
  1. Definición de objetivos: restricciones del producto y proceso, plan de gestión, identificación de riesgos
  2. Evaluación y reducción de riesgos: por ejemplo, mejor definición de requerimientos mediante prototipos
  3. Desarrollo y validación: elección de un modelo para el desarrollo
  4. Planificación: el proyecto se revisa y se decide si se continúa con el siguiente ciclo. Si es así, se planifica la siguiente fase

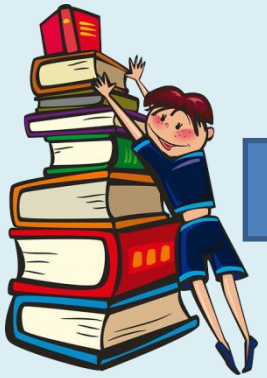
Efectos producidos por errores en las fases de desarrollo



Análisis y Diseño

Problema vs Solución

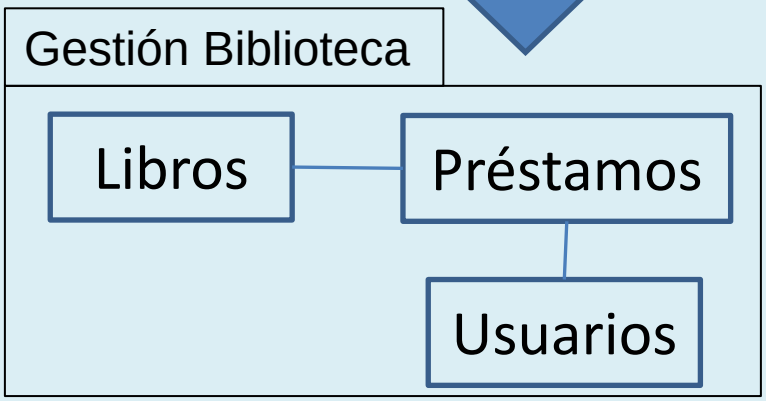
ANÁLISIS



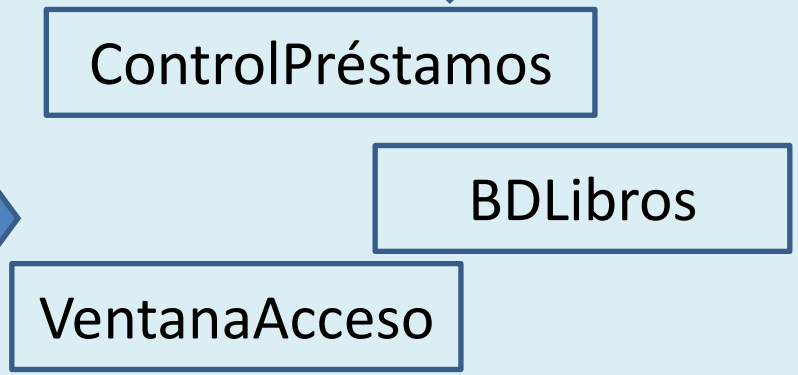
Dominio del problema

DISEÑO

Dominio de la solución

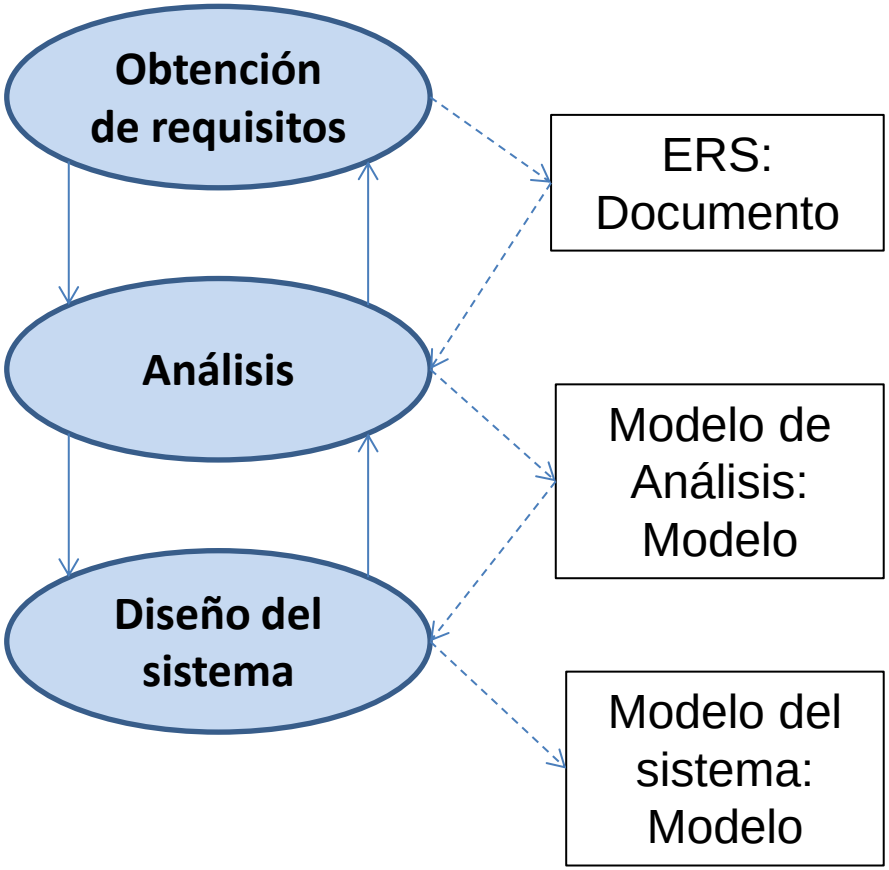


Modelo del dominio del problema



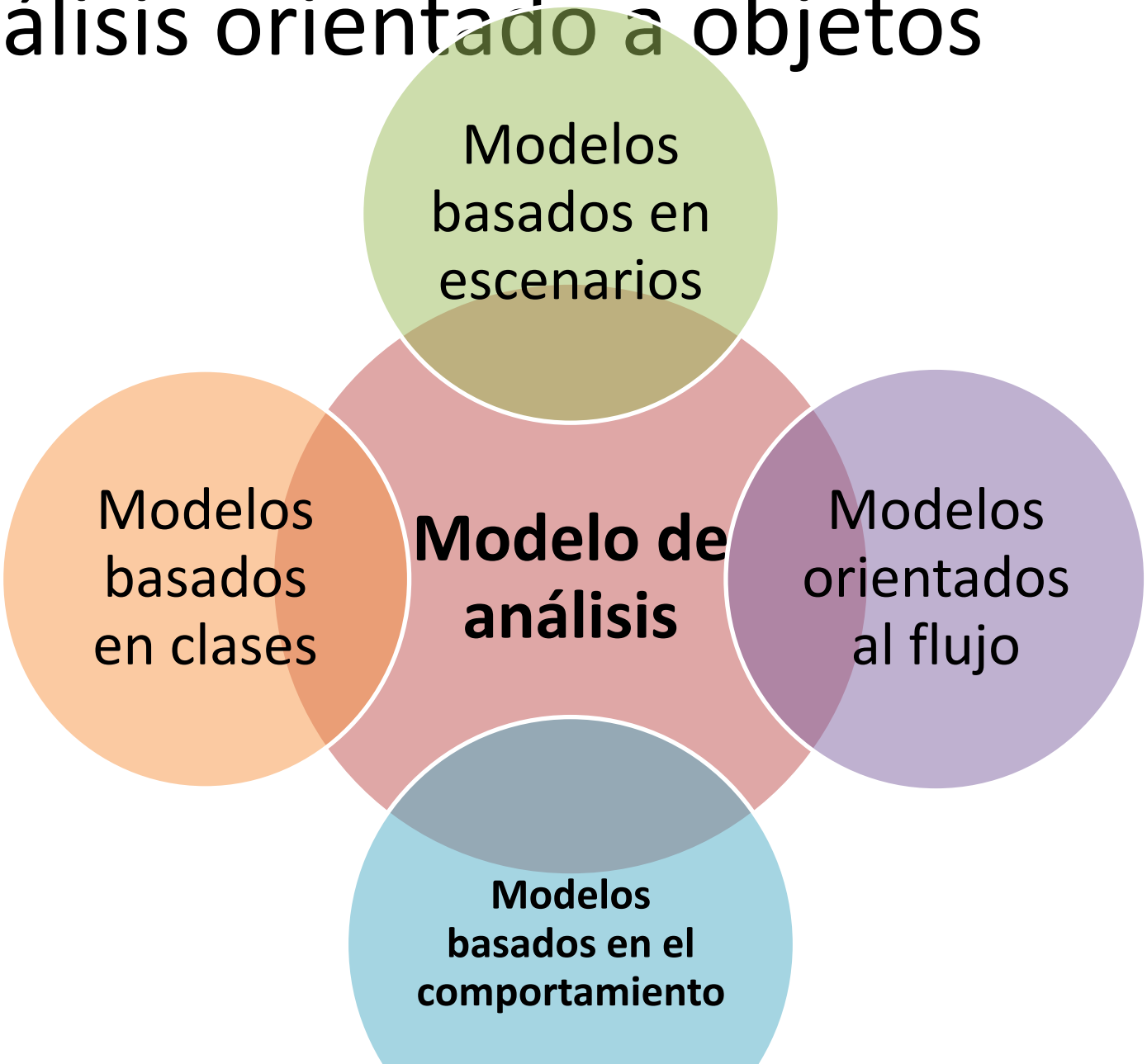
Modelo del dominio de la solución

Análisis orientado a objetos



- Centrarse en el “qué”
- La ERS describe el sistema, en lenguaje natural
- Sirve de comunicación entre desarrolladores y clientes
- El modelo de análisis usa notación formal o semi-formal: UML
- Sirve de comunicación entre desarrolladores

Análisis orientado a objetos



Modelado de software

- Un **modelo** es una **abstracción** de un sistema o entidad del mundo real
- Una **abstracción** es una **simplificación**, que incluye sólo aquellos detalles relevantes para algún determinado propósito
- El modelado permite abordar la complejidad de los sistemas

Modelado de 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;}
}
```

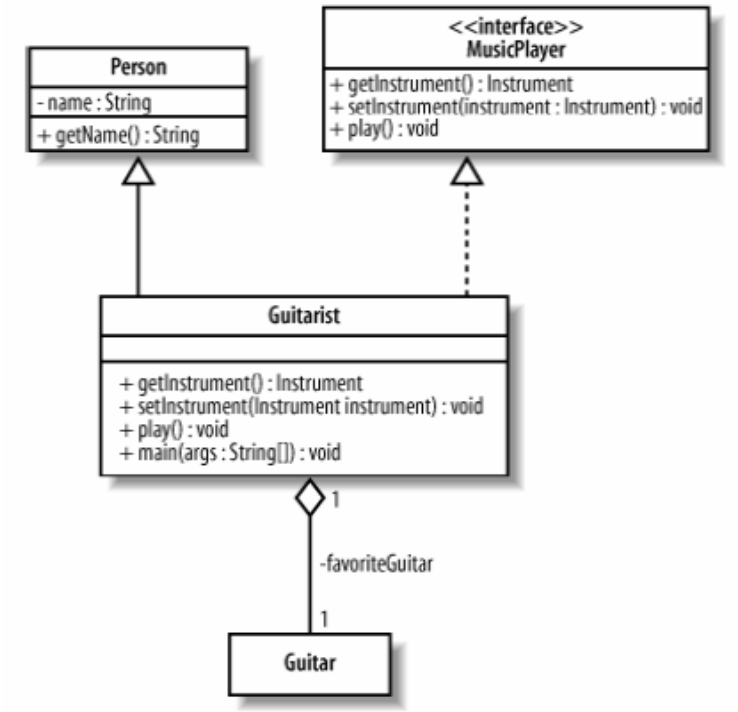
- Representa sólo la lógica e ignora el resto
- El ser humano lo interpreta muy lentamente
- No facilita la reutilización ni la comunicación

Modelado de 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

Modelado de 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

Modelado visual

- ¿Qué es el modelado visual?
  - El modelado visual proporciona una plantilla del sistema utilizando notaciones gráficas
  - Visualizar esta plantilla ayuda a entender el sistema
  - Permite entender la estructura o el comportamiento del mismo
  - Sirve de guía durante el proceso de desarrollo de SW
  - Permite documentar las decisiones que se toman

Modelado visual

- Podemos comparar el desarrollo de SW con la construcción de una casa para un perro, una casa para una familia o un rascacielos [Booch, 1999]
- Aunque la construcción que se planee hacer sea una casa sencilla, el resultado será más satisfactorio si se cuenta con el respaldo de un correcto diseño



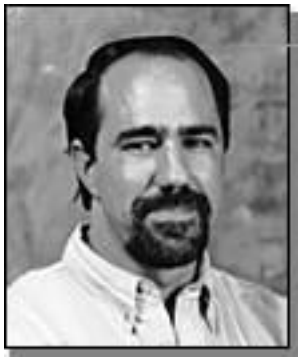
Beneficios del modelado visual

- Captura procesos desde la perspectiva del usuario
- Incrementa la comunicación
- Define la arquitectura
- Ayuda a reducir la complejidad
- Promueve la reutilización de componentes
- Es independiente del lenguaje de implementación

UML

UNIFIED MODELING LANGUAGE™

- UML es un lenguaje de modelado visual
- Es independiente del proceso
- No se considera una metodología
- Define la notación utilizada para representar los diseños
- Nació de la unión de las teorías de:



Grady Booch

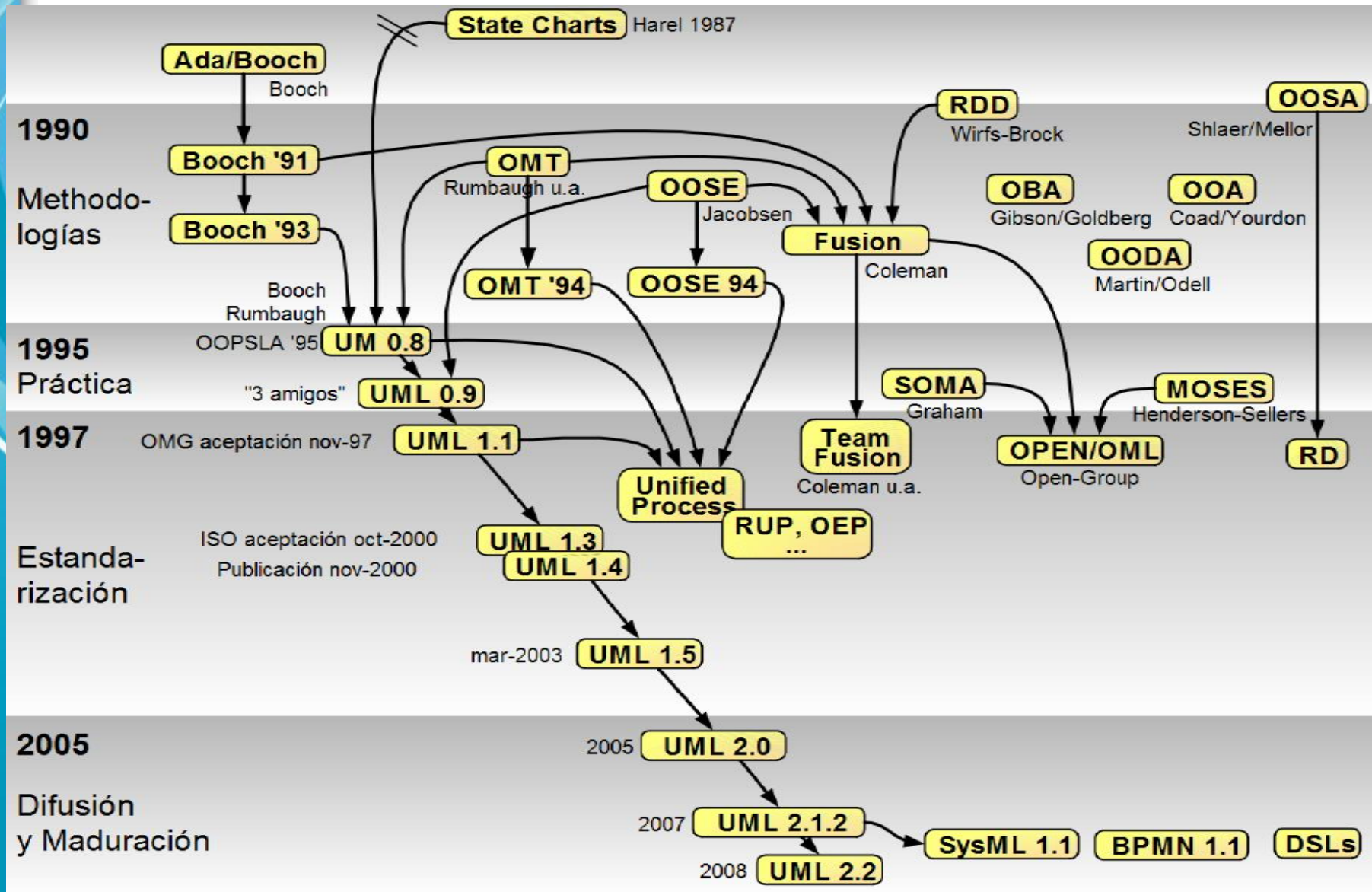
Ivar Jacobson

James Rumbaugh

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 correspondientes gurús)
- Necesidad de una notación estándar.

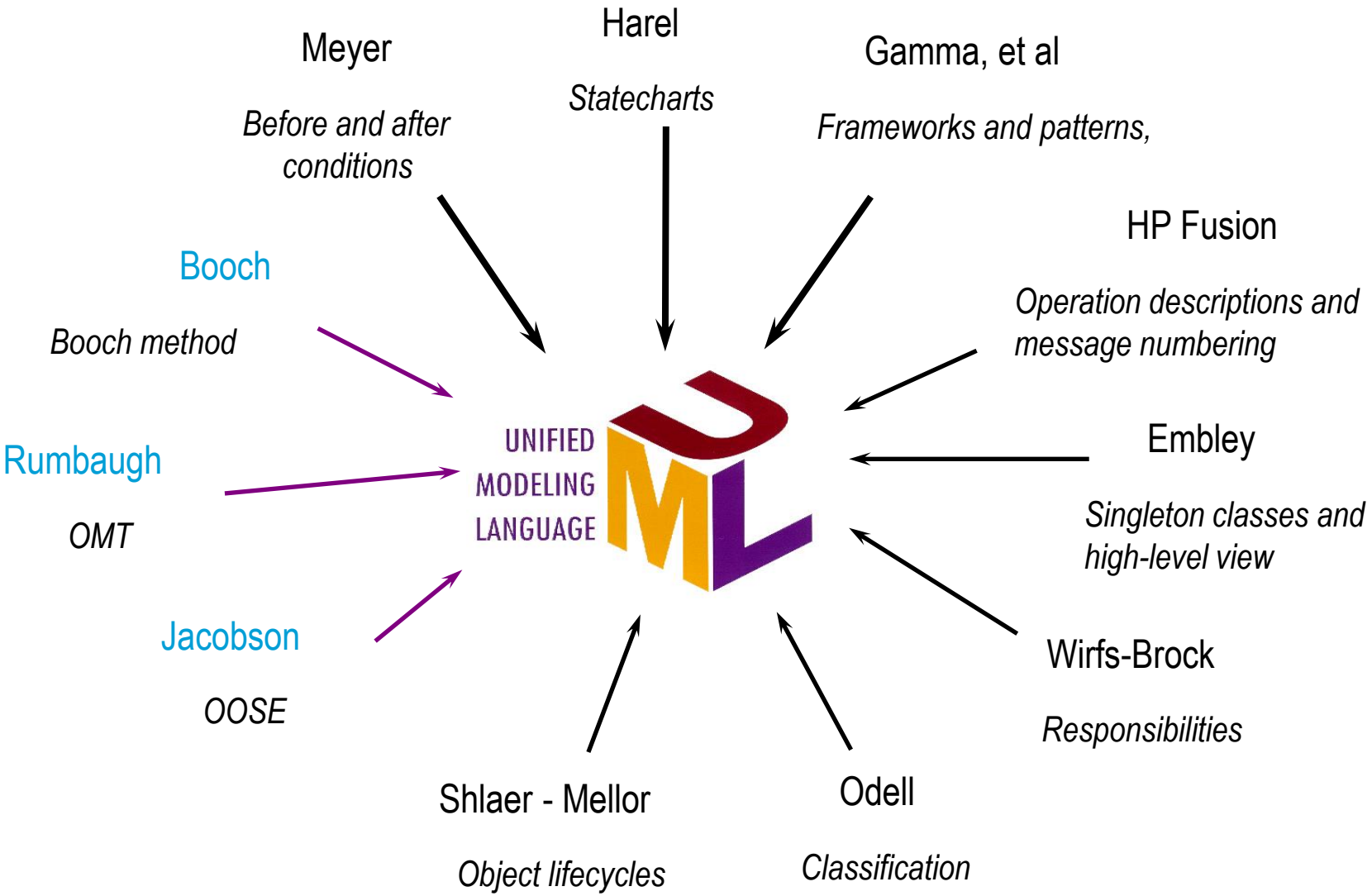
Evolución de UML



Evolución de UML

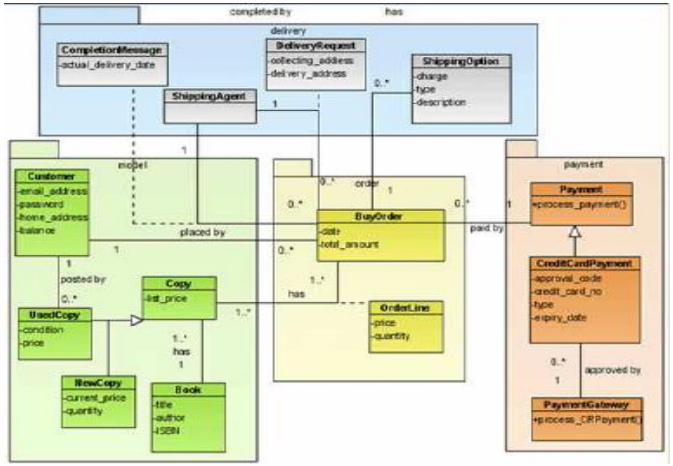
- Los pilares básicos de UML:
  - OMT: Object Modeling Technique (Rumbaugh et al.)
    - Especialmente bueno para análisis de datos de Sistemas de Información
    - Entre otros, usa extensiones de los diagramas Entidad-Relación
  - Método-Booch (G. Booch)
    - Especialmente útil para sistemas concurrentes y de tiempo real
    - Fuerte relación con lenguajes de programación, como Ada
  - OOSE: Object-Oriented Software Engineering (I. Jacobson)
    - Desarrollo guiado por casos de uso
    - Buen soporte de ingeniería de requisitos e ingeniería de información
    - Modelado y simulación de sistemas de comunicaciones

UML


Meyer
Before and after conditions
Harel
Statecharts
Gamma, et al
Frameworks and patterns,
HP Fusion
Operation descriptions and message numbering
Booch
Booch method
Rumbaugh
OMT
Jacobson
OOSE
UNIFIED MODELING LANGUAGE
Embley
Singleton classes and high-level view
Wirfs-Brock
Responsibilities
Shlaer - Mellor
Object lifecycles
Odell
Classification

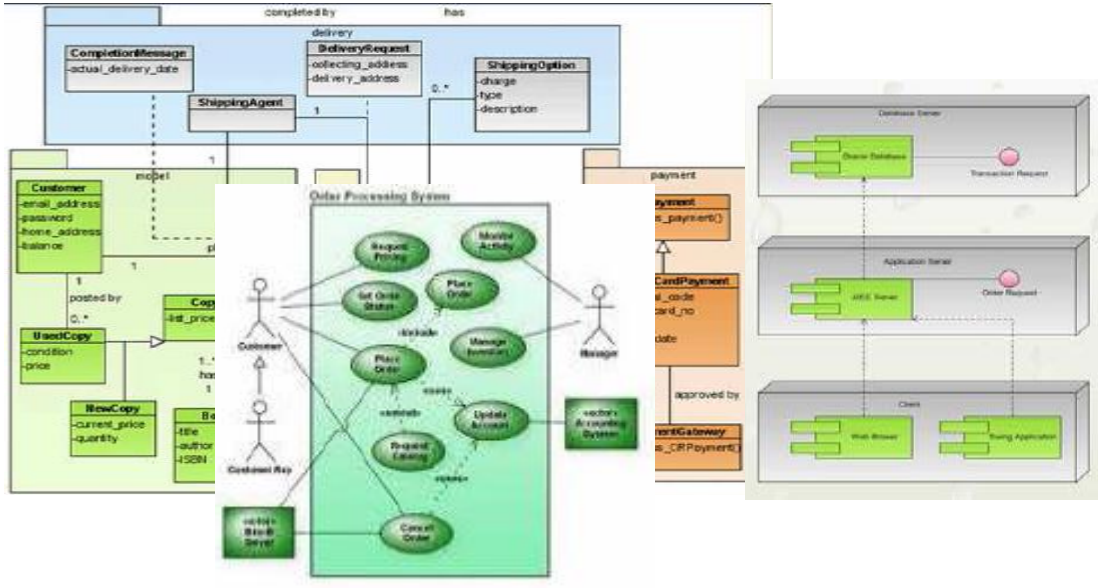
Características de UML

- UML es un lenguaje de modelado ≠ método
- Lenguaje = Notación + Reglas (Sintácticas, Semánticas)
- UML ofrece vocabulario y reglas:
  - para crear y leer modelos bien formados
  - que constituyen los **planos de un sistema software**



Características de UML

- UML es independiente del Proceso de desarrollo
- UML cubre las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de software



Ventajas de UML

- Es estándar → Facilita la comunicación
- Está basado en un metamodelo con una semántica bien definida
- Se basa en una notación gráfica concisa y fácil de aprender y utilizar
- Se puede utilizar para modelar sistemas software en diversos dominios:
  - Sistemas de información empresariales, Sistemas WEB, sistemas críticos y de tiempo real, etc
  - Incluso en sistemas que no son software
- Es fácilmente extensible

Objetivos de UML

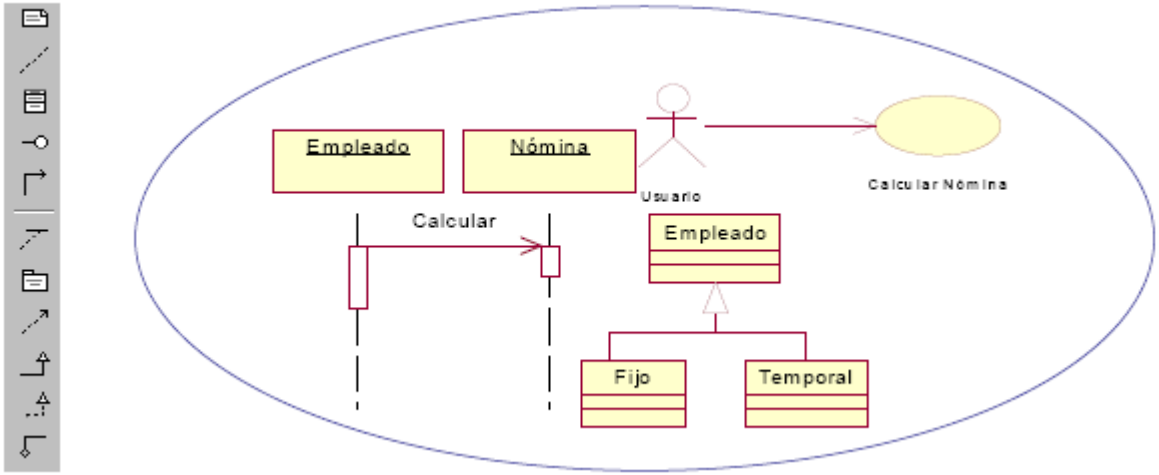
- UML es un lenguaje de modelado visual para:
  - Visualizar.
  - Especificar.
  - Construir.
  - Documentar.

los artefactos de un sistema software

independientemente de la metodología de análisis y diseño pero con una perspectiva orientada a objetos

Objetivos de UML

- Visualizar
  - Detrás de cada símbolo en UML hay una semántica bien definida
  - Es más que un montón de símbolos gráficos
  - Trasciende lo que puede ser representado en un lenguaje de programación
  - Modelo explícito, que facilita la comunicación

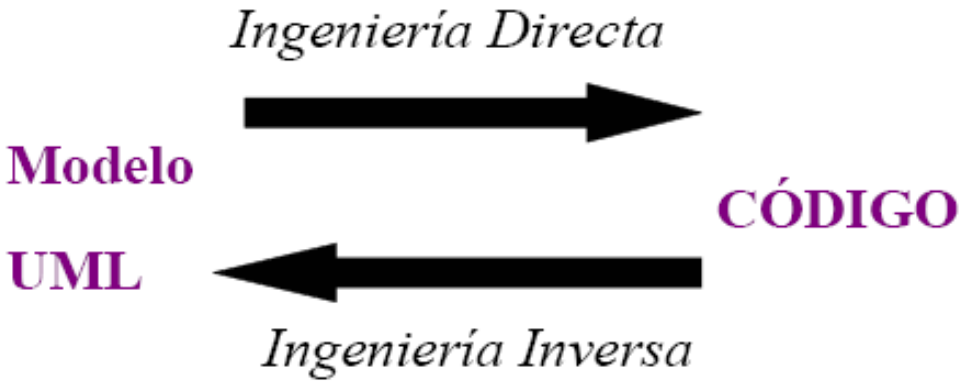


Objetivos de UML

- Especificar
  - Especificar es equivalente a construir modelos precisos, no ambiguos y completos
  - UML cubre la especificación del análisis, diseño e implementación de un sistema intensivo en software

Objetivos de UML

- Construir
  - UML no es un lenguaje de programación visual, pero es posible establecer correspondencias entre un modelo UML y lenguajes de programación (Java, C++) y bases de datos (relacionales, OO)



Objetivos de UML

- Documentar
  - UML cubre toda la documentación de un sistema:
    - Arquitectura del sistema y sus detalles (diseño)
    - Expresar requisitos y pruebas
    - Modelar las actividades de planificación y gestión de versiones

Importancia en el mantenimiento

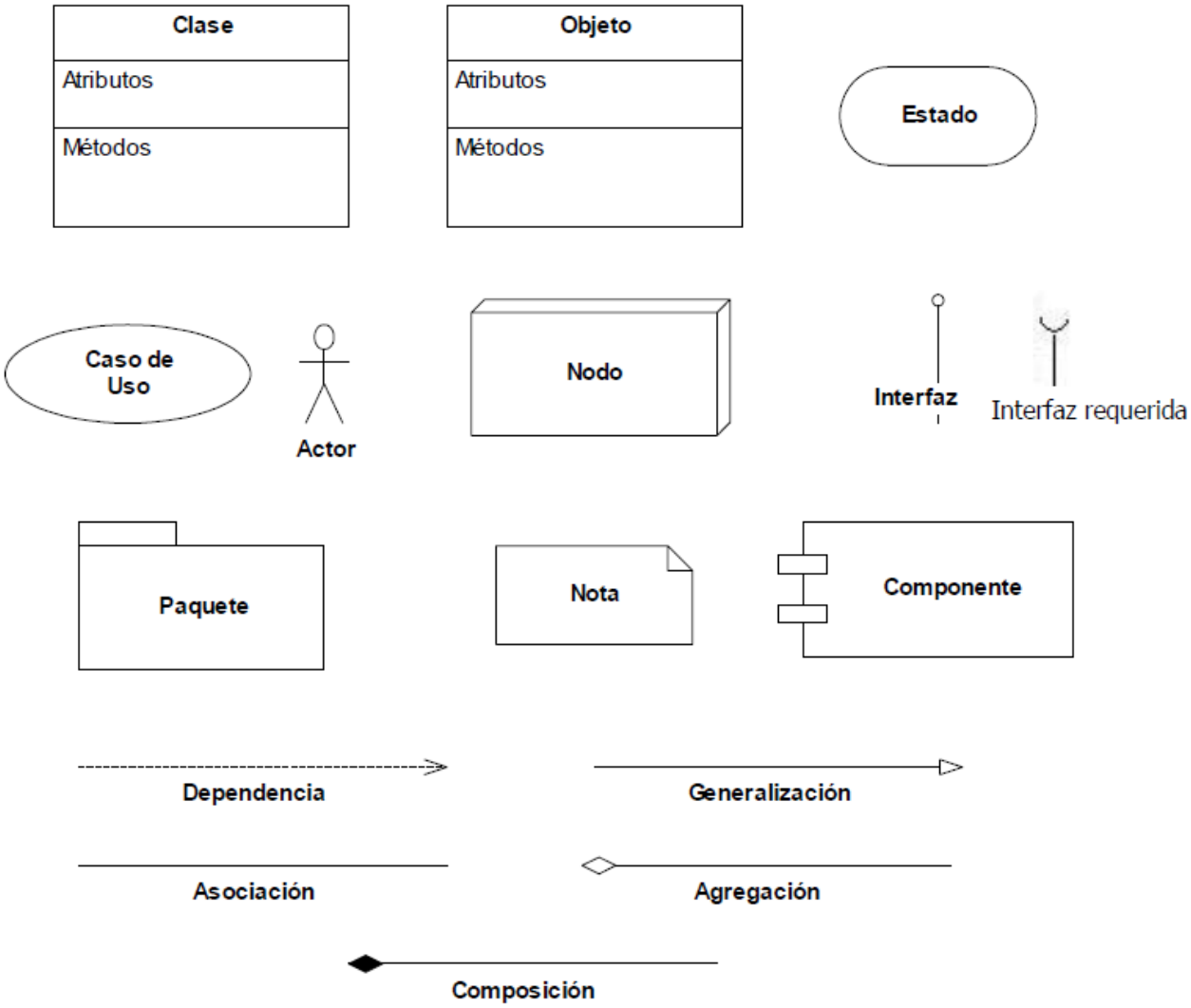
Elementos de modelado

- Los elementos son los bloques básicos de UML:
  - Representan las abstracciones del sistema en curso de modelado
  - Son los conceptos utilizados en los diagramas, que representan los conceptos del paradigma objetual (clases, objetos, relaciones...)
- Un elemento de modelado puede estar en diferentes diagramas, pero siempre con el mismo significado y símbolo asociado

Elementos de modelado

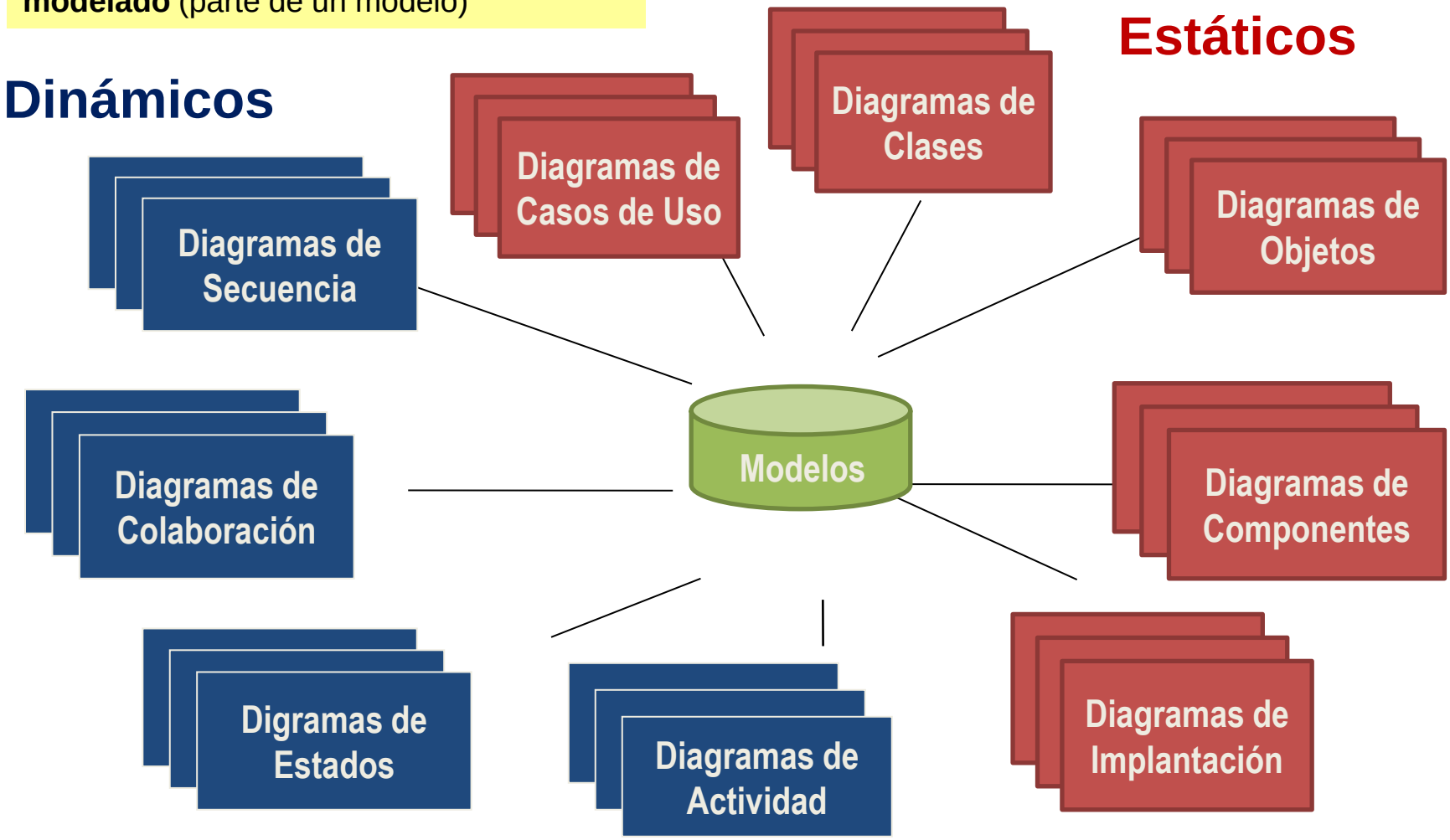
- Los elementos de modelado se pueden agrupar en paquetes:
  - Los paquetes ofrecen un mecanismo general para la partición de los modelos y la agrupación de los elementos de modelado
  - La arquitectura del sistema viene expresada por la jerarquía de paquetes y por la red de relaciones de dependencia entre paquetes
  - Un paquete puede contener otros paquetes, sin límite del nivel de anidamiento
    - Un nivel dado puede contener una mezcla de paquetes y otros elementos de modelado

Elementos de modelado



Diagramas de UML

Un **diagrama** es la **representación gráfica** de un conjunto de **elementos de modelado** (parte de un modelo)



[Jacobson, 1999]

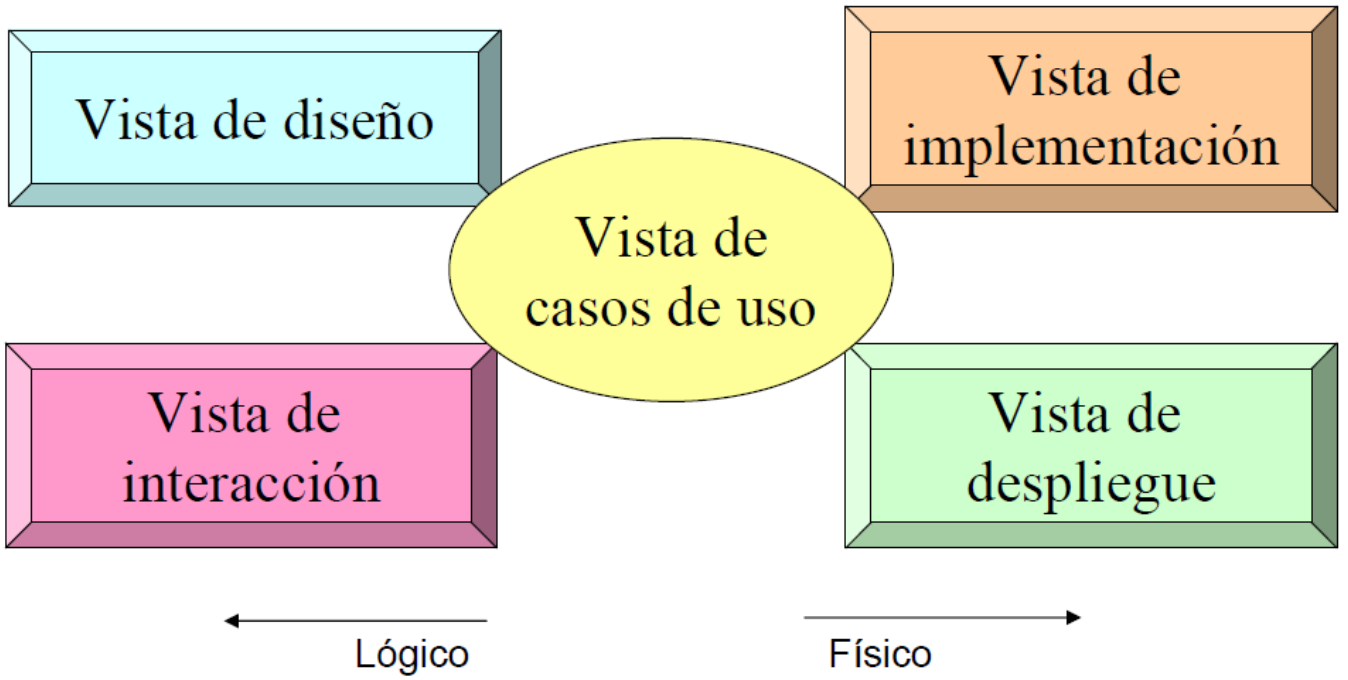
Vistas arquitecturales

- Durante el desarrollo de un sistema software se requiere que éste sea visto desde varias perspectivas
- Diferentes usuarios miran el sistema de formas diferentes en momentos diferentes
- La arquitectura del sistema es clave para poder manejar estos puntos de vista diferentes:
  - Se organiza mejor a través de vistas arquitecturales interrelacionadas
  - Proyecciones del modelo del sistema centradas en un aspecto particular
- No se requiere una vista que contenga la semántica completa de la aplicación. La semántica reside en el modelo

Vistas arquitecturales

Las 4 +1 Vistas Arquitecturales

(Philippe Krutchen)

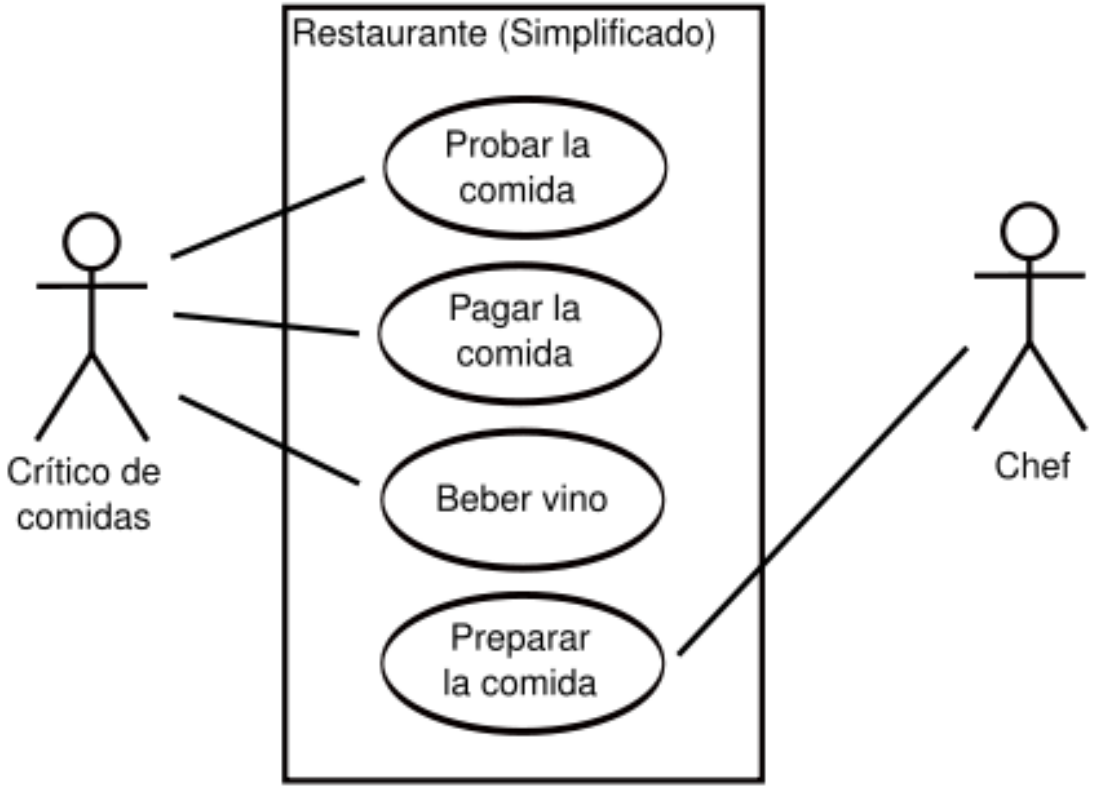


Vistas arquitecturales

- Vista de Casos de uso
  - Captura la **funcionalidad** del sistema tal y como es percibido por los usuarios finales, analistas y encargados de pruebas
  - Describe la funcionalidad en base a casos de uso
  - En esta vista no se especifica la organización real del sistema software
  - Los diagramas que le corresponden son:
    - **Aspectos estáticos**: diagramas de casos de uso
    - **Aspectos dinámicos**: diagramas de interacción, de estados y de actividades

Vistas arquitecturales

- Vista de Casos de Uso



http://es.wikipedia.org/wiki/Diagrama_de_casos_de_uso

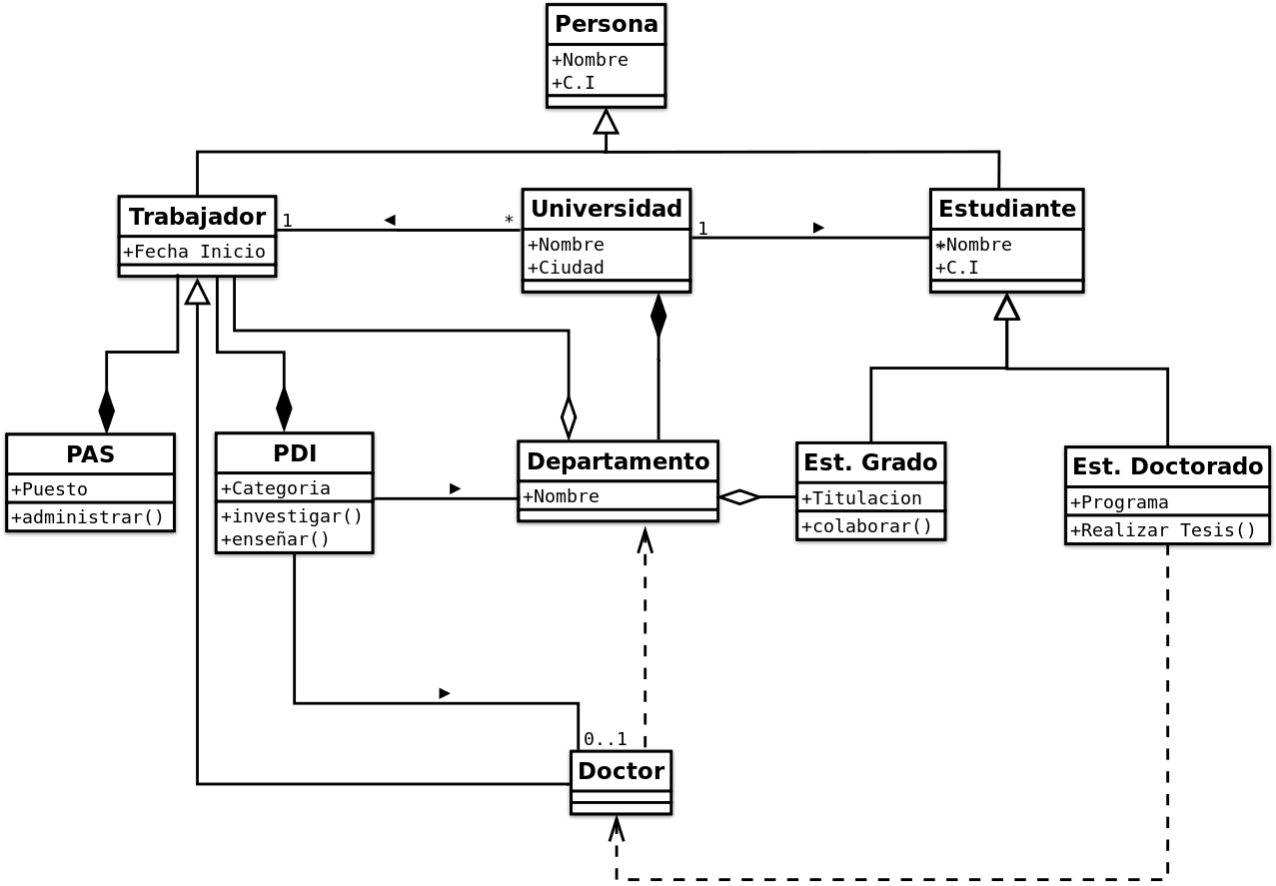
Vistas arquitecturales

- Vista de diseño
  - Captura las **clases, interfaces y colaboraciones** que describen el sistema:
    - En el dominio del problema
    - En el dominio de la solución
  - Los elementos que la forman dan soporte a los requisitos funcionales del sistema
  - Los diagramas que le corresponden son:
    - **Aspectos estáticos**: diagramas de clases y de objetos. También son útiles los diagramas de estructura compuesta de clases
    - **Aspectos dinámicos**: diagramas de interacción, de estados y de actividades

Vistas arquitecturales

- Vista de diseño

  Diagrama de Clases



http://es.wikipedia.org/wiki/Diagrama_de_clases

Vistas arquitecturales

- Vista de interacción
  - Captura el **flujo de control** entre las diversas partes del sistema, incluyendo los posibles mecanismos de **concurrencia y sincronización**
  - Abarca en especial **requisitos no funcionales** como el rendimiento, escalabilidad y capacidad de procesamiento
  - Los diagramas que le corresponden son los mismos que la vista de diseño:
    - **Aspectos estáticos**: diagramas de clases y de objetos
    - **Aspectos dinámicos**: diagramas de interacción, de estados y de actividades
  - Pero atendiendo más las clases activas que controlan el sistema y los mensajes entre ellas

Vistas arquitecturales

- Vista de interacción

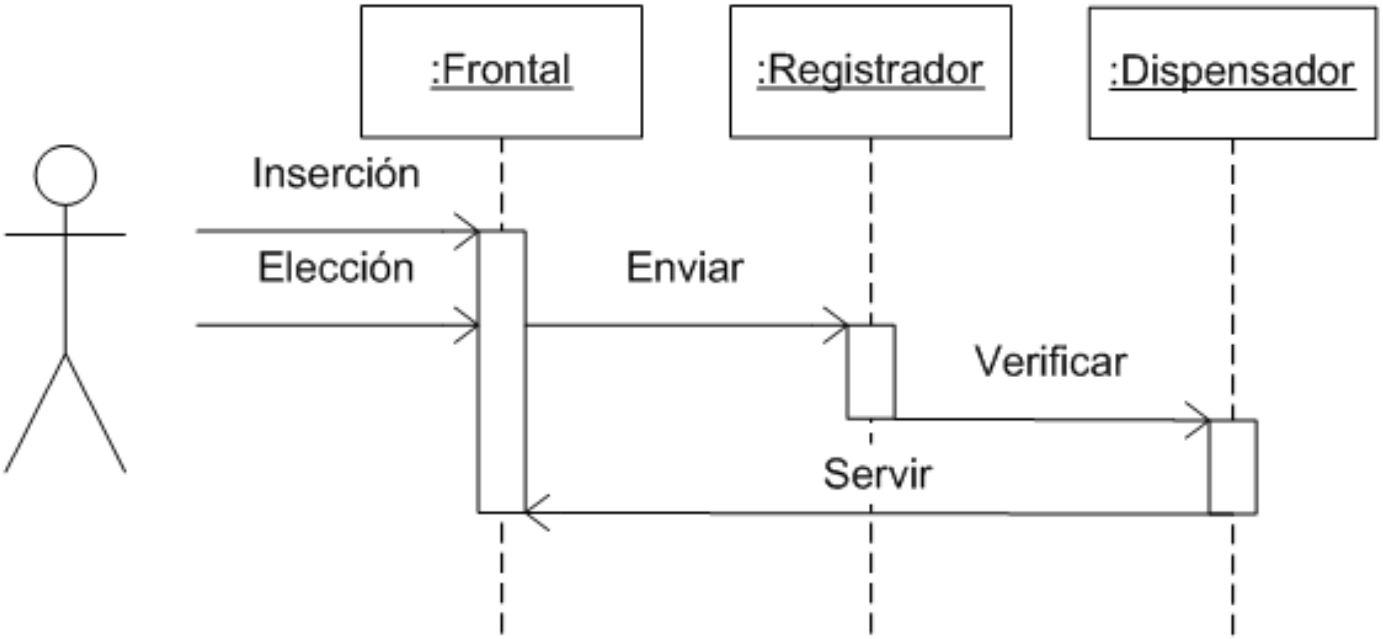


Diagrama de secuencia: Máquina expendedora

Vistas arquitecturales

- Vista de implementación
  - Captura los **artefactos** que se utilizan para ensamblar y poner en **producción el sistema software real**
  - Define la **arquitectura física del sistema**
  - Se centra en:
    - La configuración de las distintas versiones de los archivos físicos
    - Correspondencia entre clases y ficheros de código fuente
    - Correspondencia entre componentes lógicos y artefactos físicos
  - Los diagramas que le corresponden son:
    - **Aspectos estáticos**: diagramas de componentes (especialmente la versión de artefactos) y de estructura compuesta
    - **Aspectos dinámicos**: diagramas de interacción, de estados y de actividades

Vistas arquitecturales

- Vista de implementación

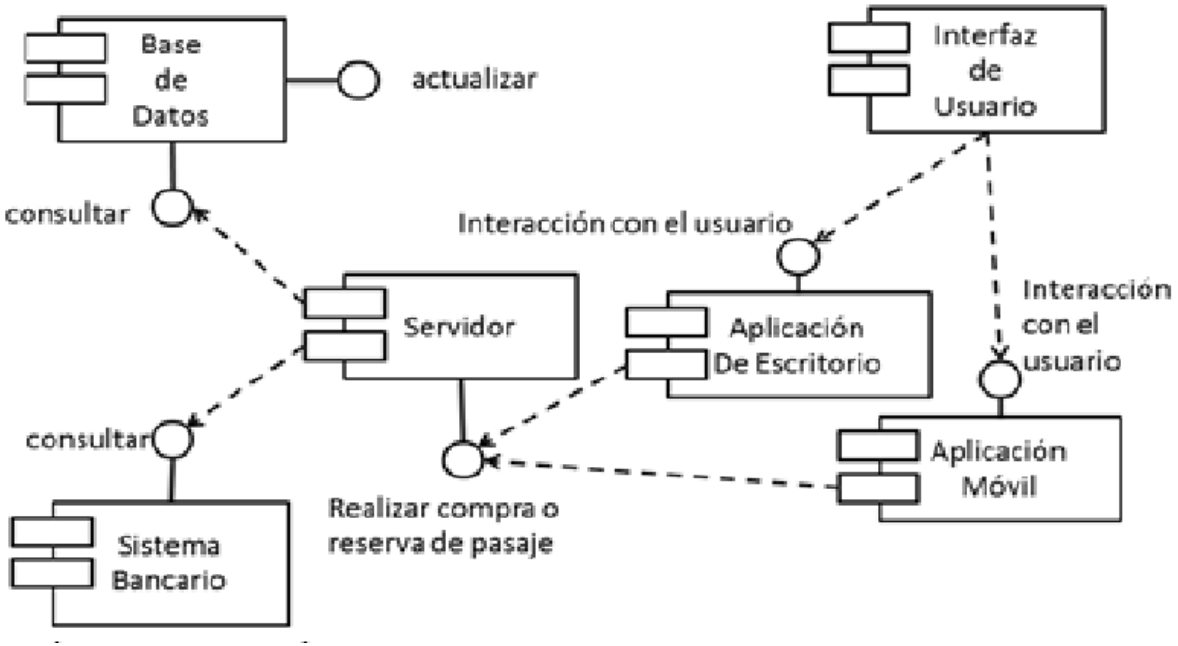


Diagrama de componentes

Vistas arquitecturales

- Vista de despliegue
  - Captura las características de **instalación y ejecución del sistema**
  - Contiene los **nodos y enlaces** que forman la topología **hardware** sobre la que se ejecuta el sistema software
  - Se ocupa principalmente de la distribución de las partes que forman el sistema software real
  - Los diagramas que le corresponden son:
    - **Aspectos estáticos:** diagramas de despliegue
    - **Aspectos dinámicos:** diagramas de interacción, de estados y de actividades

Vistas arquitecturales

- Vista de despliegue

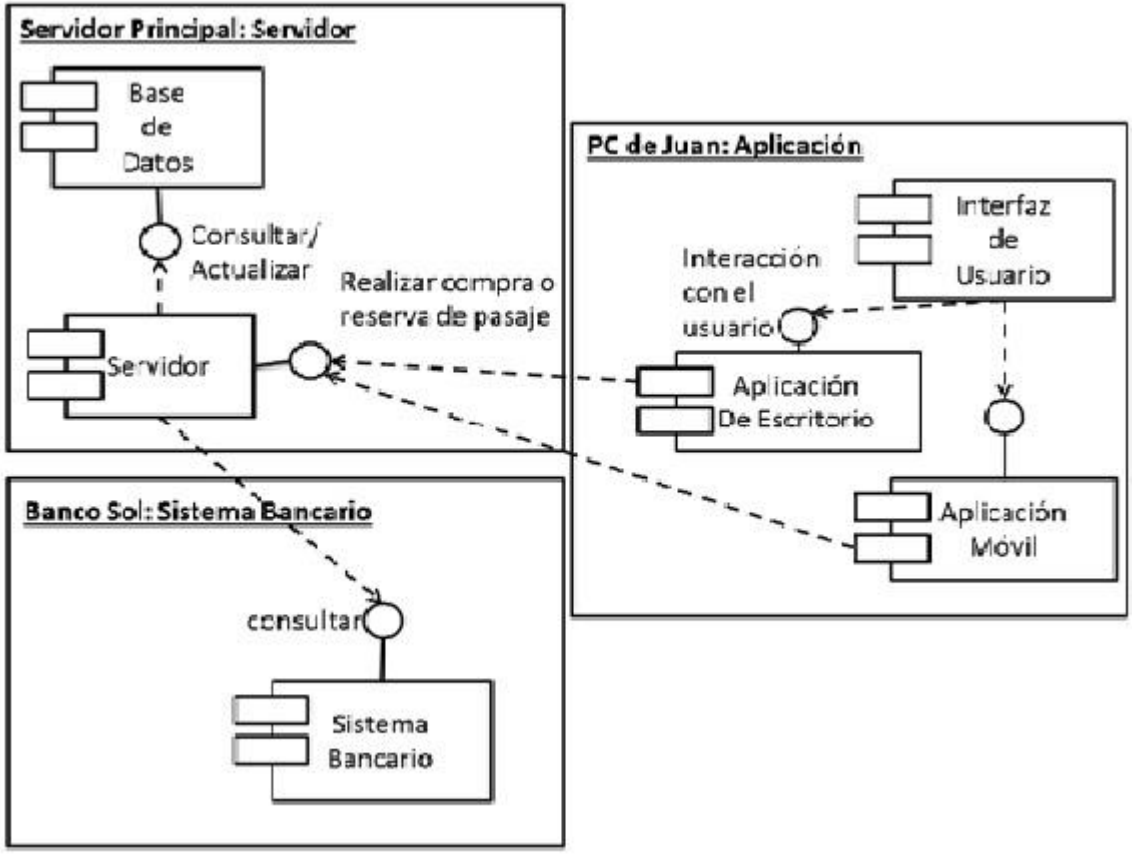
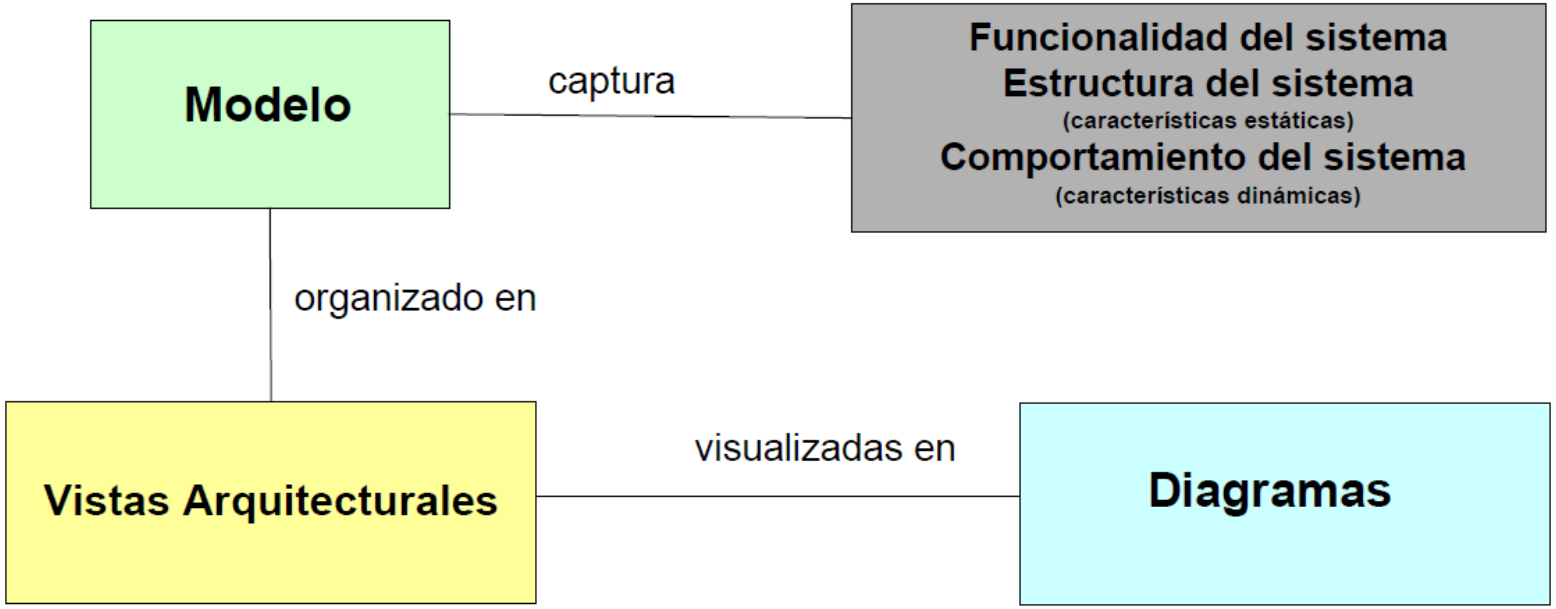


Diagrama de despliegue

Vistas arquitecturales

- Cada vista puede **existir de forma independiente**
- Pero **interactúan** entre sí:
  - Los nodos (vista de despliegue) contienen componentes (vista de implementación)
  - Dichos componentes representan la realización (software real) de las clases, interfaces, colaboraciones y clases activas (vistas de diseño y de interacción)
  - Dichos elementos de las vistas de diseño e interacción representan el sistema solución a los casos de uso (vista de casos de uso) que expresan los requisitos

Modelo UML de un sistema



- Vista de Casos de Uso (Funcionalidad)
- Vista de Diseño (Arquitectura interna)
- Vista de Interacción (Concurrencia y sincronización)
- Vista de Implementación (Arquitectura externa)
- Vista de Despliegue (Plataforma y distribución)

- Diagramas de Casos de Uso
- Diagramas de Clases
- Diagramas de Objetos
- Diagramas de Secuencia
- Diagramas de Colaboración
- Diagramas de Estado
- Diagramas de Actividad
- Diagramas de Componentes
- ……

Resumen

- El modelo UML de un sistema consiste en:
  - Un conjunto de **elementos de modelado** que definen la estructura, el comportamiento y la funcionalidad del sistema y que se agrupan en una **base de datos única**
  - La presentación de esos conceptos a través de múltiples **diagramas** con el fin de introducirlos, editarlos, y hacerlos comprensibles
  - Los diagramas pueden agruparse en **vistas**, cada una enfocada a un aspecto particular del sistema
- La gestión de un modelo UML requiere una **herramienta específica** que mantenga la **consistencia** del modelo

UML

- Compañías involucradas en el desarrollo de UML:

- Rational Software Corporation
- Hewlett-Packard
- I-Logix
- IBM
- ICON computing
- Intellicorp
- MCI Systemhouse
- Microsoft
- ObjecTime
- Oracle
- Platinum Technology
- Taskon
- Texas Instruments/Sterling Software
- Unisys

Bibliografía

- UML distilled. A brief guide to the standard object modeling language. Martin Fowler
- UML gota a gota. Martin Fowler y Kendall Scott
- Análisis y diseño orientado a objetos de sistemas usando UML. Simon Bennett