



Universidad de Guayaquil



Unidad 1

Definición y diseño de software en su ciclo de vida de desarrollo

Material docente compilado por el profesor Ph.D. Franklin Parrales Bravo para uso de los cursos de Diseño y Arquitectura de Software



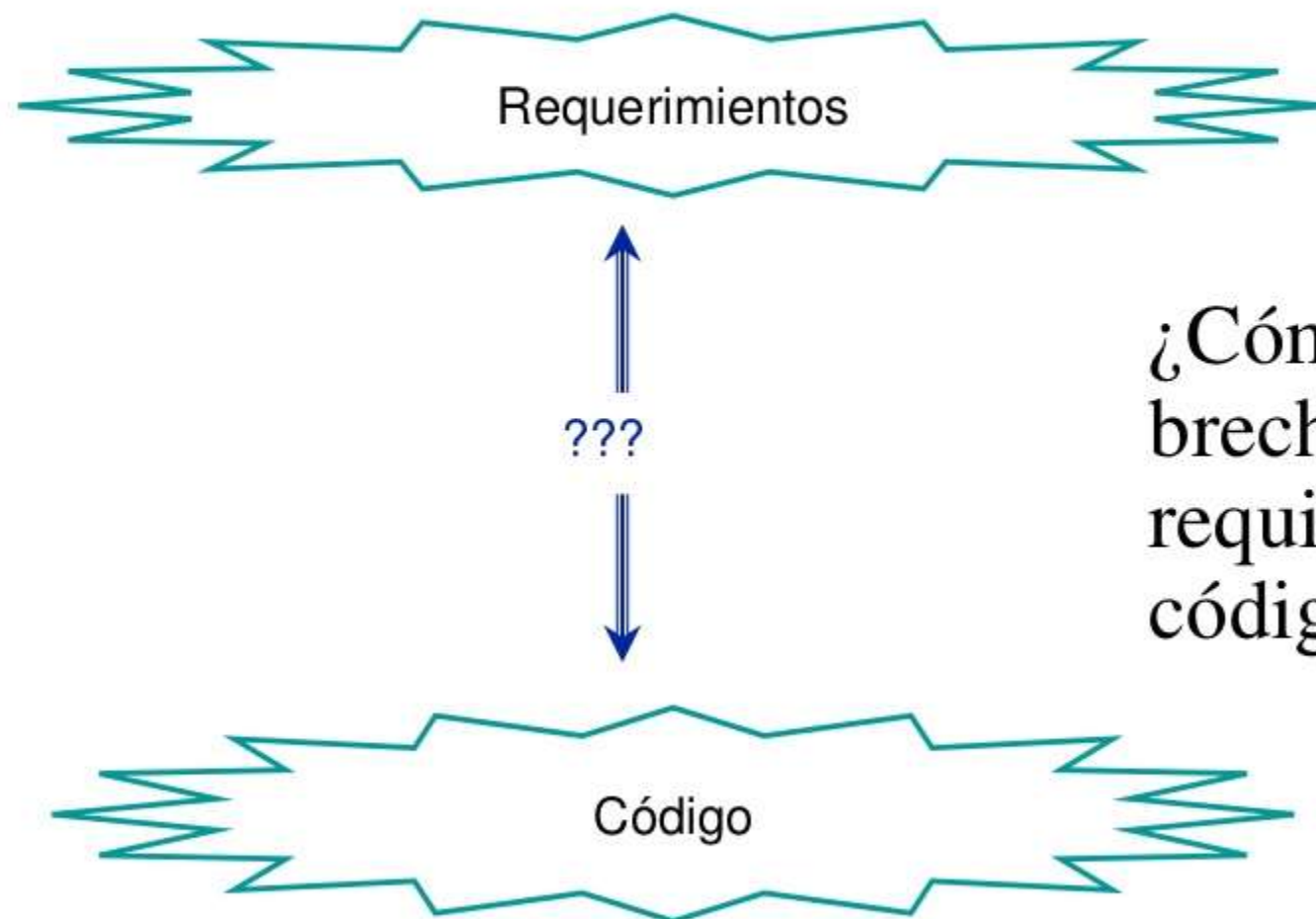
Objetivo general de la Unidad 1

Comprender el ciclo de vida de desarrollo de un software mediante la aplicación de estrategias de diseño orientados a funciones y objetos para construir software de alta calidad.

Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

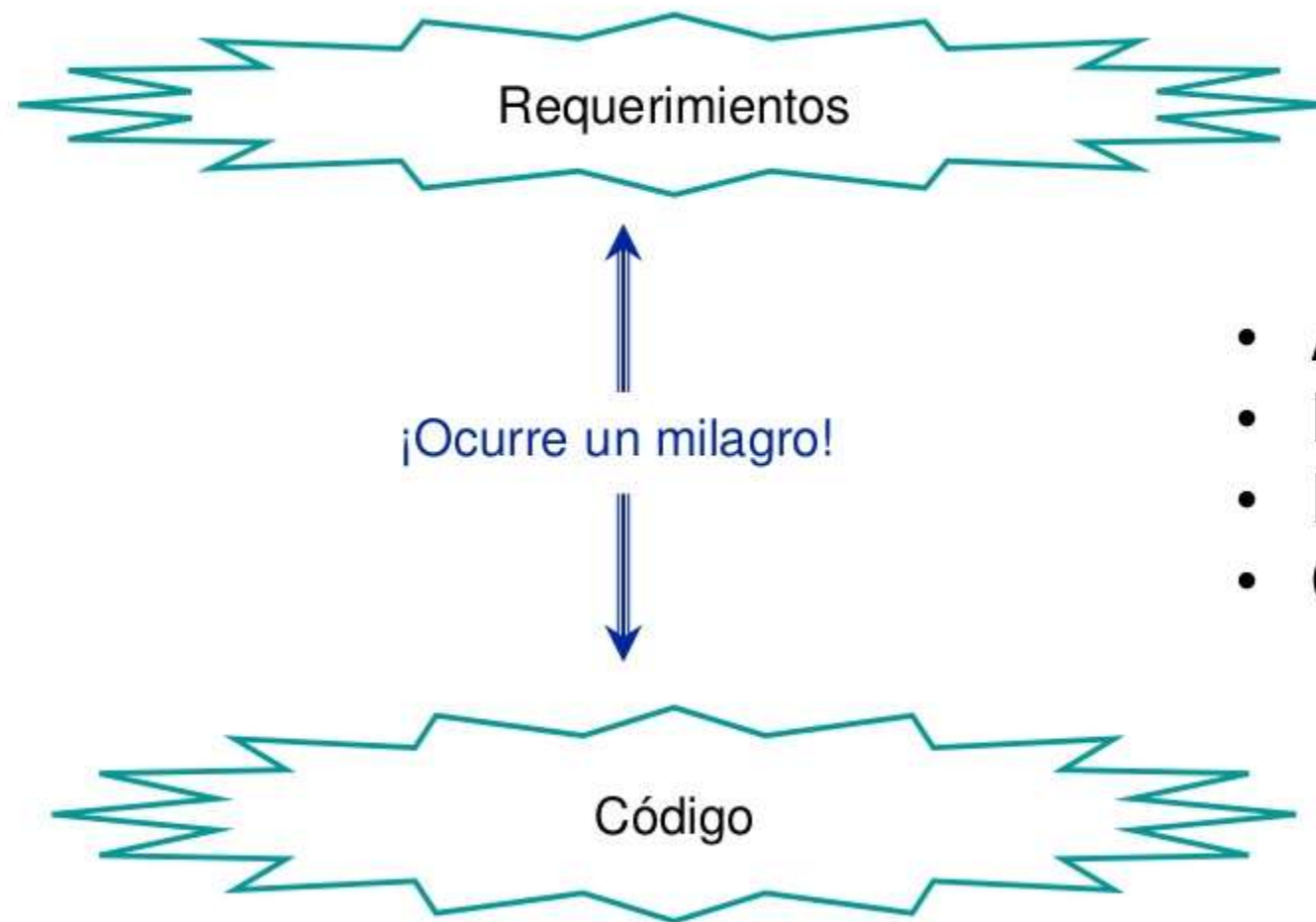
- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

El Problema



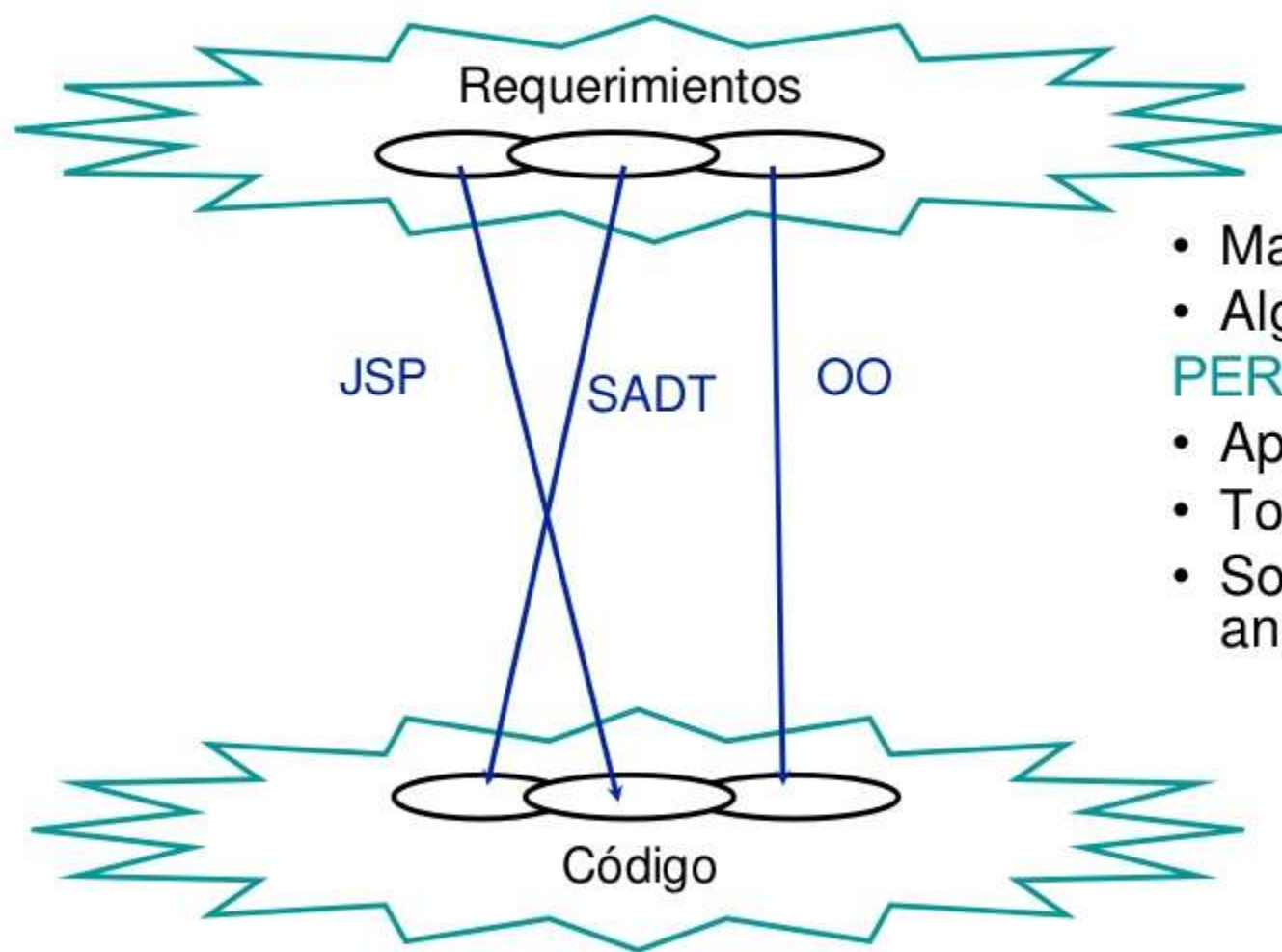
¿Cómo cerrar la brecha entre los requisitos y el código?

La respuesta tradicional



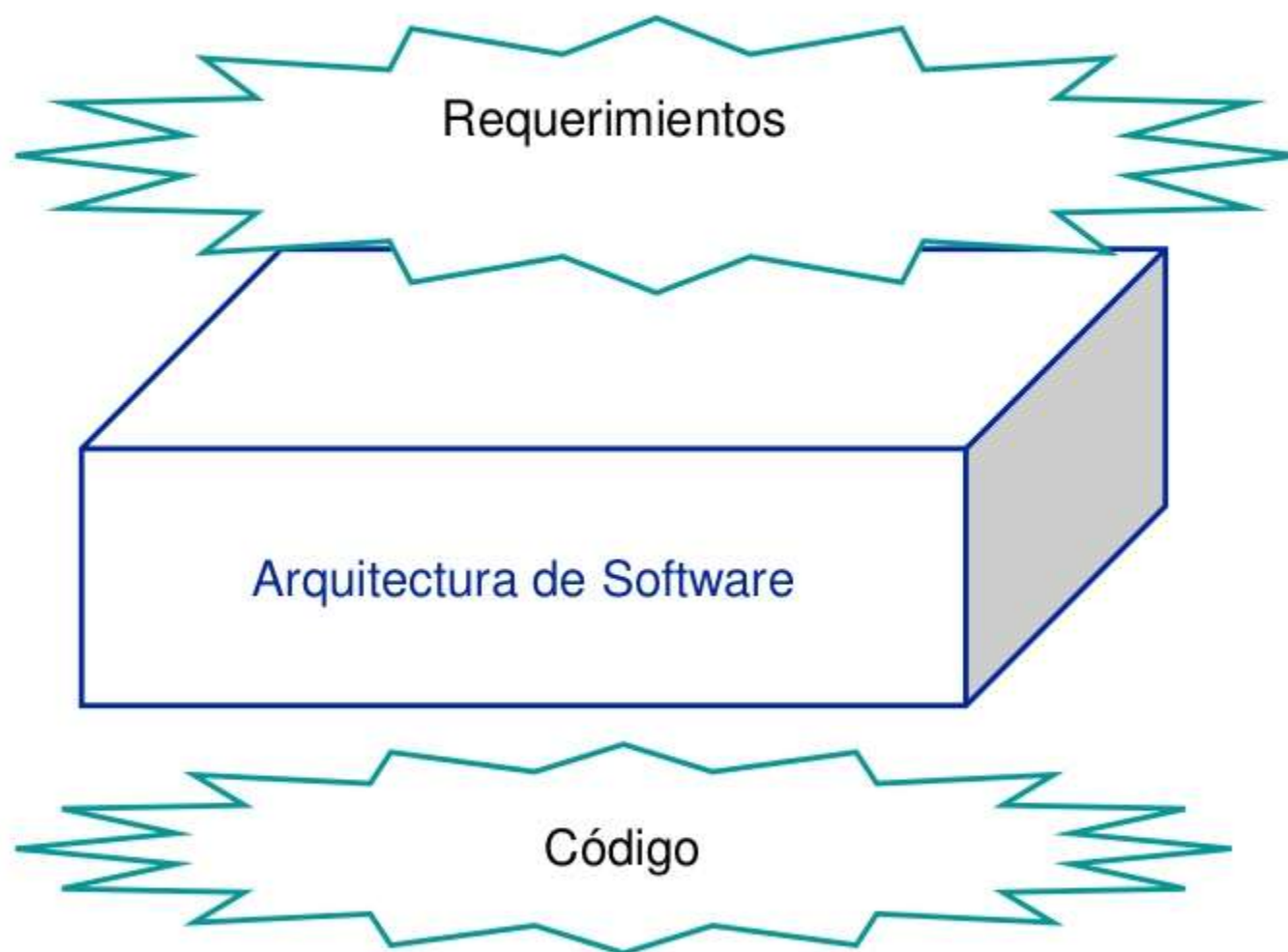
- Ad hoc
- Requiere gurús
- Impredecible
- Costoso

Métodos de desarrollo de software



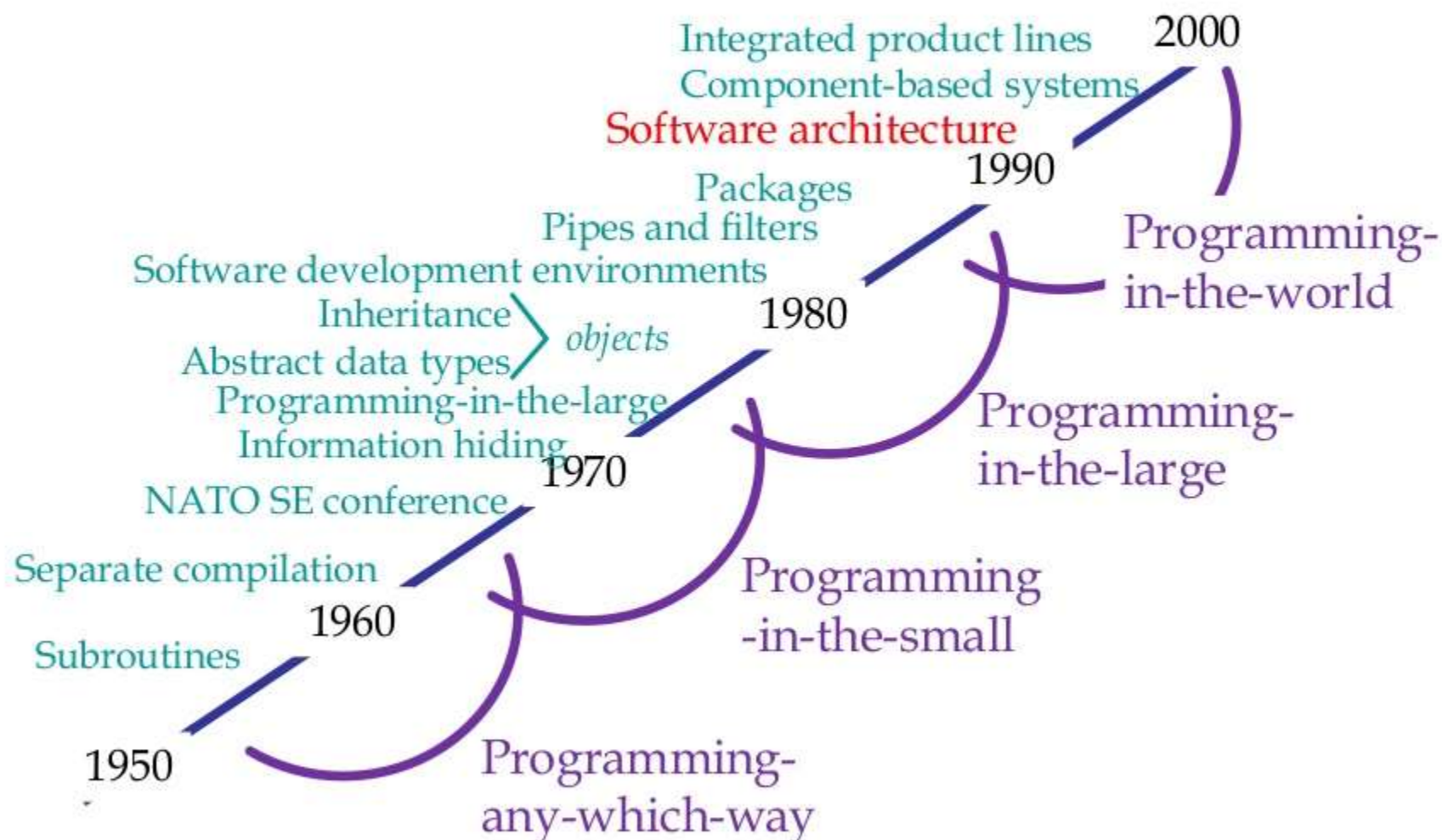
- Mayor previsibilidad
 - Alguna guía de diseño
- PERO...**
- Aplicabilidad limitada
 - Todavía requiere gurús
 - Soporte débil para el análisis de diseño

El papel de la arquitectura de software



- Composición de componentes a gran escala
- Abstracciones a nivel de sistema
- Reutilización de modismos (idioms) de diseño a nivel de sistema

Software Architecture in Context



Evolución del campo

- **Hoy**

- Reconocimiento del valor de los arquitectos en las organizaciones de desarrollo de software
- Procesos que requieren revisiones de diseño arquitectónico y documentación arquitectónica explícita
- Uso emergente de arquitecturas de líneas de productos, estándares arquitectónicos comerciales, frameworks para integración de componentes
- Codificación de vocabulario, notaciones y herramientas para el diseño arquitectónico
- Libros / cursos sobre arquitectura de software

- **Desafortunadamente**

- Poco uso de modelos, herramientas y análisis formales
- Muchas brechas en nuestra comprensión sobre cómo modelar y analizar aspectos clave de la arquitectura de software.

Importancia del DAS



- ¿Te has fijado que la Informática recibe un trato diferente que el resto de ingenierías?
 - En general, a la gente le parece normal que un programa se cuelgue, pero no que un puente se caiga...
 - pero **¿y si se cayera algo tan vistoso como un puente por algún defecto del DAS?**

Importancia del DAS



Lanzamiento del Ariane 5 en la misión VA241
(Arianespace/CNES).

- El Vuelo Ariane 501 de la Agencia Espacial Europea quedó destruido solo 40 segundos después del despegue (4 de junio, 1996).
- El prototipo de cohete, con coste de 1 billón de dólares americanos, activó el protocolo de autodestrucción debido a un *bug* en el sistema de guiado de a bordo.

Crisis del software

- La Crisis del software se refiere a los problemas que, desde sus inicios, ha ido experimentando el software, muchas veces problemas de gran magnitud, debido, principalmente, a la mínima eficacia que presentan una gran cantidad de empresas al momento de realizar un software.
- Sin embargo, no fue hasta 1968 cuando en la primera conferencia elaborada por la OTAN (Organización del Tratado del Atlántico Norte), Friedrich L. Bauer habló por primera vez del conjunto de dificultades o errores ocurridos en la planificación, estimación de los costos, productividad y calidad de un software, o bien, lo que se conoce como la crisis del software, dicho término se le atribuyó a F. L. Bauer aunque ya había sido utilizado por Edsger Dijkstra en su libro The Humble Programmer.
- Para dar solución a los problemas que se presentaban en esta conferencia se creó una nueva rama de ingeniería, la ingeniería de software.

Crisis del software

- Tanto en sus inicios, como en la época actual, una gran cantidad de proyectos de software tuvieron diversos problemas con respecto al tiempo y presupuesto que se le había estimado, causando accidentes que más allá de costos, involucraban daños a propiedades, y en el peor de los casos, la muerte de personas.
- Algunos ejemplos son:
 - Accidente de un F-18 (1986): En abril de 1986 un avión de combate se estrelló por culpa de un giro descontrolado atribuido a una expresión “if then”, para la cual no había una expresión “else”, debido a que los desarrolladores del software lo consideraron innecesario.

Crisis del software

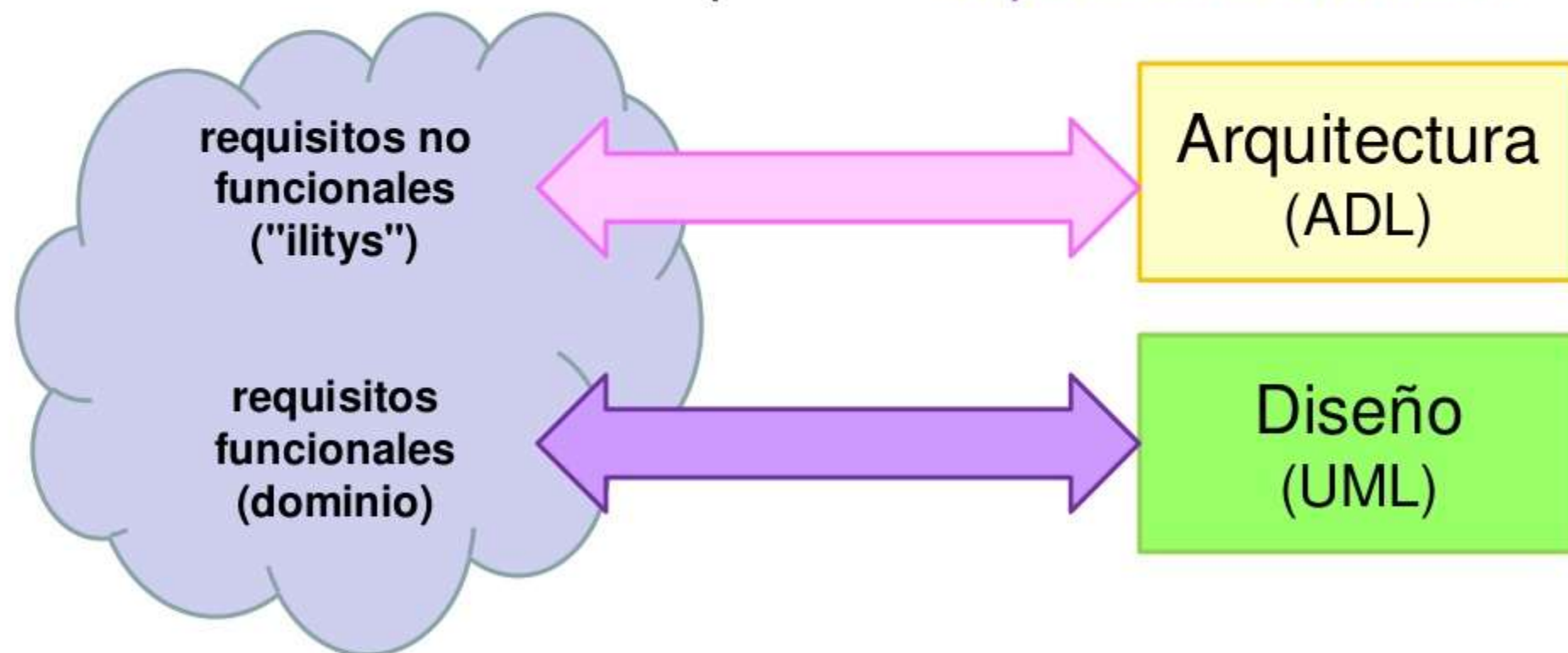
- Muertes por el Therac-25 (1985-1987): El Therac-25 fue una máquina de radioterapia que causó la muerte de varios pacientes en diversos hospitales de Estados Unidos y Canadá, debido a las radiaciones de alto poder aplicadas sin control, las cuales fueron atribuidas a la falta de control de calidad del software médico.
- Sobrecosto, retraso y cancelación en el sistema del Bank of America (1988): En el año de 1988, este banco invirtió 23 millones de dólares en un sistema computarizado llamado MasterNet, el cual servía para contabilidad y reportes de fideicomisos. No obstante, para que el sistema funcionara, se tuvo que invertir 60 millones de dólares más, por lo que finalmente el sistema fue cancelado.⁴

Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Arquitectura vs. Diseño

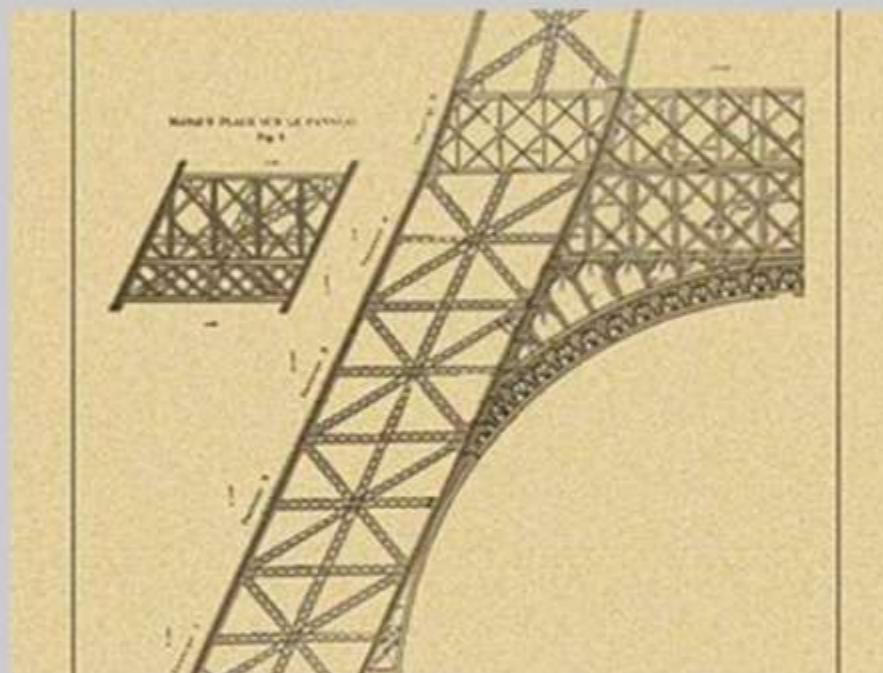
- **Arquitectura:** donde se plasman las **decisiones no funcionales** y se dividen los **requisitos funcionales**
- **Diseño:** donde se cumplen los **requisitos funcionales**



Atributos de la calidad del Software

("ilitys")

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability



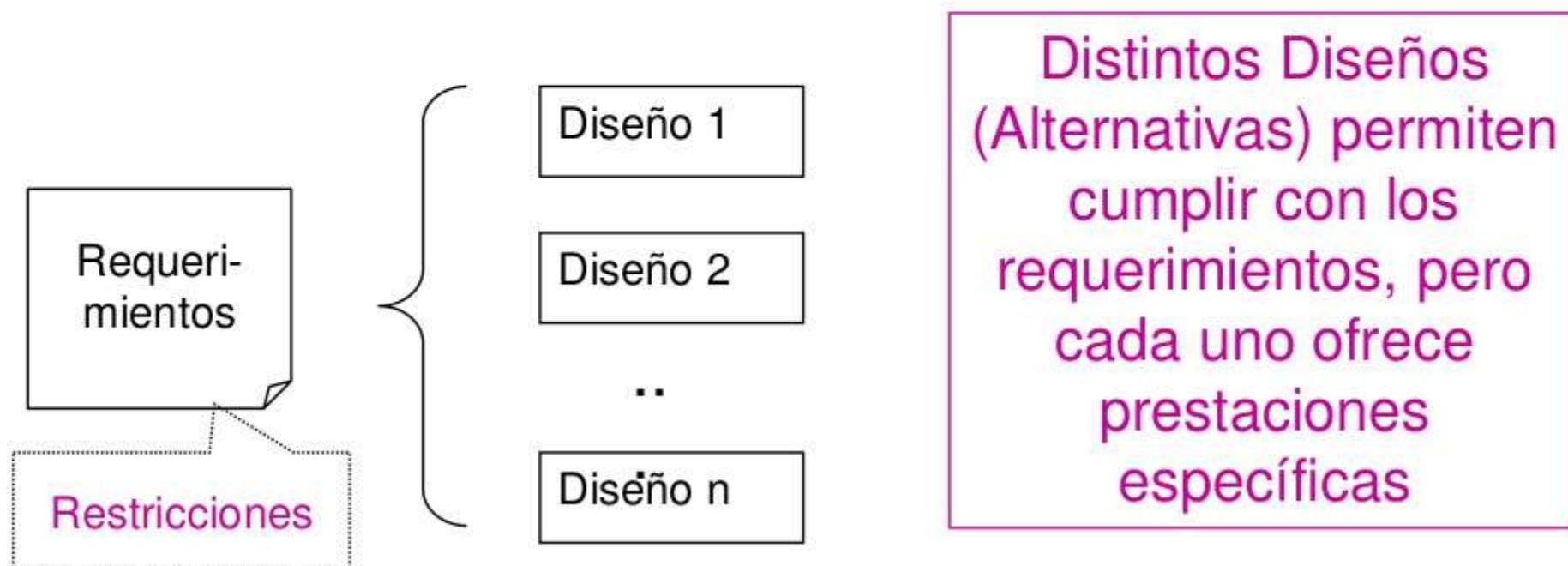
Diseño

Diseño - ¿qué es?

- El diseño es el proceso creativo de transformación del problema en una solución; la descripción de una solución también se denomina diseño.
- La solución será la que satisface todos los requerimientos planteados en la especificación de requerimientos.
- Las soluciones en muchos casos son ilimitadas

Diseño - ¿qué es?

- Significado:
 - **Proceso** por el que se genera una solución a un problema
 - **Descripción** de la solución



Diseño de software

- El diseño es la primera parte del desarrollo de cualquier proyecto.
- Etimológicamente significa componer, por lo que se obtiene la solución que habrá de implantarse.
- Todas las cosas siempre tienen primero una creación mental.
- El proceso de diseño sirve de base para la codificación del sistema. Se deben seguir algunas recomendaciones para su mejor desarrollo como las siguientes:
- Se deben especificar todos los elementos explícitos e implícitos del modelo de análisis.

Diseño de software

- El diseño deber servir de guía para que cual integrante del proyecto pueda construir y entender el software.
- El diseño debe de dar una completa idea de lo que es el software.
- El diseño debe presentar uniformidad e integración. Se deben definir reglas y estilos que deben seguir los miembros del equipo.
- El diseño debe estar estructurado, de tal forma que permita cambios.
- El diseño no es escribir código, ni codificar, **es diseñar**.

Apartados del Diseño de software

El diseño en proyectos informáticos presenta cuatro apartados: datos, arquitectura, interfaz y procedimientos.

- **El diseño de datos** se encarga de transformar el modelo de información obtenido en el proceso de análisis en estructuras de datos. Se pueden utilizar diagramas entidad-relación pero especificados a mayor detalle.
- **El diseño arquitectónico** tiene la finalidad de comprobar las relaciones con los diferentes módulos o macrorequisitos del sistema (subsistemas).
- **El diseño de interfaz** define como se comunica el software consigo mismo y hacia el exterior.
- **El diseño procedimental o basado en componentes** consiste en la traducción de cada uno de los elementos obtenidos en la especificación de procesos, datos y transición hacia elementos implantables a través de computadoras.

Resumen: Diseño de Software

- Características:
 - Proceso Creativo
 - Requiere de experiencia e ingenio
 - Necesita del aprendizaje a través del estudio de sistemas existentes

- Etapas:
 - Estudiar y entender el problema
 - Conseguir mas de una solución factible
 - Describir cada abstracción usada en la solución

Objetivos

- El diseño de un sistema es **correcto** si un sistema construido de acuerdo a ese diseño satisface los requerimientos del sistema
- Pero el objetivo del diseño no es encontrar **el** diseño correcto sino
 - Encontrar el mejor diseño posible
 - dentro de las limitaciones impuestas por
 - los requerimientos,
 - el ambiente físico del sistema
 - y el ambiente social donde el mismo va a operar
 - ¿otras limitaciones?

El Diseño Debe Ser

- Un diseño claramente debe ser:
 - Verificable
 - Completo (implementa toda la especificación)
 - “Rastreable”. Se puede rastrear hacia los requerimientos que diseña (“*traceable*”)
- Sin embargo, las dos cosas más importantes concernientes a los diseñadores es que el diseño sea:
 - Eficiente
 - Simple
 - ¿Por qué?

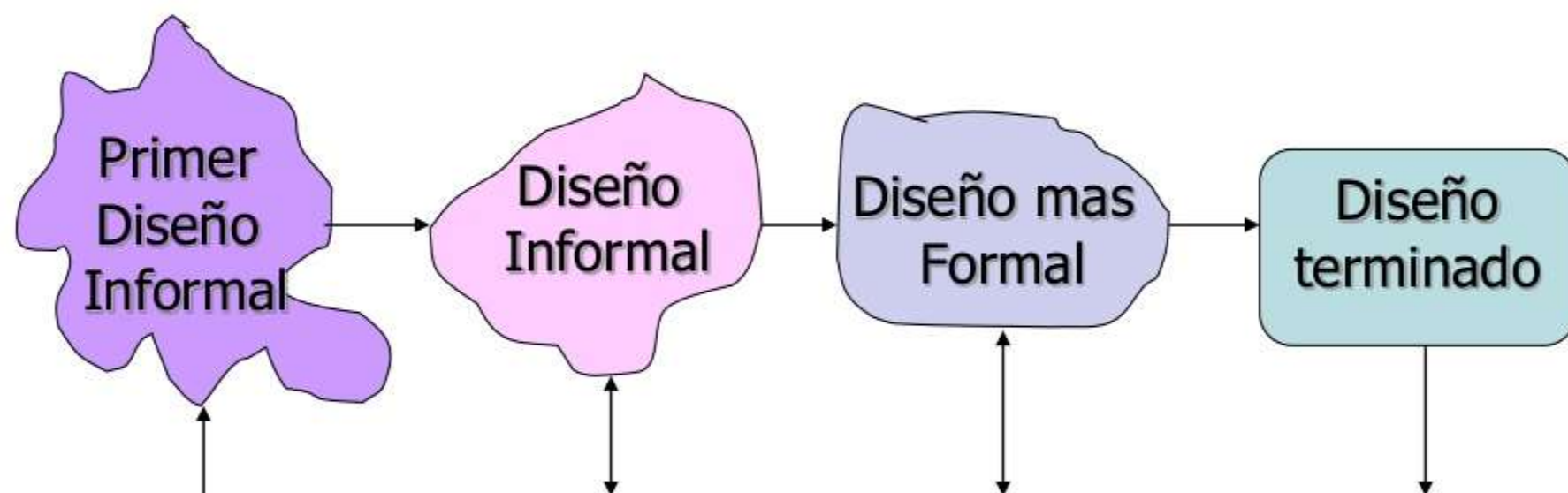
Estas dos propiedades están normalmente encontradas. Soluciones de compromiso.

Y normalmente, ¿por cuál se inclinarían?

La Tarea de Diseño

- Crear un diseño simple y eficiente de un sistema “grande” puede ser una tarea extremadamente compleja
 - Actividad básicamente creativa
 - No puede reducirse a una serie de pasos a seguir
 - Sin embargo, se pueden dar guías
- Si se logra manejar la complejidad
 - Se reducen los costos del diseño
 - Se reduce la posibilidad de introducir defectos durante el diseño

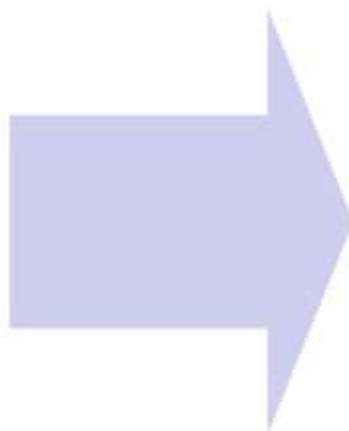
El Proceso de Diseño



Objetivos del Proceso de Diseño

- Formalizar la especificación
- Describir el sistema bajo distintos niveles de abstracción
- Especificar como se comporta el sistema y como ejecuta las funciones levantadas en la especificación

Descomposición



Errores y Omisiones
son descubiertos en las
primeras etapas del
diseño.

Las Actividades del Proceso de Diseño

- **Diseño Arquitectural.** Se definen los subsistemas y sus relaciones y se documentan
- **Especificación Abstracta.** Para cada subsistema se definen sus funcionalidades y las restricciones sobre las cuales debe operar
- **Diseño de la Interfaz.** Para cada subsistema se diseña y documenta su interfaz con otros subsistemas
- **Diseño de Componentes.** Los servicios que provee el subsistema son particionados en componentes
- **Diseño de la Estructura de Datos.** La estructura de data a ser implementada en el sistema es diseñada en detalle y documentada
- **Diseño Algorítmico.** Los algoritmos usados para proveer los servicios son diseñados en detalles y documentados

Las Actividades del Proceso de Diseño

Estas Actividades son realizadas con cada subsistema, hasta que los componentes identificados pueden ser llevados directamente a un lenguaje de programación.

La práctica recomendada:

- Diseño Estructurado: TOP - DOWN
- Diseño Orientado a Objetos: Componentes Reusables

Niveles de Diseño

- (1) Arquitectura:

Requerimientos => componentes del sistema y sus interconexiones

- (2) Diseño del Código:

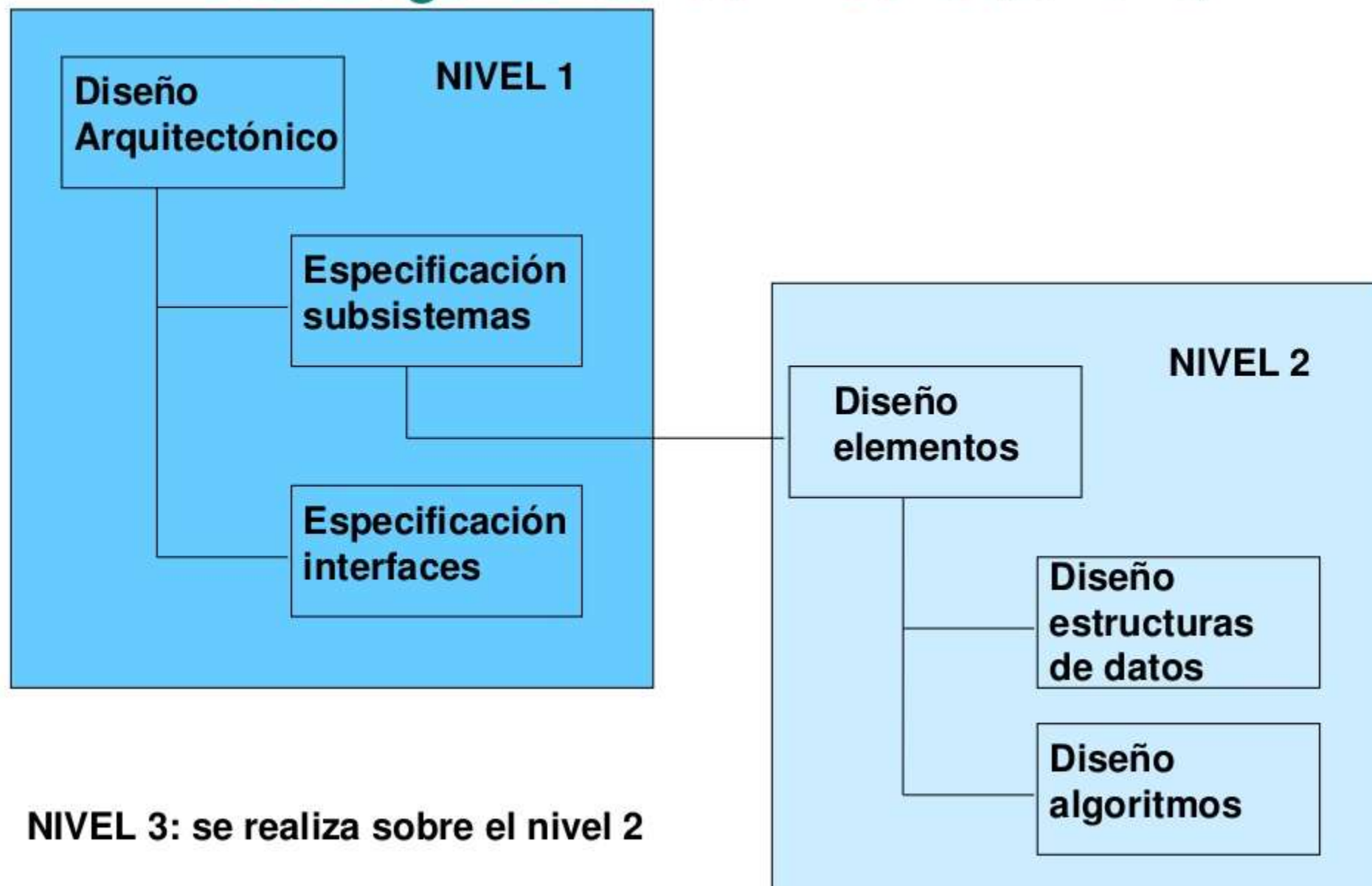
Módulos => algoritmos y estructuras de datos

- (3) Diseño de la Ejecución:

Algoritmos (código) => asignación de memoria, tiempo de ejecución, optimizaciones de código

ENFOQUE: trabajar desde lo general a lo particular

Proceso genérico de Diseño (Sommerville)



Factores de Calidad del diseño

- Al diseñar se deben tomar en cuenta
 - Factores de Calidad Externos
 - Velocidad
 - Fiabilidad
 - Utilidad
 - Factores de Calidad Internos
 - Abstracción
 - Refinamiento
 - Modularidad

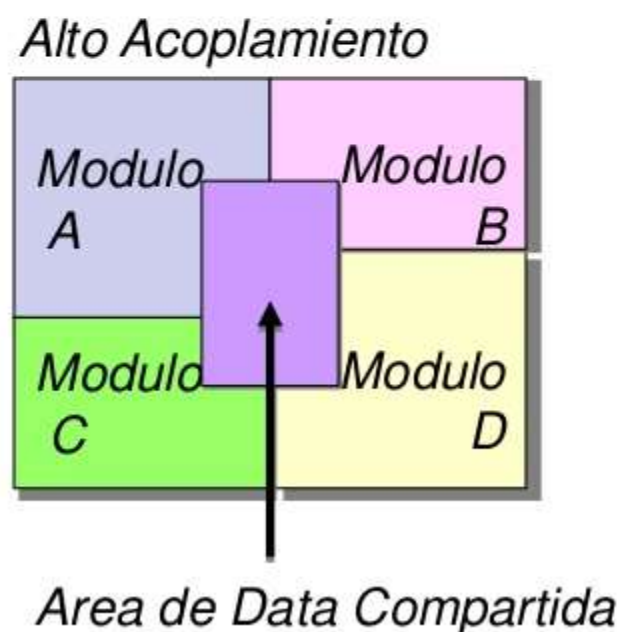
Conceptos de Diseño

- Tres conceptos claves para la calidad interna de un diseño (además de ser correcto, claro)
 1. Acoplamiento (bajo)
 2. Cohesión (alta)
 3. Principio abierto-cerrado (cumplir con el principio)

Acoplamiento

Indica la fuerza de interconexión entre las unidades de programa.

Como regla general, los módulos son altamente acoplados si hacen uso de variables compartidas o si intercambian información de control.



Acoplamiento

- Captura la fuerza de interconexión entre módulos
- Cuánto más acoplados más dependientes entre si
 - Entonces, más difícil comprenderlos y modificarlos
- El grado de acoplamiento entre módulos depende de:
 - Cuánta información se necesita sobre el otro módulo para entenderlo
 - Y qué tan compleja
 - Y explícita es esta información

La menor posible

Simple

Fácilmente visible o
identificable

Tipos de Acoplamiento

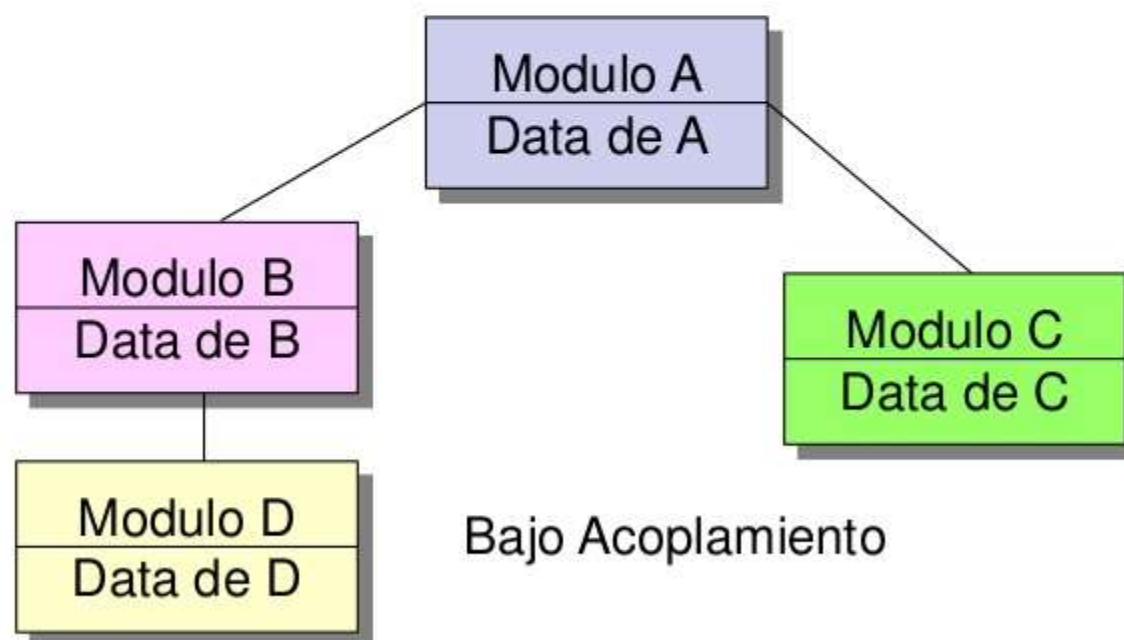
- Interacción
 - Métodos de una clase que invocan a métodos de otra clase
 - La peor forma es si se interactúa con partes internas del método
 - La del “medio” es si se accede directamente a atributos del objeto
 - La menor es si se accede al método mediante intercambio de parámetros

Tipos de Acoplamientos (2)

- Componente
 - Una clase tiene variables de otra clase
 - Tres formas:
 - Existe un atributo de la clase que es de otra clase
 - Existe un método que recibe como parámetro otra clase
 - Existe un método con una variable local de otra clase. Esta es la menos deseada
- Herencia

Acoplamiento

Bajo acoplamiento es el preferido en un buen diseño para asegurar que la información de representación está atada a una componente y que su interfaz de datos con otras unidades es vía una lista de parámetros.



Conceptos de Diseño

- Tres conceptos claves para la calidad interna de un diseño (además de ser correcto, claro)
 1. Acoplamiento (bajo)
 2. Cohesión (alta)
 3. Principio abierto-cerrado (cumplir con el principio)

Cohesión

- Se focaliza en conocer porqué los elementos de un módulo están juntos en ese módulo
- El objetivo es tener en un mismo módulo elementos que están fuertemente vinculados
 - Entonces, módulos más fáciles de entender
 - y más fáciles de modificar

Tipos de Cohesión

- Sin entrar en mucho detalle
- Método
 - Más alta cohesión cuando el método implementa una única función claramente definida
 - y todas las sentencias contribuyen a esa función
- Clase
 - Se focaliza en porqué distintos atributos y métodos están en una misma clase
 - Una clase implementando un único concepto o abstracción
 - Y todos sus elementos contribuyendo a soportar ese concepto o abstracción

Tipos de Cohesión (2)

- Herencia
 - Se focaliza en conocer porqué las clases están agrupadas en una jerarquía
 - Se considera que la cohesión es baja si la jerarquía es usada sólo por reuso de código
 - y no existe una relación de generalización-especialización

Ser Entendible

Los cambios en el diseño de un componente implican que la persona responsable de hacer el cambio entienda la operación de dicho componente.

Este entendimiento está atado a unas características:

- **Cohesión**, Puede el componente ser entendido sin referencias de otro componentes?
- **Nombre**, Los nombres de los componentes son significativos?
- **Documentación**, La documentación de la componente muestra una relación clara entre las entidades del mundo real y la componente?
- **Complejidad**, Cuan complejos son los algoritmos para implementar la componente?

Adaptabilidad

- Si un diseño va a ser mantenido, este debe ser adaptable. Esto implica, que sus componentes deben ser de bajo acoplamiento.
- La adaptabilidad significa, que el diseño debe estar muy bien documentado, que la documentación sea entendible y consistente con la implementación y la implementación debe estar escrita en una forma leíble.
- Un diseño adaptable debe tener alto nivel de visibilidad y deben estar determinada las relaciones entre los diferentes niveles.
- Debe ser fácil incluir modificaciones tanto al diseño como al documento de diseño.
- Los componentes deben ser autocontenidos. Una componente debe ser de bajo acoplamiento y solamente cooperar con otras componentes bajo el paso de mensajes.

Conceptos de Diseño

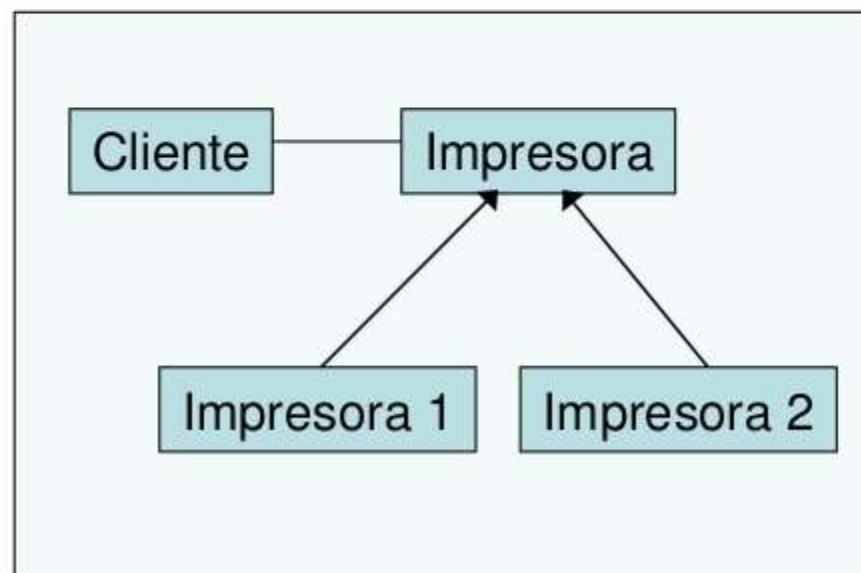
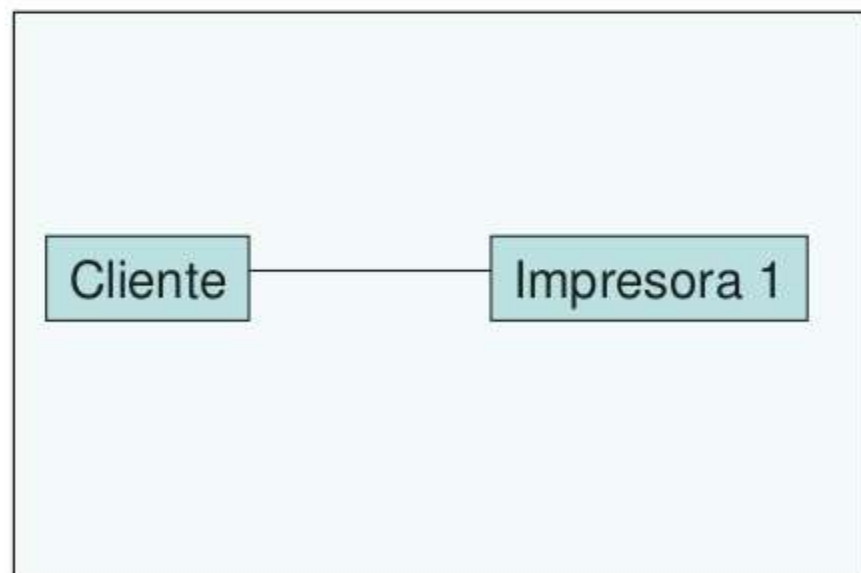
- Tres conceptos claves para la calidad interna de un diseño (además de ser correcto, claro)
 1. Acoplamiento (bajo)
 2. Cohesión (alta)
 3. Principio abierto-cerrado (cumplir con el principio)

Principio Abierto-Cerrado

- Objetivo (otra vez): promover la construcción de sistemas que sean fáciles de modificar
- Módulos **abiertos** para la **extensión**
 - El comportamiento puede ser extendido
- Módulos **cerrados** para la **modificación**
 - Extremo: cuando se hacen mejoras no es necesario tocar el código existente
 - La interfaz no debe cambiar y se debe asegurar que se mantiene el comportamiento

Principio Abierto-Cerrado (2)

- En OO el principio se puede alcanzar con un buen uso de la herencia
 - Este permite extender una clase sin modificar el código de esa clase



Principios de Diseño

- Algunos principios de diseño a tener en cuenta:
 - Dividir y conquistar
 - Abstracción
 - Modularidad

Dividir y Conquistar (1)

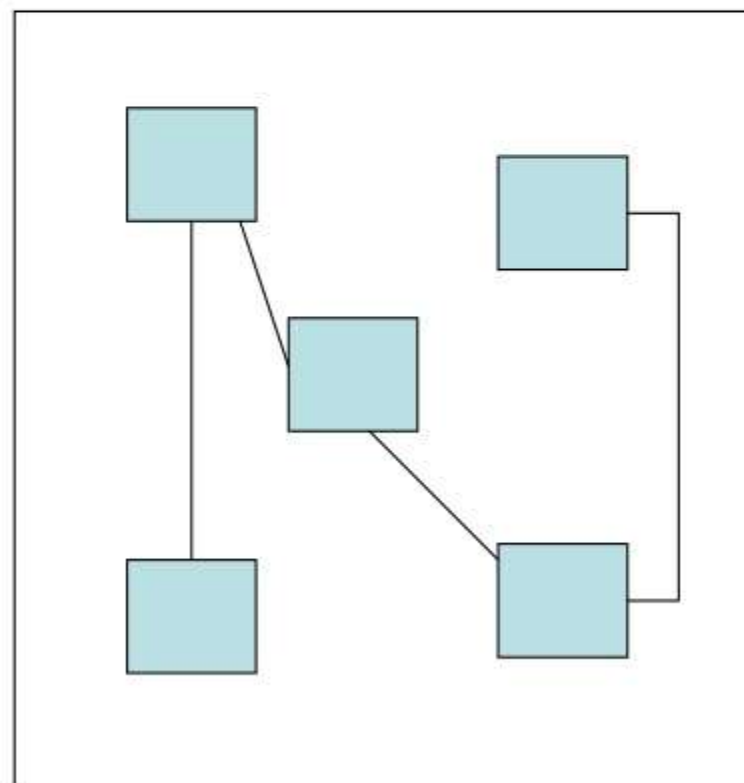
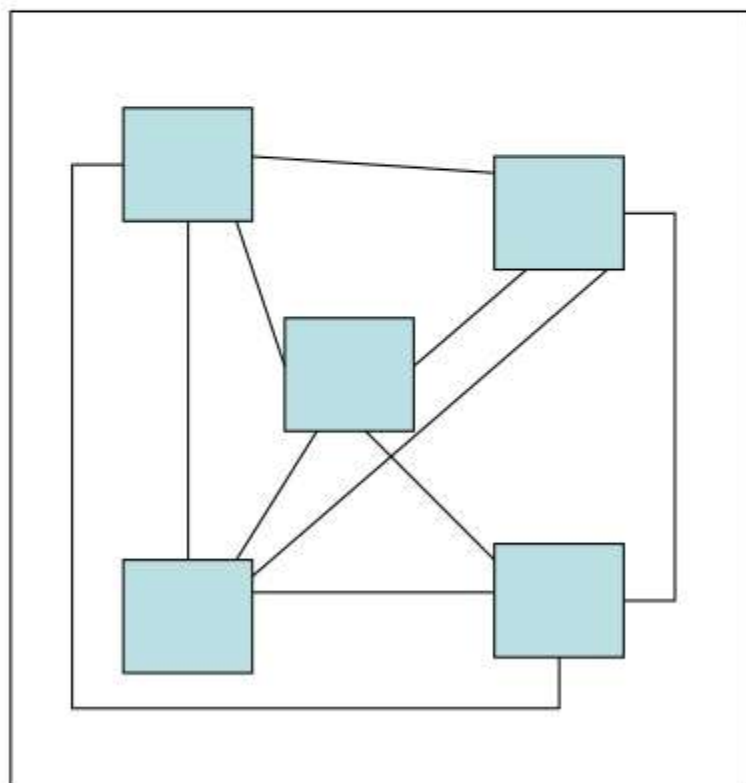
- Debido a la complejidad de los grandes problemas y las limitaciones de la mente humana estos no se pueden atacar como una unidad monolítica
 - Aplicar dividir y conquistar
 - Dividir en piezas que pueden ser “conquistadas” por separado. Sino se hizo una división poco inteligente
- Dividir en piezas con esta propiedad es una tarea compleja para el diseño de software

Dividir y Conquistar (2)

- Las piezas
 - Están relacionadas. Juntas forman el sistema
 - Entonces, existe comunicación y cooperación entre las mismas
 - Esto agrega complejidad, que surge de la propia partición
 - Cuando el costo de particionar sumado la complejidad agregada supera los ahorros logrados por particionar se debe detener el proceso

Dividir y Conquistar (3)

- ¿De qué sirve lograr la mayor cantidad de piezas independientes respecto a otras?



Principios de Diseño

- Algunos principios de diseño a tener en cuenta:
 - Dividir y conquistar
 - Abstracción
 - Modularidad

Abstracción

- Permite al diseñador considerar una componente a un cierto nivel de abstracción sin preocuparse por los detalles de implementación de dicha componente
 - Se describe el comportamiento exterior sin describir los detalles internos
- ¿Cómo se relaciona con el proceso de partición?

Abstracción procedimental

- En un diseño descendente, un modulo
 - Equivale a una “caja negra” y ejecuta una tarea determinada
- Cada “caja negra” o procedimiento
 - Nos interesa **lo que hace**, no *como lo hace*
 - Puede conocer otras cajas negras y usarlas
- Un procedimiento/modulo ya escrito
 - Se puede usar sin saber como esta escrito
 - Basta con su nombre y una descripción

Abstracción de datos

- Al analizar un problema, podemos centrarnos en:
 - Los datos que vamos a necesitar y
 - Las operaciones que se pueden realizar sobre los mismos
 - Definirán sus TIPOS DE DATOS ABSTRACTOS o CLASES
- TDAs
 - Los módulos los conocerán y usarán
 - Sin interesarles como se escribieron sus operaciones
 - Se pueden implementar usando estructuras de datos
- Para diseñar una solución se deben desarrollar
 - Algoritmos y
 - TDAs

Principios de Diseño

- Algunos principios de diseño a tener en cuenta:
 - Dividir y conquistar
 - Abstracción
 - Modularidad

Modularidad

- Un sistema se considera modular si
 - El sistema consiste de componentes
 - y estas se pueden implementar separadamente
 - y el cambio en una tiene mínimos impactos en otras componentes
- La modularidad es donde se juntan la abstracción y la partición

Modularización (1)

- Módulo: un componente bien definido de un sistema de software que provee recursos y servicios
 - Están bien definidos:
 - Recursos y Servicios
 - La forma de suministrarlos (Interfaces)
 - Un módulo puede ser un programa o varios, una subrutina o varias
- Principio de Separación de Intereses
 - cada módulo se ocupa de aspectos específicos

Modularización (2)

- Definir módulos e interfaces
 - Interfaces definen el acoplamiento entre módulos
 - Comportamiento (Pre-Post condiciones) y colaboraciones
- Ocultamiento de Información
 - La información de un módulo debe ser privada a menos que se declare pública, única garantía de que otros módulos no la utilicen (- impacto de cambios, + fácil de comprender y usar)
 - Diseño interfaz: ¿Qué servicios brindar, qué ocultar?
 - Mínima, bien definida, independiente de implementación
- Alta Cohesión – Bajo acoplamiento
 - Alta cohesión interna de cada módulo (los elementos del módulo están fuertemente relacionados entre sí)
 - Bajo acoplamiento entre módulos (débiles conexiones entre módulos -impacto reducido de cambios)

Categorías de módulos

- Sin persistencia (estado)
 - Abstracciones de procedimientos (algoritmo)
 - Bibliotecas (ej. rutinas matemáticas)
- Sólo Datos
 - Repositorio común de datos (ej. Configuración)
- Algoritmos + Estado
 - Objetos abstractos (ej. Stack)
- Tipos Abstractos de Datos
 - paramétrico en tipo (ej. Stack (?))
- Clases (OO)

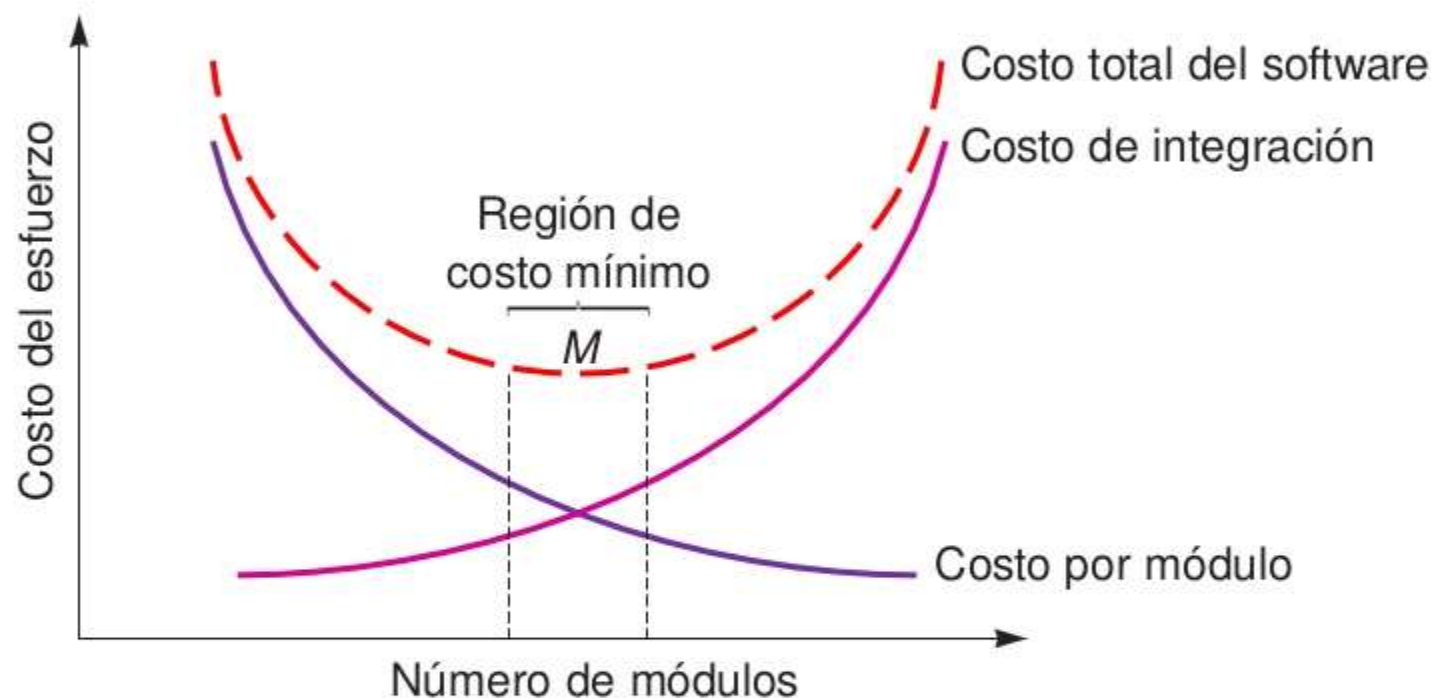
Cuando modularizar

- Si un conjunto de sentencias realiza una tarea
 - Recurrente, repetitiva, identificable
- Debe ser un módulo
- Sin embargo,
 - Una tarea no necesita ser recurrente para hacerla un módulo

Criterios para modularizar

- Descomposición
 - Descomponer el problema en sub-problemas (diseño top-down)
- Composición
 - A partir de los componentes es posible obtener un nuevo sistema (diseño bottom-up)
- Continuidad del impacto por cambios
 - Pequeños cambios en la especificación afectan pocos módulos
- Protección durante ejecución
 - Efectos de anomalías durante la ejecución están localizados

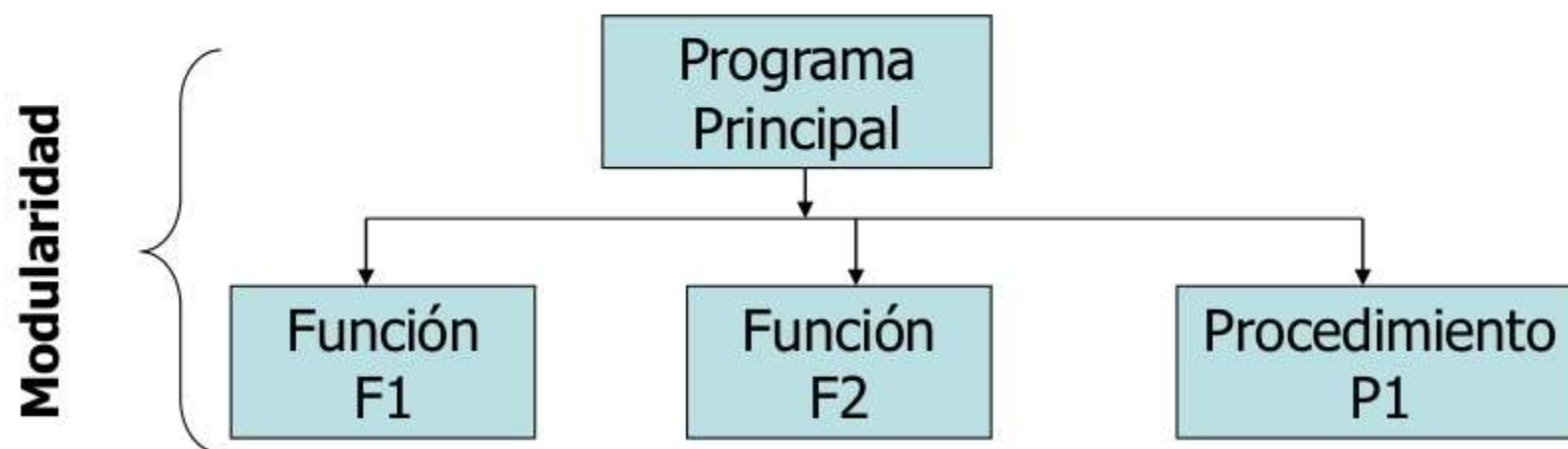
Descomposición



Deben hacerse módulos, pero con cuidado para **permanecer en la cercanía de M** . Debe evitarse hacer pocos o muchos módulos.

Diseño descendente (top-down)

- Permite abordar problemas complejos
- Descomponer el problema grande en varios pequeños
 - Módulos independientes
 - Procedimientos, funciones y bloques de código



Modularización - Beneficios

- Separar aspectos del diseño
 - Permitir cambiar algunos sin afectar al resto
- Permite
 - Extender fácilmente artefactos (extensibilidad)
 - Reusar artefactos existentes (reusabilidad)
 - Ocultar dependencias de la plataforma (portabilidad)
 - Diseñar interfaces que se adapten a otras (compatibilidad)
 - Minimizar impacto de cambios (mantenibilidad)
 - Facilitar la comprensión
 - Organizar y distribuir el trabajo
- **Alta modularización** => Alta cohesión – Bajo acoplamiento

Características de un buen diseño

- Los diseños de calidad superior deben tener características que dan como resultado productos de calidad: facilidad de comprender, de implementar, de probar, de modificar y la correcta traducción de la especificación de requerimientos.
- Atributos que reflejan la calidad del diseño:
 - Independencias de los componentes.
 - Identificación y tratamiento de excepciones.
 - Prevención de defectos y tolerancia de defectos.

Independencia de los componentes

- En la mayoría de los diseños se trata de que los componentes sean independientes unos de otros. Ya que, un componente independiente es mucho más fácil de modificar.
- Para reconocer y medir el grado de independencia de los componentes de un diseño se aplican dos conceptos: **acoplamiento** y **cohesión** (Yourdon y Constantine 1978).

Características de un buen diseño

- Los diseños de calidad superior deben tener características que dan como resultado productos de calidad: facilidad de comprender, de implementar, de probar, de modificar y la correcta traducción de la especificación de requerimientos.
- Atributos que reflejan la calidad del diseño:
 - Independencias de los componentes.
 - Identificación y tratamiento de excepciones.
 - Prevención de defectos y tolerancia de defectos.

Identificación y tratamiento de excepciones

- El diseño debe realizarse defensivamente, tratando de anticiparse a situaciones que podrían derivar en problemas en el sistema.
- No es fácil diseñar defensivamente, ya que la especificación dice lo que debe hacer el sistema no lo que no debe.
- Para poder hacer un diseño defensivo se debe hacer un **manejo de excepciones**.
 - **Excepción**: son situaciones que se saben contrarias a lo que se espera realmente que haga el sistema.
- El manejo de excepciones debe permitir que el sistema dirija cada excepción en una forma satisfactoria que no degrade las funciones del sistema.

Identificación y tratamiento de excepciones

Excepciones más comunes:

- Fracaso al proporcionar un servicio.
- Proporcionar un servicio o dato erróneo.
- Corrupción de datos.

Las excepciones se pueden manejar según uno de los siguientes criterios:

- ✂ **Corregir:** se restaura el sistema a su estado previo y se intenta realizar el servicio utilizando una estrategia diferente.
- ✂ **Reintentar:** se restaura el sistema a su estado previo y se intenta realizar el servicio usando la misma estrategia.
- ✂ **Informar:** se restaura el sistema a su estado previo, se informa el problema a un componente de tratamiento de error y no se proporciona el servicio.

Corregir

[Yes](#)

[No](#)

zlibrary

Part of Z-Library project. The world's largest ebook library

New post ["New options: "book requests" and save your search filters"](#) in our blog.

[General Search](#)

[Fulltext Search](#)

[Search](#)

[Search options](#)

[Books \(500+\)](#)

[Articles](#)

[Most Popular](#)

[List](#)

The books listed below don't fit your search query exactly but very close to it.
Do you want to subscribe on this query using ZAlerts?

[Diane Franklin: The Excellent Adventures of the Last American, French-Exchange Babe of the 80s](#)

Diane Franklin

Diane Franklin, Michael Picarella, Savage Steve Holland

Year: 2012 Language: english File: EPUB, 1.65 MB

Reintentar

The screenshot shows a Gmail interface with a connection error message highlighted in a red box. The message reads: "Sin conexión. Se volverá a intentar en 1 s. [Intentar ahora](#)".

The interface includes the Gmail logo, a search bar, and a left sidebar with navigation options: Redactar, Recibidos (304), Destacados, Pospuestos, Enviados, Borradores, Más, Meet (Nueva reunión, Unirse a una reunión), and Hangouts (Raquel, Intentando volver a conectar...). The main inbox area shows a list of emails under the 'Principal' tab, including PDFs and research papers. The bottom of the page displays links for Términos, Privacidad, Política del programa, and the last activity date (23/11/20).

Principal	Social 50 nuevos	Promociones
☐ ☆ PDF By Israel Celi .	☐ "Investigación Jurídica" by Israel Celi Toledo - You read the paper VIII Investigación Guía didáctic...	16:59
☐ ☆ PDF By Pedro Villan.	☐ "UNIVERSIDAD PARA LA COOPERACION INTERNACIONAL (UCI) DISEÑO DEL SISTEMA INFORM...	20 feb
☐ ☆ PDF By Karina Vargas	☐ "UNIVERSIDAD TÉCNICA DE AMBATO UNIVERSIDAD TÉCNICA DE AMBATO UNIVERSIDAD TÉCN...	19 feb
☐ ☆ PDF By Wysky Myke	☐ "VIII Investigación Guía didáctica 5 créditos Departamento de Psicología Sección Psicología Clí...	18 feb
☐ ☆ PDF By Ivonne Marti.	☐ "UNIVERSIDAD DEL BÍO-BÍO MANUAL PARA EL DISEÑO DE PROYECTOS DE GESTIÓN EDUCACIO...	17 feb
☐ ☆ PDF By Erick Martin.	☐ "222El aseguramiento" by Erick Martinez - You read the paper PRELIMINARES Guía didáctica: Dis...	16 feb
☐ ☆ PDF By Elena Malo	☐ "Expresión Plástica I" by Elena Malo - You read the paper PRELIMINARES Guía didáctica: Diseñ...	15 feb
☐ ☆ PDF By Fausto Mosco.	☐ "tesis de maestria" by Fausto Moscoso - You read the paper PRELIMINARES Guía didáctica: Dise...	14 feb
☐ ☆ PDF By Gabriel Espi.	☐ "Instructivo Planificaciones Curriculares20191011 106584 yh60ul" by Gabriel Espinoza - You re...	13 feb
☐ ☆ PDF By Luis Rodrigo.	☐ "EXPERIENCIA EN EVALUACIÓN Y ACCESIBILIDAD A HERRAMIENTAS DE COMPUTACIÓN EN LA...	12 feb

Informar



Universidad de Guayaquil - Servicio de acceso federado
Una sola credencial para todos sus servicios



Error

Nombre de usuario o contraseña incorrectos

No existe un usuario con el identificador indicado, o la contraseña indicada es incorrecta. Por favor revise el identificador de usuario e inténtelo de nuevo.

Introduzca su nombre de usuario y contraseña

Un servicio solicita que se autentique. Por favor, introduzca su nombre de usuario y contraseña en el siguiente formulario.



Nombre de usuario

Contraseña

Organización

¡Ayuda! Se me ha olvidado la contraseña.

Sin su nombre de usuario y contraseña no se puede identificar y acceder al servicio. Quizás haya alguien que pueda ayudarle. ¡Contacte con el centro de atención al usuario de su organización!

Características de un buen diseño

- Los diseños de calidad superior deben tener características que dan como resultado productos de calidad: facilidad de comprender, de implementar, de probar, de modificar y la correcta traducción de la especificación de requerimientos.
- Atributos que reflejan la calidad del diseño:
 - Independencias de los componentes.
 - Identificación y tratamiento de excepciones.
 - Prevención de defectos y tolerancia de defectos.

Prevención de defectos y tolerancia de defectos

- La meta es hacer un código tan libre de defectos como sea posible incorporando al sistema la prevención y la tolerancia de defectos.
- Una de las características de un buen diseño es la forma en que previene y tolera los defectos.
- En lugar de esperar hasta que el sistema falle para corregir el problema, los diseñadores pueden anticipar lo que puede ocurrir y construir el sistema para que reaccione de manera aceptable.

Prevención de defectos y tolerancia de defectos

- **Detección activa de defectos:** Es la forma de controlar periódicamente la presencia de síntomas de defectos, se intenta anticipar la producción de fallas.
- Ejemplo:
 - **Sospecha Mutua:** cada componente del sistema supone que los otros componentes contienen defectos. Cada componente controla la precisión y consistencia de sus entradas. Este manejo inmediato del defecto limita el daño que produce, evita que se transforme en una falla.
 - **Redundancia:** según los resultados obtenidos por dos procesos se comparan si son idénticos. Ej.: En un programa de contabilidad primero se suman las filas, y luego se suman las columnas para ver si son iguales.

Prevención de defectos y tolerancia de defectos

- **Corrección de defectos:** Una vez descubierto el defecto se debe corregir.
- Por lo general, la corrección del defecto subsana el daño producido, así como modifica el producto para eliminar tal defecto.
- Los diseñadores deben incluir una estrategia para los defectos a medida que son encontrados.
- Se puede optar por detener el sistema cuando lo afecta el defecto de algún modo o simplemente se registra la existencia de la falla, apuntar el estado del sistema en el momento de producirse la falla y arreglar el daño después.
- La criticidad del sistema determinara la estrategia a usar.

Prevención de defectos y tolerancia de defectos

- **Tolerancia a defectos:** Muchas veces la corrección de un defecto es demasiado costosa, riesgosa e inconveniente.
- La tolerancia de fallas en el aislamiento del daño producido por una falla es conveniente y hasta deseable en muchas circunstancias.
- Para poder aplicar tolerancia debe existir una anticipación de los defectos, lo que es difícil sobre todo en sistemas complejos.



Universidad de Guayaquil



ARQUITECTURA

Arquitectura de Software

Especificación de Requerimientos del sistema (SRS)

En este camino hay mucho por hacer.

¿Comenzamos a programar para terminar lo antes posible?

- ¿Cuáles serían los riesgos?



Sistema instalado y funcionando

Arquitectura de Software

Especificación de Requerimientos del sistema (SRS)

No es un proceso en cascada. No se está definiendo un proceso.

- Arquitectura de Software
- Diseño detallado
- Implementación
- Verificación

Sistema instalado y funcionando

Arquitectura de Software

Los sistemas complejos están compuestos de subsistemas que interactúan bajo el control de un diseño de sistema

Arquitectura de Software

- Los subsistemas que componen el sistema,
- las interfaces y
- las reglas de interacción entre ellos.

Arquitectura de Software

- Proceso de diseño a través del cual:
 - Se **identifican** los subsistemas y
 - Se **establece** un **marco de trabajo** para el control y **comunicación** de éstos.

Arquitectura

Definición, estilos y evaluación:

- **Primer nivel** de descomposición, que muestra como se organiza el sistema en términos de sus componentes y las interacciones entre ellos.
- **Cambiar la Arquitectura** de un producto ya construido en general exige **mucho esfuerzo**
 - => **Evaluación de Arquitecturas**
- Distintos “**estilos**” que definen familias de sistemas en términos de patrones de organización estructural.
- Un **estilo de arquitectura** implica sus componentes, conectores y exigencias al combinarlos.
- Identificarlos y caracterizarlos para facilitar la **comunicación entre diseñadores**

Arquitectura

Influencia y características:

- Sus características **condicionan** las características del **producto final** (escalabilidad, capacidad, desempeño, consistencia, mantenibilidad, compatibilidad, etc.)
- Estilo y estructura particular elegidos pueden depender de **Requerimientos No Funcionales**:

- 1 - **Desempeño**: localizar operaciones críticas en un número reducido de subsistemas con poca comunicación. Componentes de **grano grueso**.
- 2 - **Seguridad**: estructurar en capas con los recursos más críticos protegidos por las capas más internas con alto nivel de validación.
- 3 - **Mantenibilidad**: componentes autocontenidos que puedan ser intercambiados con facilidad, evitando estructuras de datos compartidas. Componentes de **grano fino**.

CONFLICTO DE INTERESES: entre los requerimientos 1 y 3, si se tienen ambos se deberá buscar una solución intermedia.

Arquitectura

Elementos para la documentación:

- **SAD (Software Architecture Description)** salida del proceso de diseño de arquitectura, donde se incluyen modelos gráficos que muestran perspectivas distintas del sistema y descripciones textuales.
- **Documentarla** para que pueda ser utilizada y mantenida por otros, con suficiente detalle, sin ambigüedades ni repeticiones, registrando decisiones tomadas.
- **Notaciones:** UML general, accesible.
ADL's formales, para expertos.
- **Complejidad** se maneja documentando diferentes vistas de la arq., proyección en una dimensión mostrada desde una perspectiva, sin tener en cuenta entidades no relevantes a esa perspectiva.
- The "4+1" View Model of Software Architecture – Kruchten'95
Vistas definidas: lógica, procesos, implementación, física y CU. Todas son guiadas por los CU (o escenarios) significativos a la arquitectura

Ventajas de diseñar y documentar de manera explícita la arquitectura de software

- **Comunicación con los participantes:** La arquitectura es una presentación de alto nivel del sistema
 - Puede usarse como un enfoque para la discusión de un amplio número de participantes.
- **Análisis del sistema:** En una etapa temprana en el desarrollo del sistema, aclarar la arquitectura del sistema requiere cierto análisis.
 - Las decisiones de diseño arquitectónico tienen un efecto profundo sobre si el sistema puede o no cubrir requerimientos críticos como rendimiento, fiabilidad y mantenibilidad.
- **Reutilización a gran escala:** Un modelo de una arquitectura de sistema es una descripción corta y manejable de cómo se organiza un sistema y cómo interoperan sus componentes.
 - Por lo general, la arquitectura del sistema es la misma para sistemas con requerimientos similares y, por lo tanto, puede soportar reutilización de software a gran escala.
 - Así pues, es posible desarrollar arquitecturas de línea de productos donde la misma arquitectura se reutilice mediante una amplia gama de sistemas relacionados.

Ventajas de una arquitectura

- **Comunicación con Stakeholder**
 - Permite establecer diálogos entre los stakeholders en torno a lo que será el sistema.
- **Análisis del sistema**
 - Para definir cuáles son y cómo alcanzar los requerimientos no funcionales.
- **Reuso a gran escala**
 - Una buena arquitectura puede reutilizarse para otros sistemas.

Arquitectura

Beneficios esperados de prestarle atención:

- Mejorar la **comunicación** entre los **distintos interesados**:
 - Cliente – diseñadores
 - Diseñadores – desarrolladores
- Clarificar **intenciones de diseño**
 - la arquitectura concebida a menudo se pierde, comunicación en gral. informal (difícil)
- Proporcionar bases para **análisis del diseño**
 - predecir desempeño y otras características y ajustar el diseño como tarea rutinaria
- Mejorar el **mantenimiento**
 - gran parte del esfuerzo de mantenimiento se dedica a entender
- Identificar **cuestiones interesantes**
 - incluso careciendo de métodos formales

Arquitectura

Métodos para evaluación de arquitecturas:

- Analizar la arquitectura para ver si cumple requisitos de calidad establecidos (ej. confiabilidad, interoperabilidad)
- Preferible realizar evaluaciones tempranas que permitan introducir cambios con menor impacto y mejorar los aspectos de riesgo identificados
- Evaluaciones a posteriori resultan útiles como forma de aprendizaje y estudio de posibilidades de mejora, por ej. para una nueva versión del producto
- Software Engineering Institute (SEI) propone:
 - Architecture Tradeoff Analysis Method (ATAM)
 - Software Architecture Analysis Method (SAAM)

Diseño: Arquitectura vs. Programas

	Arquitectura	Programas
Muestra	Interacciones entre partes	Implementaciones de partes
Considera	Propiedades estructurales	Propiedades computacionales
Análisis	En general estático	En general dinámico
Evaluación	Desempeño a nivel del Sistema	Desempeño de algoritmos
Visión	Fuera de los límites de módulo	Dentro de los límites de módulo
Reutilización	Composición de subsistemas	Copia de código o llamado a bibliotecas

Arquitectura de referencia

- El conjunto de decisiones de diseño principales que son aplicables simultáneamente a múltiples sistemas relacionados, normalmente dentro del dominio de la aplicación con puntos de variación explícitamente definidos, se denomina *arquitectura de referencia*.
 - Por ello, en este curso *revisaremos diferentes arquitecturas de referencia* para, en base a ellas, elaborar nuestros diseños arquitectónicos de sistema ;)

Aspecto temporal

- Las decisiones de diseño arquitectónico se toman y se deshacen durante la vida útil de un sistema
 - La arquitectura tiene un **aspecto temporal**
- En cualquier momento dado, el sistema tiene solo una arquitectura
- La arquitectura de un sistema **cambiará con el tiempo**

Arquitectura Prescriptiva vs. Descriptiva

- Arquitectura Prescriptiva:
 - Durante el proceso de ingeniería de software, los arquitectos de sistemas habrán tomado una serie de decisiones de diseño que reflejan su intención. Estas decisiones de diseño comprenden la arquitectura prescriptiva del sistema.
- La arquitectura prescriptiva de un sistema captura las *decisiones de diseño tomadas antes de la construcción del sistema*.
 - Es la arquitectura que presenta *cómo se concibió* o *cómo se pretendía* el sistema

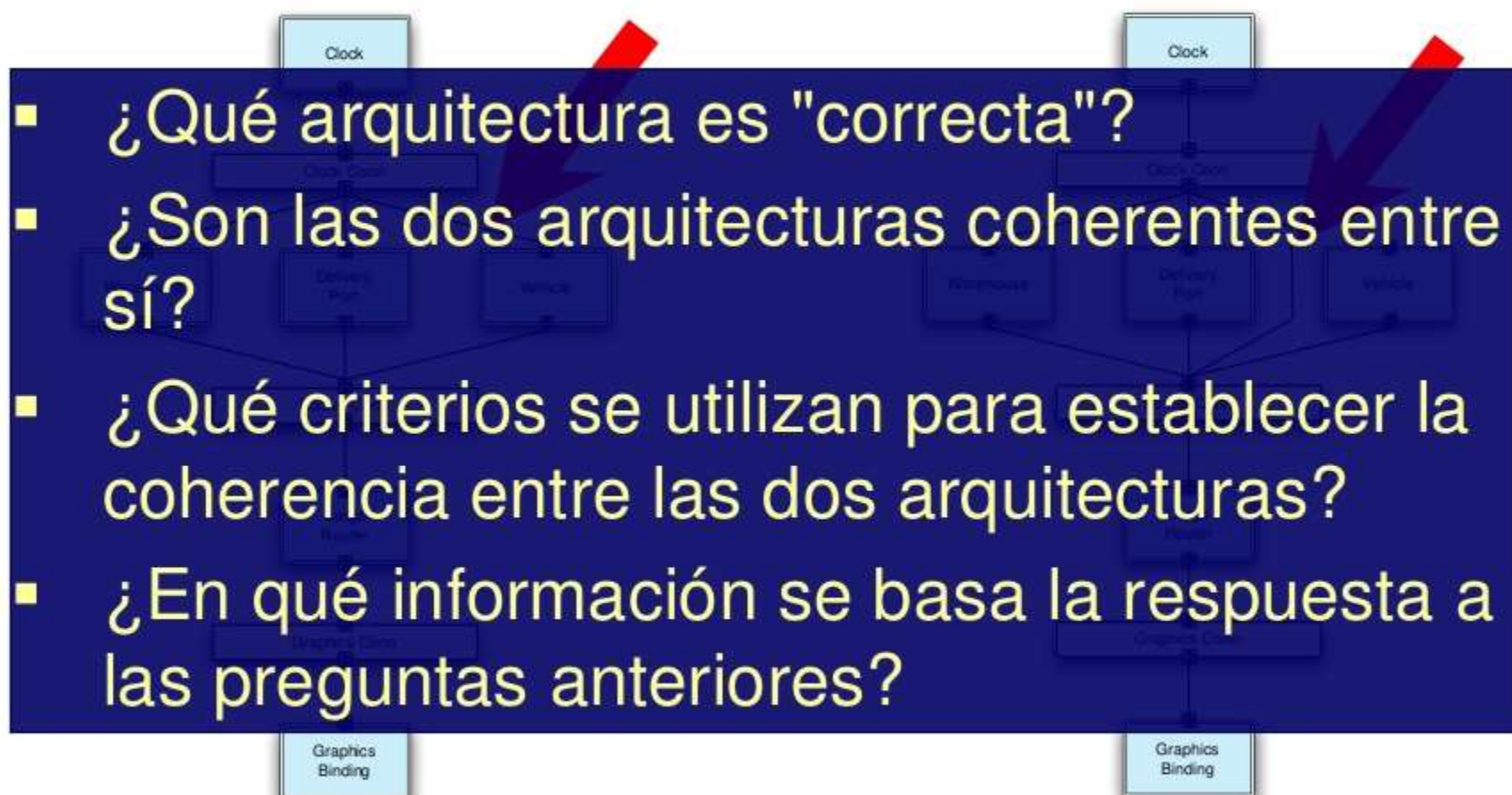
Visión global del desarrollo (Arquitectura Prescriptiva)



Arquitectura Prescriptiva vs. Descriptiva

- Arquitectura Descriptiva:
 - El conjunto completo de decisiones de diseño principales D , incorporadas en el conjunto de artefactos A , se denomina arquitectura descriptiva del sistema.
- La arquitectura descriptiva de un sistema describe *cómo se ha construido* el sistema
 - Es la arquitectura *implementada* o *realizada*

Dos arquitecturas: una al lado de la otra



*Arquitectura
Prescriptiva*

*Arquitectura
Descriptiva*

Evolución de la arquitectura

- Cuando un sistema evoluciona, idealmente su arquitectura **prescriptiva** se modifica primero
- En la práctica, el código del sistema –y así su arquitectura **descriptiva**– a menudo se modifica directamente
- Esto sucede debido a
 - Descuido del desarrollador
 - Percepción de plazos cortos que impiden pensar y documentar
 - Falta de arquitectura prescriptiva bien documentada
 - Necesidad o deseo de optimizar código
 - Técnicas inadecuadas o soporte de herramientas

Evolución de la arquitectura

- **P**- decisiones principales de diseño
- **A**- artefactos
- **D**- decisiones de diseño correspondientes
- A medida que estos tres crecen, la arquitectura se vuelve más completa
- En cualquier momento, cualquier par de **PA** y **DA** representa la arquitectura de software del sistema.

Degradación arquitectónica

- En cualquier momento, la diferencia en la arquitectura **prescriptiva** y la arquitectura **descriptiva** del sistema es una *degradación arquitectónica*
- Causas:
 - Descuido de los desarrolladores, límite de tiempo, plazos compactos, etc...
- Dos conceptos relacionados
 - Architectural drift
 - Architectural erosion

Tipos de degradación arquitectónica

- **Architectural drift** es la introducción de las principales decisiones de diseño en la arquitectura descriptiva de un sistema que
 - no están incluidos, abarcados o implícitos en la arquitectura prescriptiva
 - pero *que no violen ninguna de las decisiones de diseño de la arquitectura prescriptiva*
 - Ejemplo: adición de decisiones que no están prohibidas

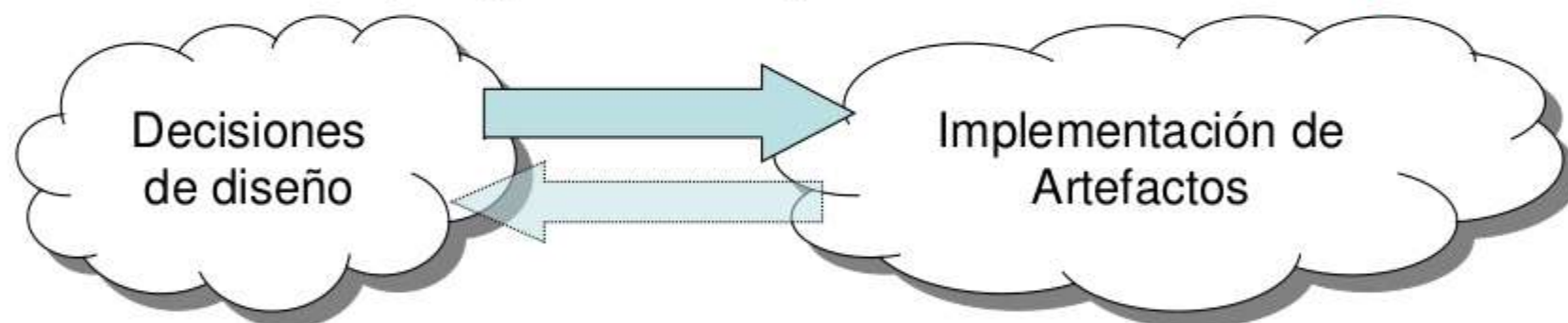


- **Architectural erosion** es la introducción de decisiones de diseño arquitectónico en la arquitectura descriptiva de un sistema que *violan su arquitectura prescriptiva*
 - **Demasiada deriva (drift) causa erosión**



El problema del mapeo

- La **implementación** es la única fase de la ingeniería de software que **no es opcional**
- El desarrollo basado en la arquitectura proporciona un giro único al problema clásico
 - Se convierte, en gran medida, en una actividad *de mapeo*



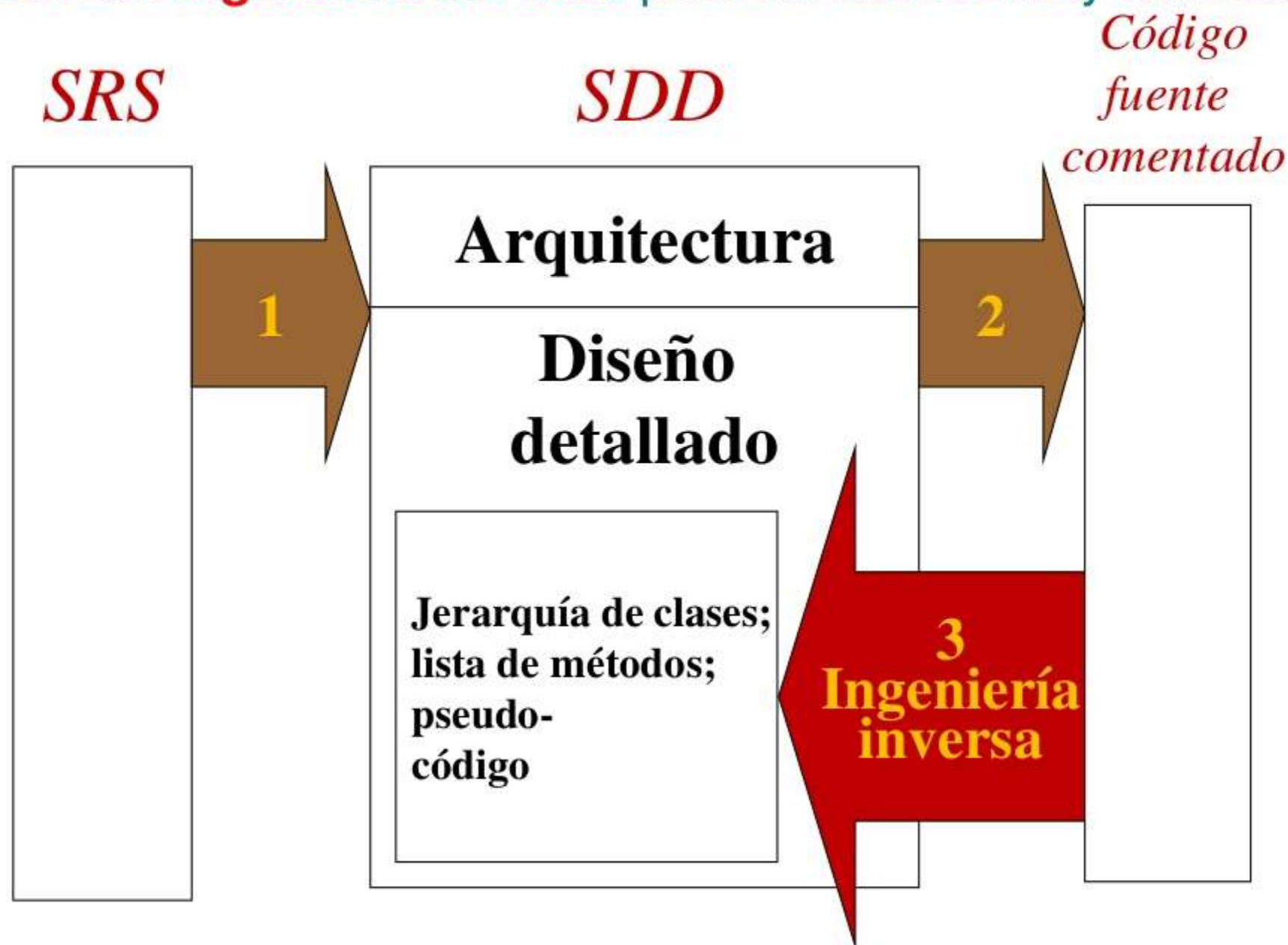
- Mantener el **mapeo** significa asegurar que **nuestra intención arquitectónica se refleje en nuestros sistemas construidos**

Recuperación arquitectónica

(Architectural Recovery)

- *Architectural recovery* es el proceso de determinar la arquitectura de un sistema de software a partir de sus artefactos a nivel de implementación
- Los artefactos a nivel de implementación pueden tratarse de
 - Código fuente
 - Archivos ejecutables
 - Archivos Java .class

Uso de **ingeniería inversa** para un diseño muy detallado



Perspectiva o vista arquitectónica

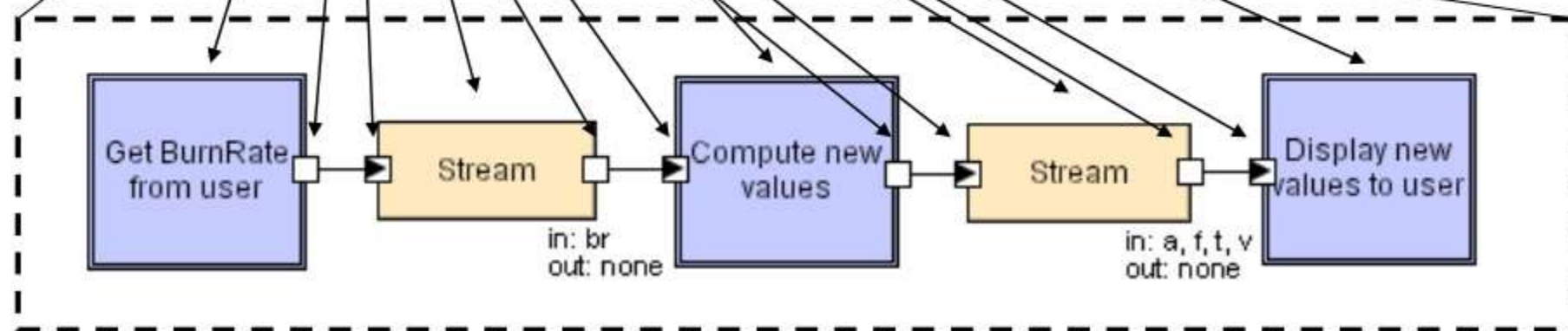
- Definición:
 - Es un conjunto no vacío de tipos de decisiones de diseño arquitectónico.
- La perspectiva debe fijarse para obtener una vista adecuada
- Ejemplos:
 - Perspectiva de implementación
 - Perspectiva estructural
 - Perspectiva de comportamiento o funcional

Elementos de la arquitectura de software

- La arquitectura de un sistema de software debe ser una **composición** y una rica **interacción** de diferentes elementos.
- Los elementos abordan preocupaciones clave del sistema como:
 - Procesamiento / funcionalidad / comportamiento
 - Datos, también denominados información o estado
 - Interacción

¿Qué modelaremos?

- Elementos arquitectónicos básicos
 - Componentes
 - Conectores
 - Interfaces
 - Configuraciones
 - Justificación – razonamiento detrás de las decisiones



Componentes



- Los elementos que encapsulan el procesamiento y los datos en la arquitectura de un sistema se denominan *componentes de software (específicos de la aplicación)*.
- **Definición**
 - El *componente de software* es una entidad arquitectónica que
 - encapsula un subconjunto de la funcionalidad y / o datos del sistema
 - restringe el acceso a ese subconjunto a través de una interfaz definida explícitamente
 - tiene dependencias definidas explícitamente en su contexto de ejecución requerido
- Los componentes suelen proporcionar servicios específicos de la aplicación

Componentes



- Los componentes pueden ser complejos o simples dependiendo de:
 - perspectiva,
 - arquitectura tomada por desarrolladores
 - necesidad del sistema
- Visible solo a través de la interfaz, sin interfaz es una caja negra (entidad abstracta)
- Abstracción, modularidad y encapsulación
- Reutilizable y evolutivo
- La reutilización a veces da como resultado una deriva (razón principal de la deriva)

Conectores

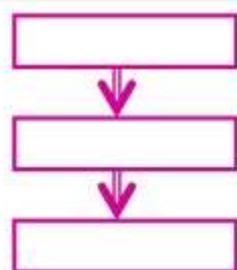


- En sistemas complejos, la *interacción* puede llegar a ser más importante y desafiante que la funcionalidad de los componentes individuales
- **Definición:**
 - Un *conector de software* es un bloque de construcción arquitectónico encargado de efectuar y regular las interacciones entre los componentes
- En muchos sistemas de software, los conectores suelen ser simples llamadas a procedimientos o accesos a datos compartidos.
- Los conectores suelen proporcionar facilidades de interacción independientes de la aplicación.

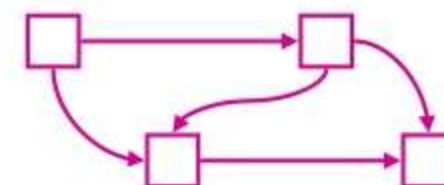
Ejemplos de Conectores



- Conectores de llamada a procedimiento
- Conectores de memoria compartida
- Conectores de paso de mensajes
- Conectores de transmission
- Conectores de distribución
- Conectores wrapper / adaptador



Configuraciones

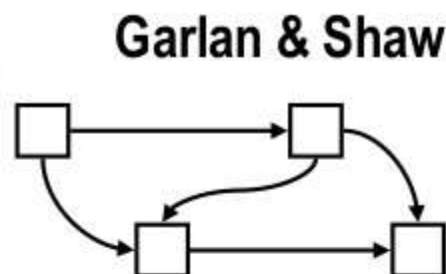


- Los componentes y conectores se organizan de una manera específica en la arquitectura de un sistema dado para lograr el objetivo de ese sistema
- **Definición**
 - Una *configuración arquitectónica*, o topología, es un conjunto de asociaciones específicas entre los componentes y conectores de la arquitectura de un sistema de software.

Clasificación de configuraciones (pat.arq.)

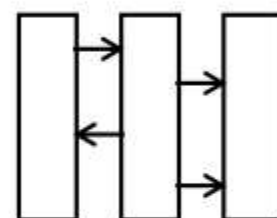
❑ *Flujo de datos*

- Datos fluyen entre los elementos funcionales



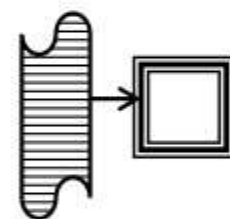
❑ *Componentes independientes*

- -- ejecución en paralelo, ocasionalmente se están comunicando



❑ *Máquinas virtuales*

- Interpretador + programa en lenguaje de propósito especial



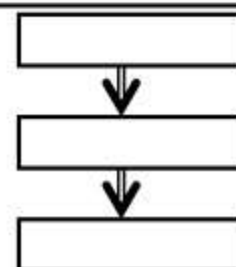
❑ *Repositorios*

- Construido principalmente en torno a un gran almacén de datos



❑ *Capas jerárquicas (layers)*

- Subsistemas, cada uno de los cuales depende unidireccionalmente de otro subsistema



Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Recordando... Modularización - Beneficios

- Separar aspectos del diseño
 - Permitir cambiar algunos sin afectar al resto
- Permite
 - Extender fácilmente artefactos (extensibilidad)
 - Reusar artefactos existentes (reusabilidad)
 - Ocultar dependencias de la plataforma (portabilidad)
 - Diseñar interfaces que se adapten a otras (compatibilidad)
 - Minimizar impacto de cambios (mantenibilidad)
 - Facilitar la comprensión
 - Organizar y distribuir el trabajo
- **Alta modularización** => Alta cohesión – Bajo acoplamiento

Evidencias de diseño erróneo

(Problemas fundamentales de diseño)

- **Rigidez:** problemas para insertar algún cambio.
- **Fragilidad:** el software falla en muchos lugares al insertar un cambio.
- **Inmovilidad:** no se pueden rehusar partes del proyecto.
- **Viscosidad:**
 - **De diseño:** cuando se deben hacer cambios, es más fácil hacer cosas mal, que bien.
 - **De entorno:** entorno de desarrollo ineficiente

Cambios de requerimientos

(Problemas fundamentales de diseño)

- Los cambios en un diseño de software, si no fueron cambios previstos en el diseño original, **degradan** el mismo. Incluyen dependencias.
- Generalmente lo hacen ingenieros que no estaban relacionados con la filosofía de diseño original.

Recordando... Tipos de degradación arquitectónica

- **Architectural drift** es la introducción de las principales decisiones de diseño en la arquitectura descriptiva de un sistema que
 - no están incluidos, abarcados o implícitos en la arquitectura prescriptiva
 - pero *que no violen ninguna de las decisiones de diseño de la arquitectura prescriptiva*
 - Ejemplo: adición de decisiones que no están prohibidas



- **Architectural erosion** es la introducción de decisiones de diseño arquitectónico en la arquitectura descriptiva de un sistema que *violan su arquitectura prescriptiva*
 - **Demasiada deriva (drift) causa erosión**



Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

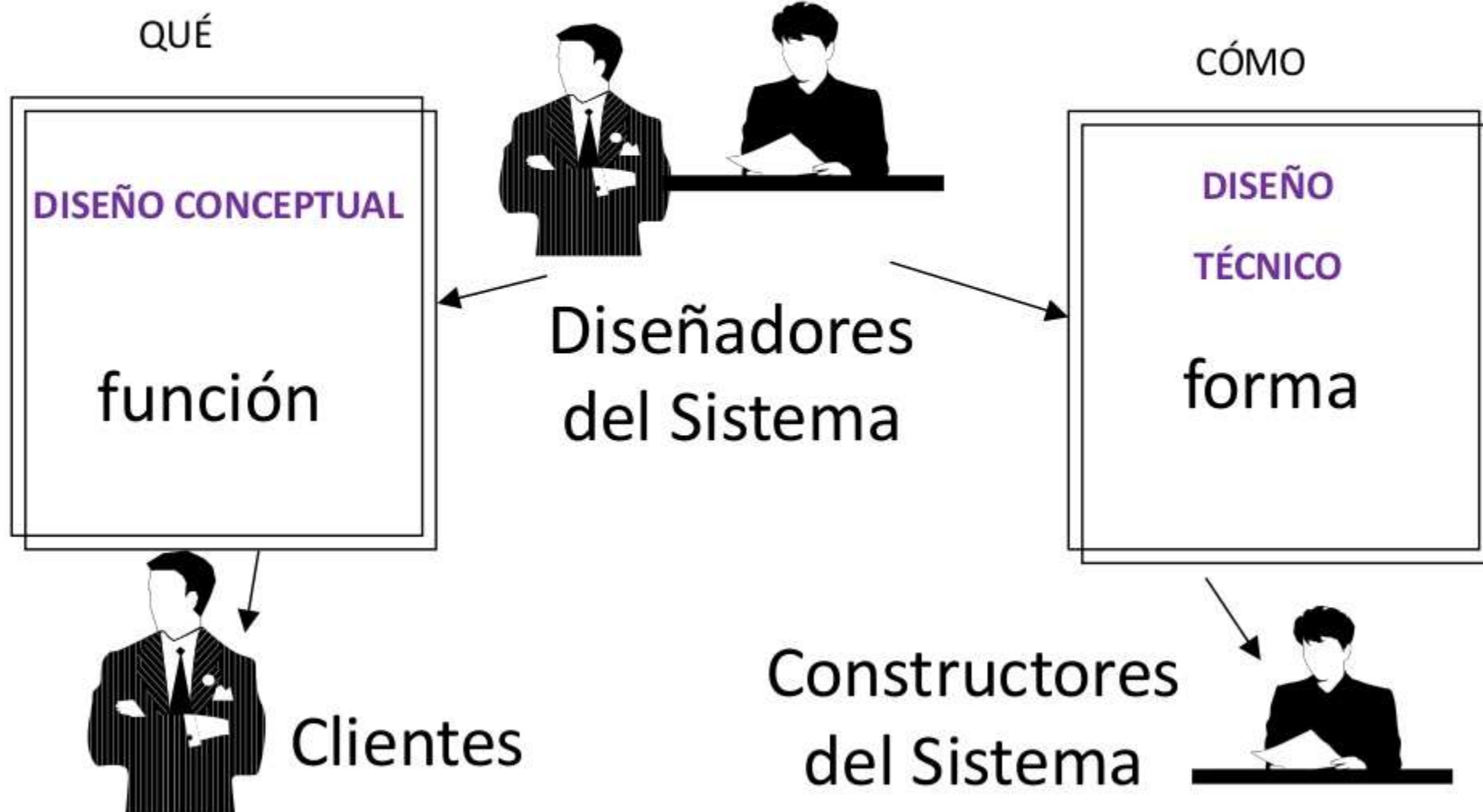
Requisitos y diseño

- En principio, los requisitos deben indicar lo **qué** el sistema debe hacer y el diseño debe describir **cómo** lo debe hacer.
- En la práctica, **es prácticamente imposible excluir toda la información de diseño** al especificar en un nivel adecuado los requisitos de software.
 - La arquitectura del sistema puede ser diseñada para estructurar los requisitos.
 - El sistema puede interactuar con otros sistemas que generan requisitos de diseño.
 - El uso de una arquitectura específica para satisfacer los requisitos no funcionales puede ser un requisito.

Diseño Conceptual y Técnico

- Para transformar los requerimientos en un sistema que funcione, los diseñadores deben satisfacer tanto a los clientes como a los constructores de sistemas.
- Los clientes deben comprender lo que el sistema debe hacer, y los constructores deben comprender cómo debe operar el sistema.
- El diseño es un proceso iterativo que consta de dos partes: **Diseño conceptual o del sistema, Diseño técnico.**

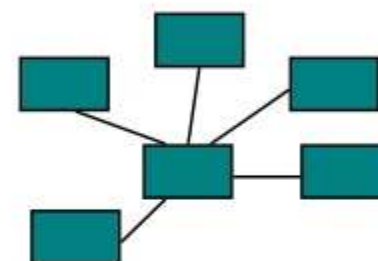
Diseño y Especificación de Requerimientos(1)



Diseño y Especificación de Requerimientos(2)

“El usuario podrá enviar mensajes a cualquier usuario en cualquier otra computadora en red”

DISEÑO
CONCEPTUAL



Topología de Red
Protocolo
Velocidad (bps)
...

DISEÑO
TÉCNICO

Diseño Conceptual o del Sistema (1)

- Se realiza primero, y le dice al cliente que es lo que hará realmente el sistema.
- Una vez que se aprueba este diseño, se vuelca el trabajo a un trabajo de diseño más detallado.
- El diseño conceptual describe el sistema en un lenguaje que el cliente pueda comprender, en lugar de usar términos técnicos o jergas de computación.

Diseño Conceptual o del Sistema (2)

- Un buen diseño conceptual debería tener las siguientes características:
 - Se escribe en el lenguaje del cliente.
 - No contiene expresiones de jerga técnica.
 - Describe claramente las funciones del sistema.
 - Es independiente de la implementación.
 - Esta vinculado con los documentos del requerimiento.

Diseño Técnico (1)

- Permite que los constructores comprendan el hardware y el software concretos que se necesitan para resolver el problema del cliente.
- Es decir este diseño describe:
 - La configuración de hardware.
 - Las necesidades de software.
 - Las interfaces de comunicación.
 - Las entradas y salidas del sistema.
 - La arquitectura de red.
 - Y cualquier otro aspecto que incida en la transformación de los requerimientos en una solución.

Diseño Técnico (2)

- El diseño técnico incluye los siguientes ítem:
 - Una descripción de los principales componentes de hardware y de sus funciones.
 - La jerarquía y funciones de los componentes de software.
 - Las estructuras de datos y los flujos de datos.

Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Atributos de calidad del diseño

- Rendimiento (performance)
- Seguridad de acceso y de la información (Security)
- Seguridad en el uso de la aplicación (Safety)
- Disponibilidad (Availability)
- Mantenibilidad (Maintainability)

Calidad del Diseño

Se dice que es un diseño con calidad, si es:

- Mantenable: facilidad de modificar o añadir funcionalidades
- Entendible
- Modular: los cambios tienen un efecto local
- Cohesivo: todas las partes de una componente del sistema tienen relación lógica
- Acoplado: relación/unión entre los componentes. La cual debe ser baja

“Los atributos de calidad no están atados a las estrategias de diseño”.

La arquitectura del sistema afecta al...

(requerimientos críticos)

- **Rendimiento:** arquitectura debe identificar las operaciones críticas en un pequeño número de subsistemas con tan poca comunicación como sea posible entre esos subsistemas.
 - Puede significar el uso relativo de componentes de **grano grueso (grandes)** en vez de componentes de **grano fino (muy pequeños)** para *reducir las comunicaciones entre ellos*
- **Protección:** usar una arquitectura basada en capas, con los recursos *más protegidos en las capas más internas* y aplicando una validación de seguridad de alto nivel en dichas capas.
- **Seguridad:** arquitectura debe diseñarse para que las operaciones relacionadas con la *seguridad se localicen en un único subsistema o en un pequeño número de subsistemas*.
- **Disponibilidad:** arquitectura debe incluir *componentes redundantes y debe ser posible reemplazar* y actualizar componentes sin detener el sistema (arquitecturas de sistemas tolerantes a defectos).
- **Mantenibilidad:** arquitectura debe incluir *componentes independientes* de grano fino que puedan *modificarse con facilidad*. Los productores de datos deben separarse de consumidores y deben evitarse las estructuras de datos compartidas.

Conflictos entre requerimientos críticos

- Cuando se usan componentes de grano grueso se mejora el **rendimiento** pero se reduce la **mantenibilidad**.
- Si se introducen datos/hardware redundantes se mejora la **disponibilidad** pero se dificultan los aspectos de **seguridad**.
- Las características de **seguridad** en el uso de la aplicación usualmente implica mayor **comunicación** y se reduce el **rendimiento**.
- Etc, etc, etc...



Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Diseño orientado a funciones

(Function-oriented Design)

- Diseño con unidades funcionales que transforman entradas en salidas.
- Algunos sistemas están *naturalmente* orientados a funciones
- Sistemas que mantienen información mínima sobre el estado.
 - Es decir, cuando el sistema se ocupa de procesar acciones independientes cuyos resultados no se ven afectados por acciones anteriores
- Intercambio de información se da a través de listas de parámetros

ALGORITMOS Y LÓGICA DE PROGRAMACIÓN

Se considera dado y conocido

Diseño orientado a funciones

(Function-oriented Design)



- Los sistemas de procesamiento de transacciones (ATMs) entran en esta categoría. **¿Por qué?**
 - En ellos, cada transacción es independiente.

```
loop
  loop
    Print_input_message (" Welcome - Please enter your card");
    exit when Card_input ;
  end loop ;
  Account_number := Read_card ;
  Get_account_details (PIN, Account_balance, Cash_available) ;
  if Validate_card (PIN) then
    loop
      Print_operation_select_message ;
      case Get_button is
        when Cash_only =>
          Dispense_cash (Cash_available, Amount_dispensed) ;
        when Print_balance =>
          Print_customer_balance (Account_balance) ;
        when Statement =>
          Order_statement (Account_number) ;
        when Check_book =>
          Order_checkbook (Account_number) ;
      end case ;
      Eject_card ;
      Print ("Please take your card or press CONTINUE") ;
      exit when Card_removed ;
    end loop ;
    Update_account_information (Account_number, Amount_dispensed) ;
  else
    Retain_card ;
  end if ;
end loop ;
```


Diseño funcional y orientado a objetos

- Para muchos tipos de aplicaciones, es probable que el diseño orientado a objetos conduzca a un sistema más confiable y fácil de mantener.
- Para aplicaciones que mantienen un estado pequeño, el diseño orientado a funciones es apropiado.
- Los estándares, métodos y herramientas CASE para el diseño funcional están bien establecidos.
- Se deben mantener los sistemas existentes. El diseño orientado a funciones seguirá siendo utilizado hasta, al menos, la mitad del siglo XXI.

Proceso de diseño funcional

- **Diseño de flujo de datos**
 - Modela el procesamiento de datos en el sistema usando diagramas de flujo de datos
- **Descomposición estructural**
 - Modela cómo las funciones se descomponen en subfunciones utilizando tablas de estructura gráfica
- **Diseño detallado**
 - Las entidades en el diseño y sus interfaces se describen en detalle. Estos pueden registrarse en un diccionario de datos y el diseño puede expresarse mediante una PDL (Program Design Language)

PDL: Está relacionado con el pseudocódigo, pero a diferencia del pseudocódigo, está escrito en lenguaje sencillo sin ningún término que pueda sugerir el uso de cualquier lenguaje de programación o biblioteca.

Proceso de diseño funcional

- Diseño de flujo de datos
- Descomposición estructural
- Diseño detallado

Diagramas de flujo de datos

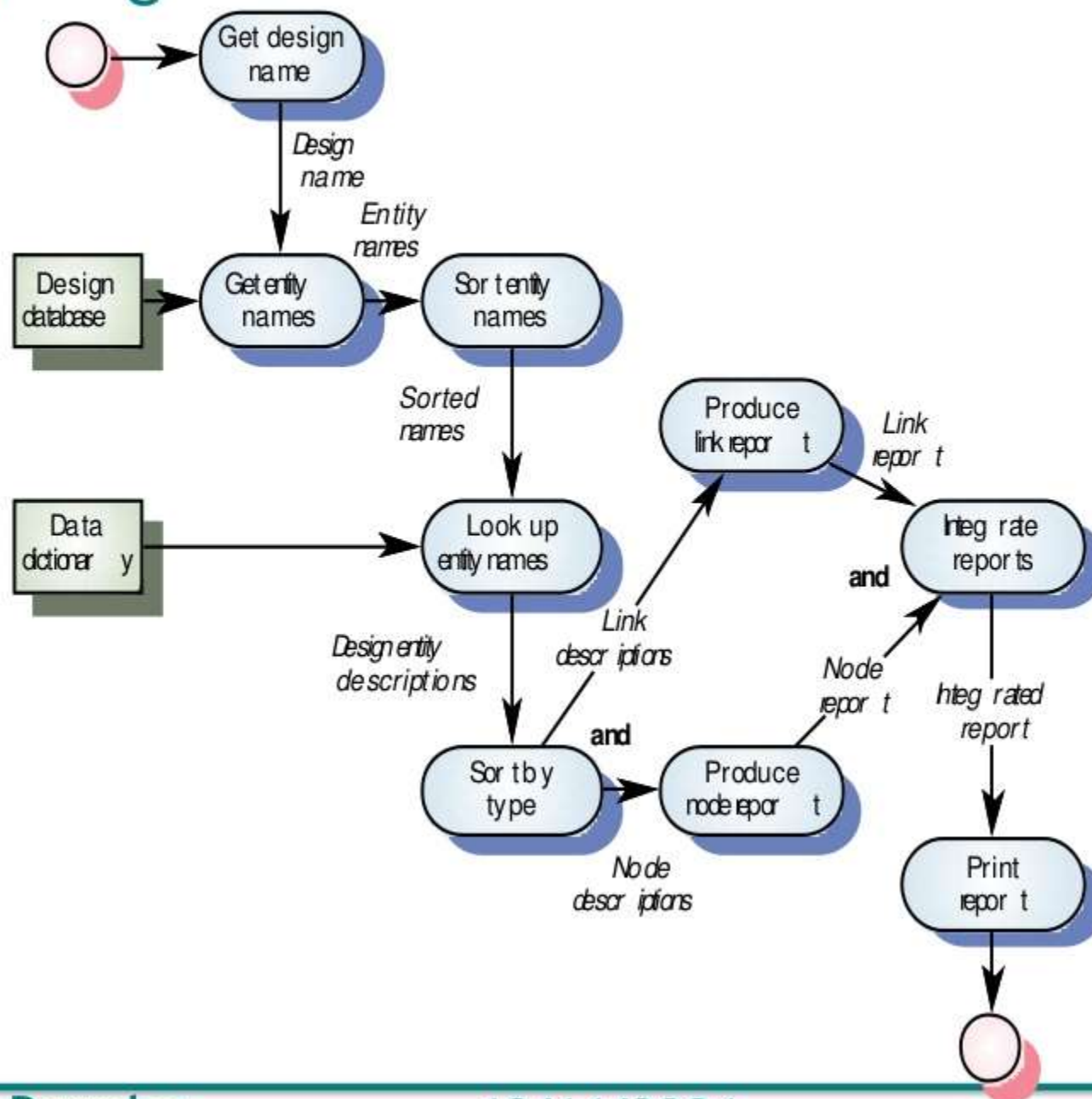
(DFD- Data Flow Diagrams)

- Muestra cómo un elemento de datos de entrada es transformado funcionalmente por un sistema en un elemento de datos de salida
- Son una parte integral de muchos métodos de diseño y son compatibles con muchos sistemas CASE

Notación DFD

- Rectángulo redondeado: función o transformación
- Rectángulo - almacén de datos
- Círculos: interacciones del usuario con el sistema
- Flechas: muestran la dirección del flujo de datos
- Palabras clave y / o: se utiliza para vincular flujos de datos

Ejemplo: generador de informes de diseño



Proceso de diseño funcional

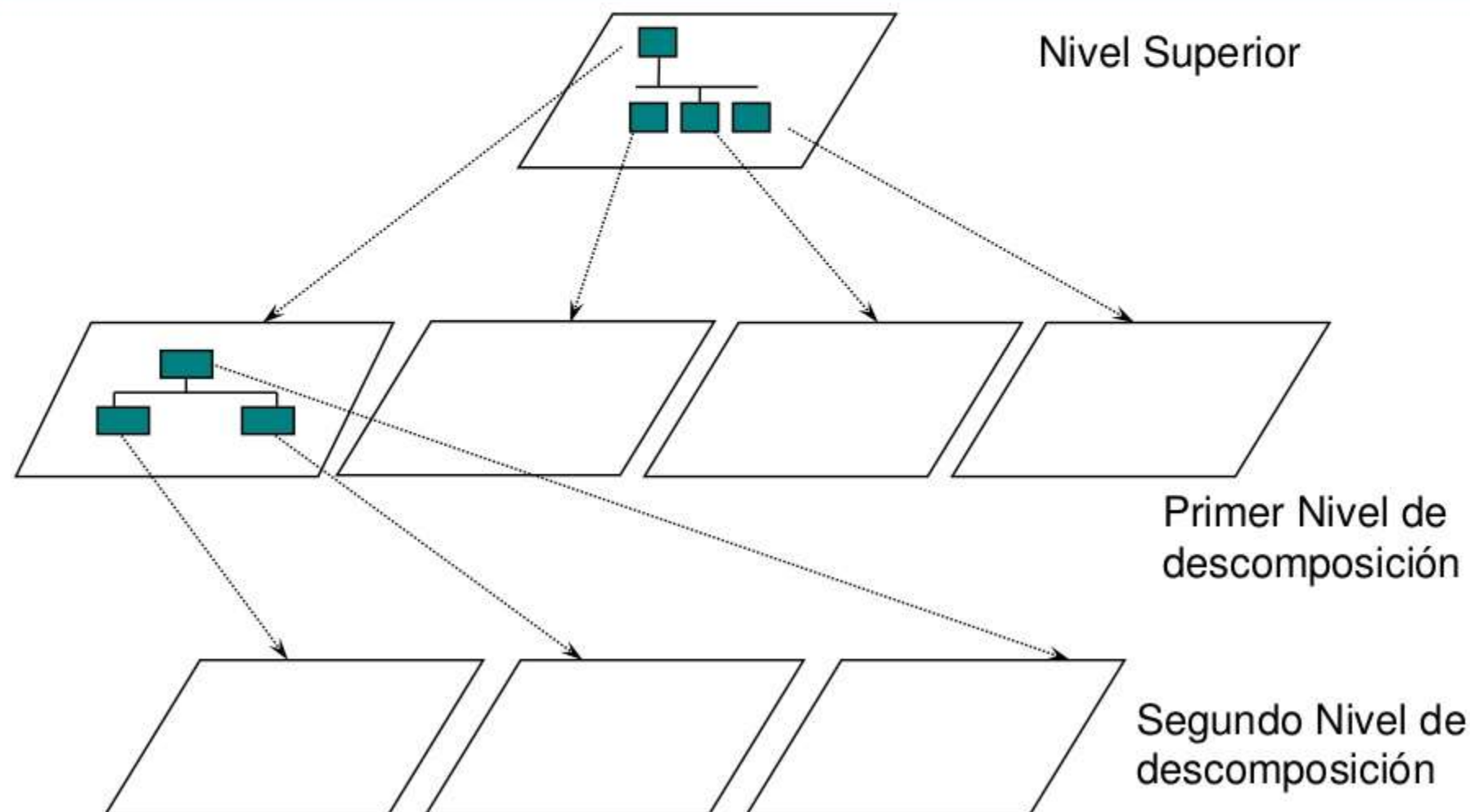
- Diseño de flujo de datos
- Descomposición estructural
- Diseño detallado

Descomposición estructural

- La descomposición estructural se ocupa de desarrollar un modelo del diseño que muestre la ***estructura dinámica***; es decir, llamadas a funciones.
- El objetivo del diseñador debe ser derivar unidades de diseño que sean altamente cohesivas y poco acopladas.
- En esencia, *un diagrama de flujo de datos se convierte en un diagrama de estructura*.

Descomposición estructural

Todo método de diseño abarca un tipo de descomposición a partir de una descripción de alto nivel de los elementos claves del sistema, y creando visiones a niveles inferiores para ver como las características y funciones del sistema se acomodaran en conjunto.



Directrices de descomposición

- Para aplicaciones comerciales, el **nivel superior** puede tener cuatro funciones, a saber, entrada, proceso, actualización de archivo maestro y salida.
- Las funciones de validación de datos deben estar subordinadas a una función de entrada.
- La coordinación y el control deben ser responsabilidad de funciones cercanas a la cima de la jerarquía.

Directrices de descomposición

- El objetivo del proceso de diseño es identificar funciones poco acopladas y altamente cohesivas.
 - Por tanto, cada función debería hacer una cosa y sólo una cosa.
- Cada nodo en el diagrama de estructura debe tener entre dos y siete subordinados.

Los pasos del proceso

- Identificar transformaciones de procesamiento del sistema
 - Transformaciones en el DFD que se refieren al procesamiento más que a las actividades de entrada / salida.
 - Agruparlas bajo una sola función en el diagrama de estructura
- Identificar transformaciones de entrada
 - Transformaciones relacionadas con la lectura, validación y formateo de entradas.
 - Agruparlas bajo la función de entrada
- Identificar transformaciones de salida
 - Transformaciones relacionadas con el formato y la escritura de salida.
 - Agruparlas bajo la función de salida

Gráfico de estructura inicial

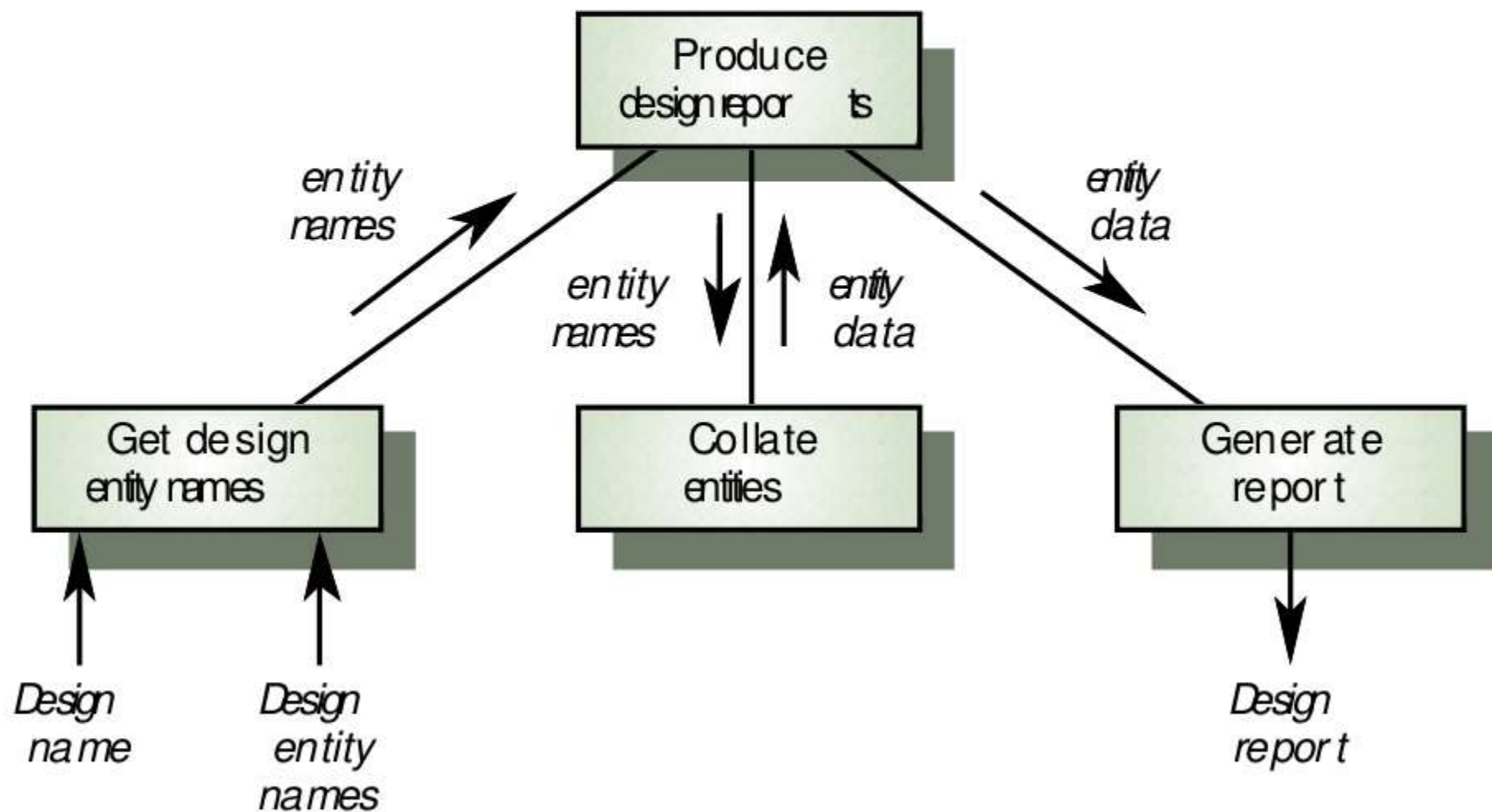


Gráfico de estructura expandido

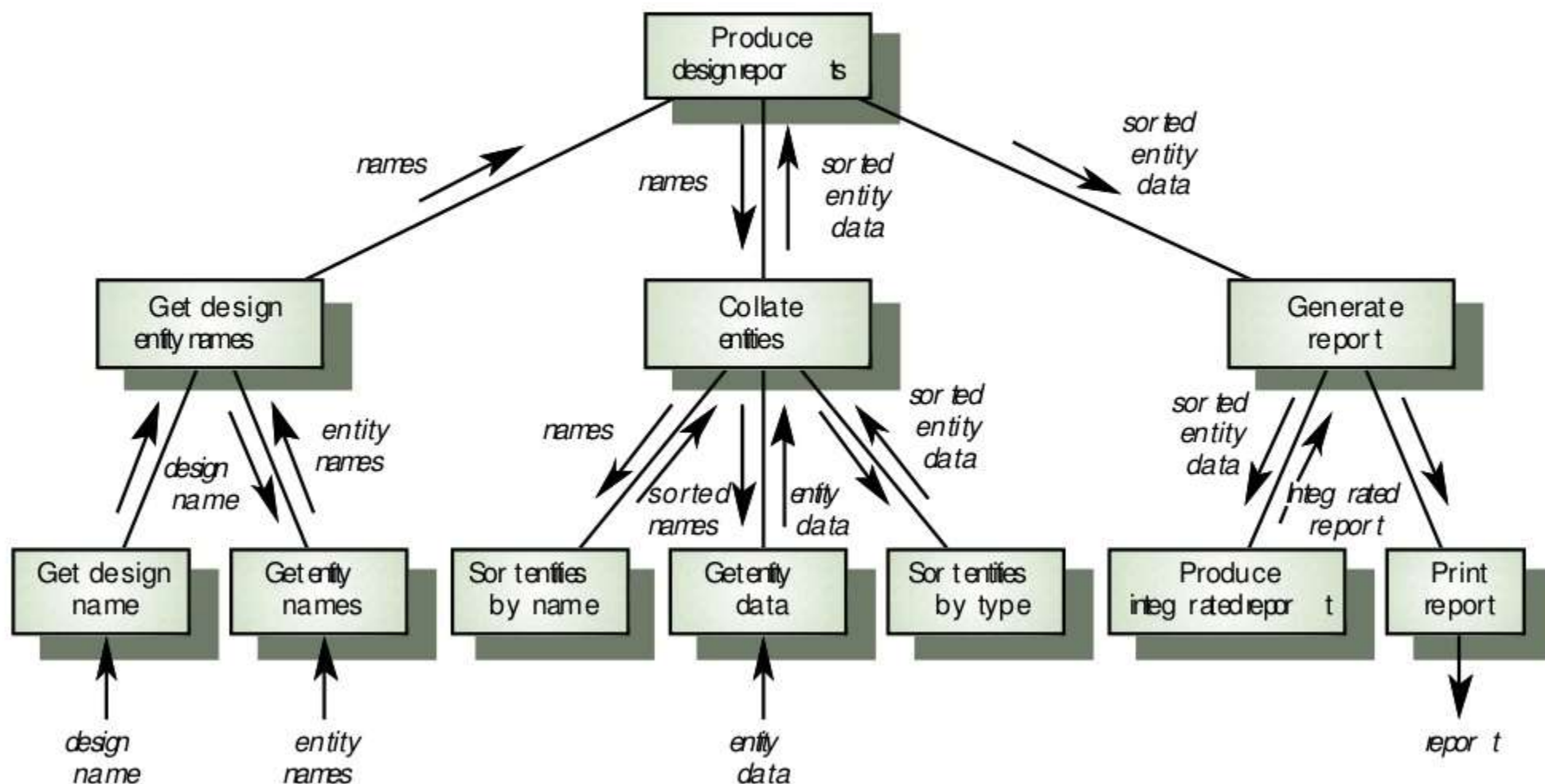
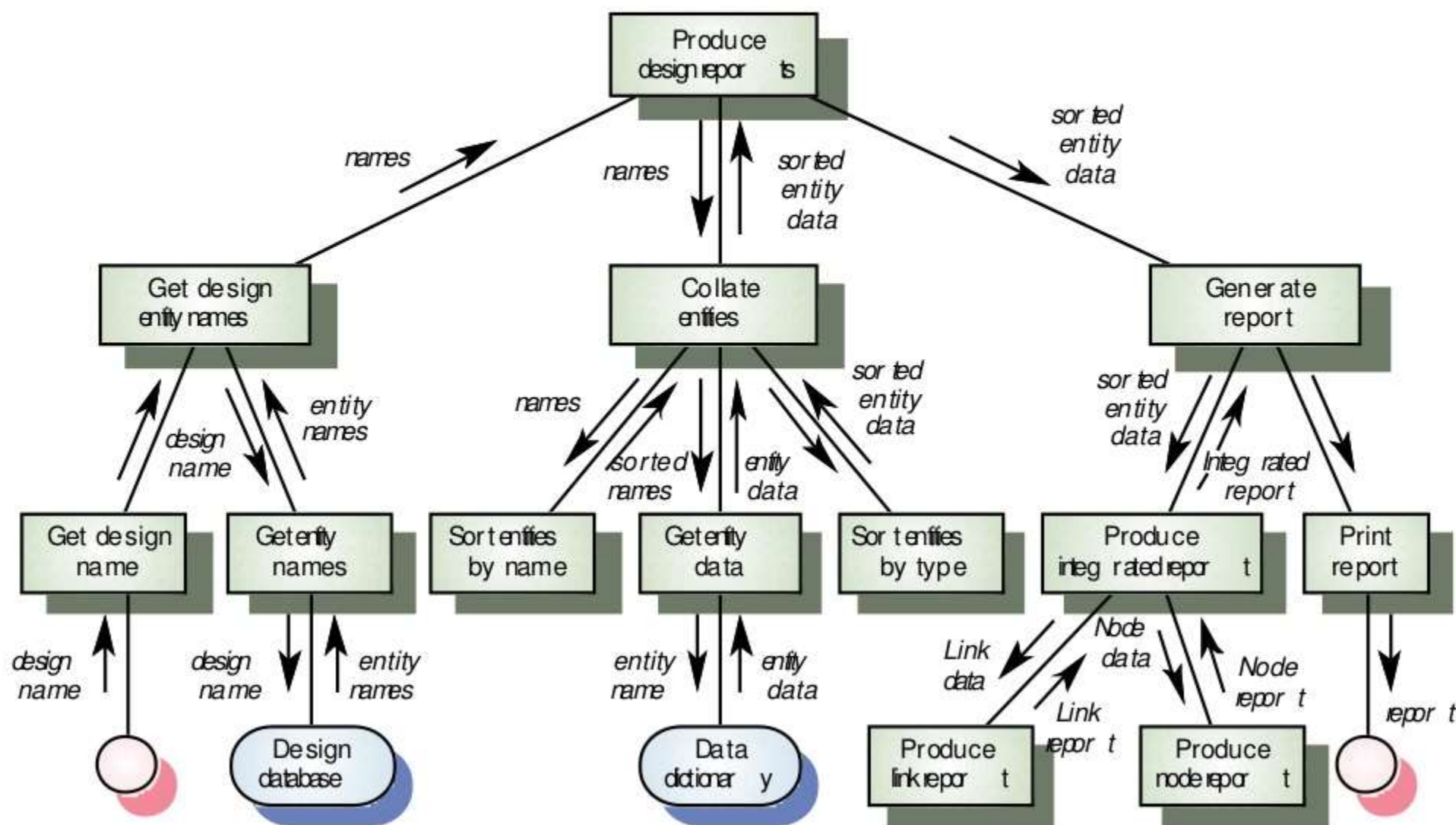


Gráfico de estructura final



Proceso de diseño funcional

- Diseño de flujo de datos
- Descomposición estructural
- Diseño detallado

Diseño detallado

- Preocupado por producir una breve especificación de diseño (mini-especificación) de cada función. Esto debe describir el procesamiento, las entradas y las salidas.
- Estas descripciones deben gestionarse en un diccionario de datos.
- A partir de estas descripciones, se pueden producir descripciones de diseño detalladas, expresadas en una PDL.

Entradas del diccionario de datos

Entity name	Type	Description
Design name	STRING	The name of the design assigned by the design engineer.
Get design name	FUNCTION	<i>Input:</i> Design name <i>Function:</i> This function communicates with the user to get the name of a design that has been entered in the design database. <i>Output:</i> Design name
Get entity names	FUNCTION	<i>Input:</i> Design name <i>Function:</i> Given a design name, this function accesses the design database to find the names of the entities (nodes and links) in that design. <i>Output:</i> Entity names
Sorted names	ARRAY of STRING	A list of the names of the entities in a design held in ascending alphabetical order.

Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

¿Por qué la Orientación a Objetos?

- Por su proximidad de los conceptos de modelado respecto de las entidades del mundo real
 - Mejora la captura y validación de requisitos
 - Acerca el “espacio del problema” y el “espacio de la solución”
- Modelado integrado de propiedades estáticas y dinámicas del ámbito del problema
 - Facilita construcción, mantenimiento y reutilización
- Podríamos dar muchas razones pero hay problemas.

Problemas en OO

"...Los conceptos básicos de la OO se conocen desde hace dos décadas, pero su aceptación todavía no está tan extendida como los beneficios que esta tecnología puede sugerir"

"...La mayoría de los usuarios de la OO no utilizan los conceptos de la OO de forma purista, como inicialmente se pretendía. Esta práctica ha sido promovida por muchas herramientas y lenguajes que intentan utilizar los conceptos en diversos grados"

--Wolfgang Strigel

... Problemas en OO

- Un objeto contiene datos y operaciones que manipulan los datos, pero ...
- Podemos distinguir dos tipos de objetos **degenerados**:
 - Un objeto sin datos (que sería lo mismo que una biblioteca de funciones). Si los métodos son estáticos, “peor” aún.
 - Un objeto sin “operaciones”, con sólo atributos lo que permitiría crear, recuperar, actualizar y borrar su estado (que se correspondería con las estructuras de datos tradicionales)
- Un sistema construido con objetos degenerados **NO** es un sistema verdaderamente orientado a objetos

Diseño orientado a objetos

- Clases y objetos
- Diagramas de clases
- Diagramas de interacción entre objetos
- Patrones de diseño
- Herencia, polimorfismo, sobrecarga de operadores
- Etc.

PROGRAMACIÓN ORIENTADA A OBJETOS y MODELAMIENTO DE SOFTWARE

Se considera dado y conocido

- Repasemos. ¿Qué me pueden decir de diseño OO?

Diseño orientado a objetos

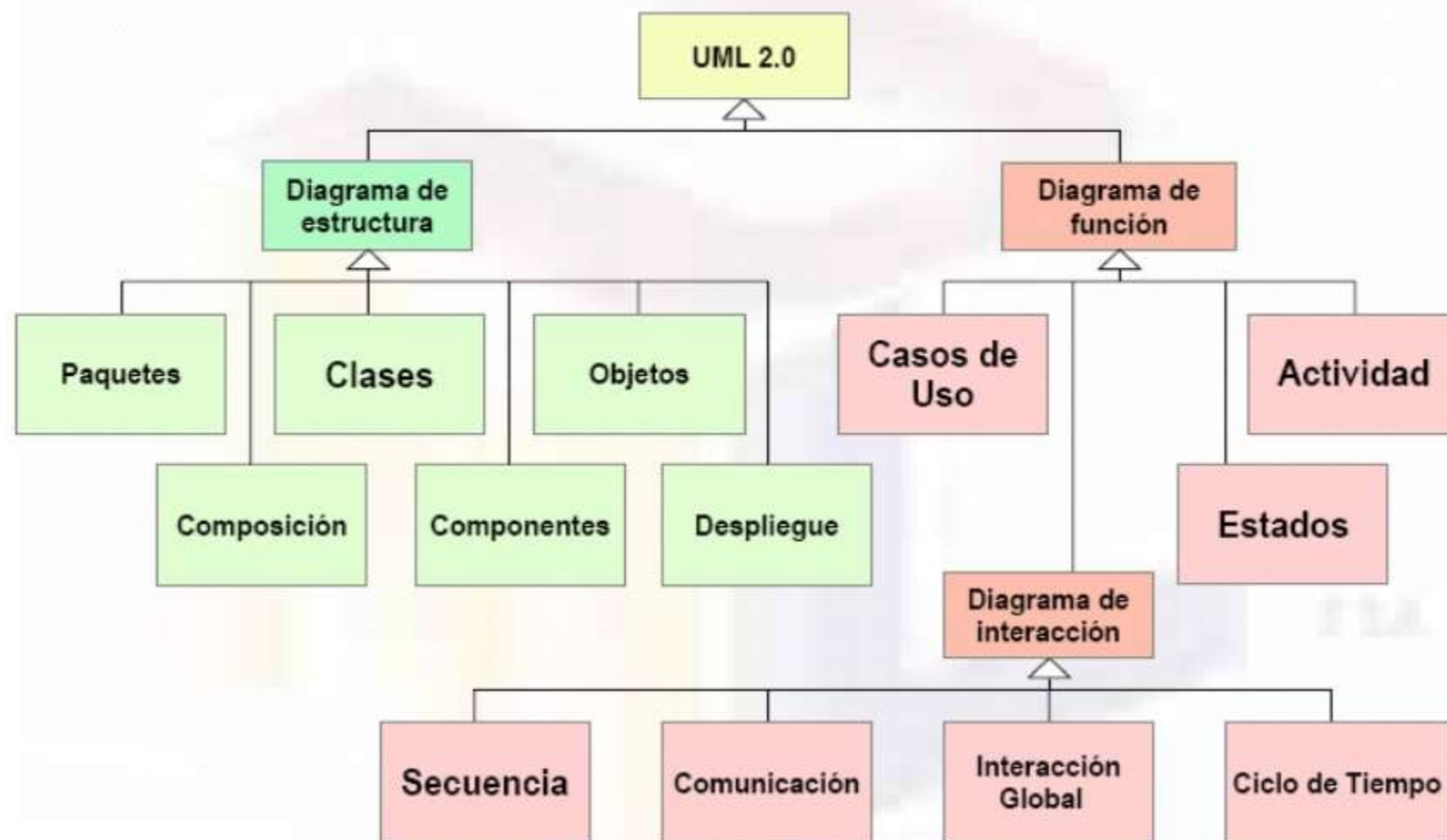
- Existen varias metodologías
 - Siempre es una actividad creativa por lo que no es un conjunto de pasos que mágicamente produce un diseño
- Se asume que:
 - Durante el diseño de la arquitectura se partió el sistema en varios subsistemas
 - Se va a producir un diseño orientado a objetos
- Entonces, el problema que se ataca es cómo producir un diseño orientado a objetos de un subsistema

Diseño orientado a objetos

- Un DOO completo es tal que en la fase de implementación sólo se agregan detalles
 - La mayoría de las clases y sus relaciones se identifican durante el diseño
- ¿Ustedes cómo diseñan?

Diagramas en UML

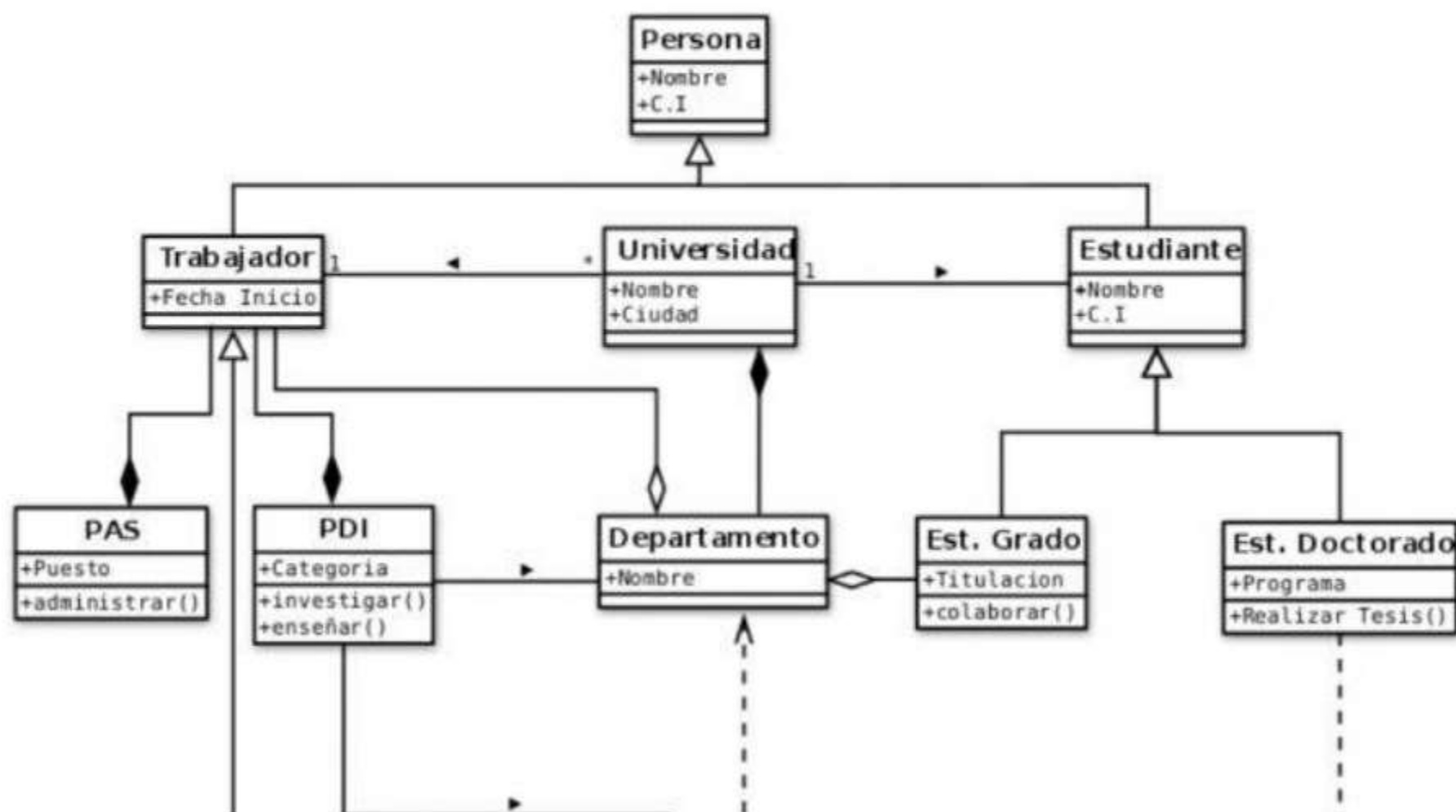
En UML 2.0 hay 13 tipos diferentes de diagramas.



UML – Diagrama de Clases

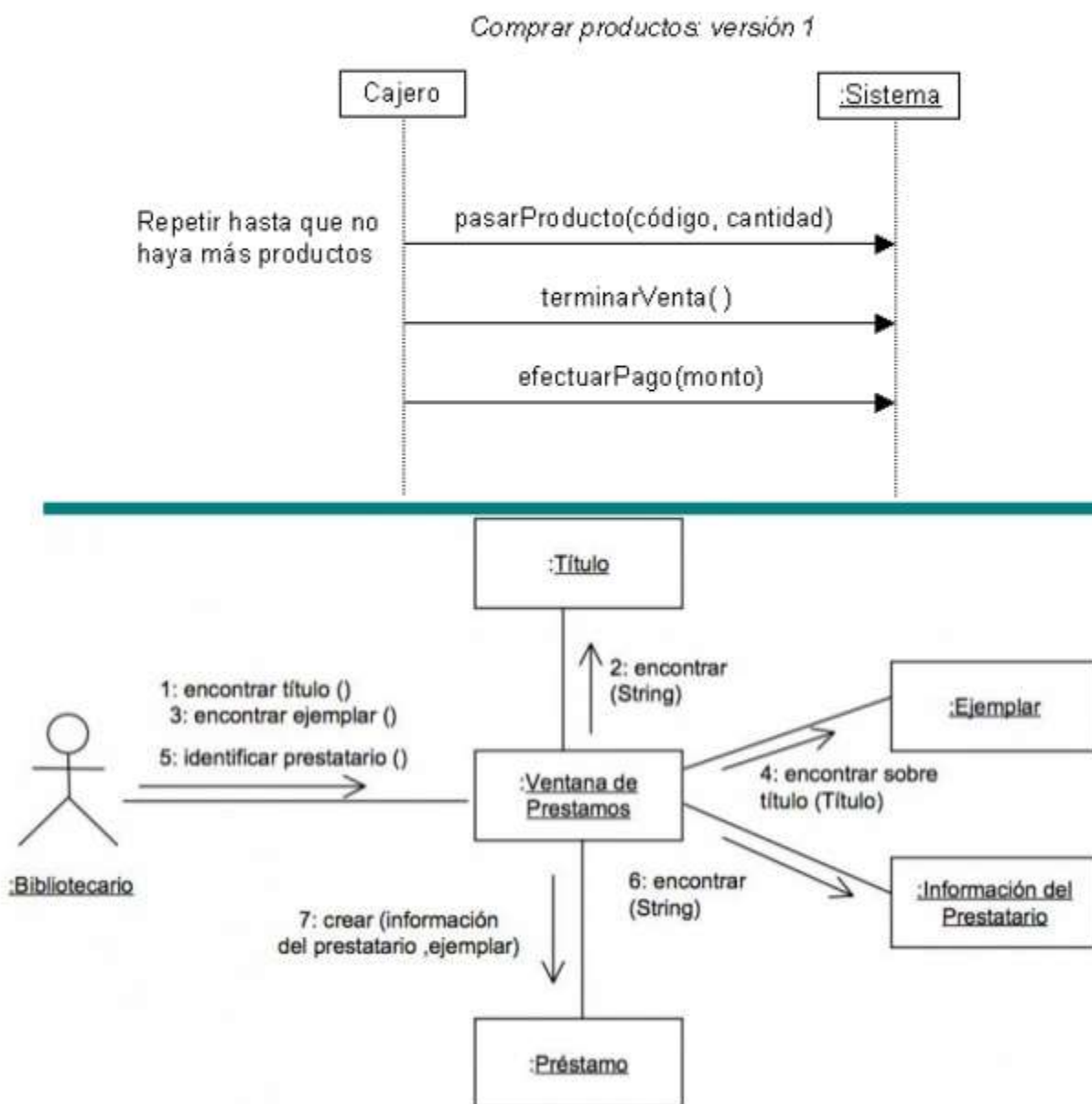
- Parte central del diseño
- Relación muy cercana con el código final
 - Herramientas que generan el esqueleto del código de forma automática
 - Se evitan defectos propios de la conversión manual
- Se describen
 - Clases (nombre, atributos, métodos)
 - Asociaciones entre clases
 - Relaciones de herencia

UML – Diagrama de Clases (2)



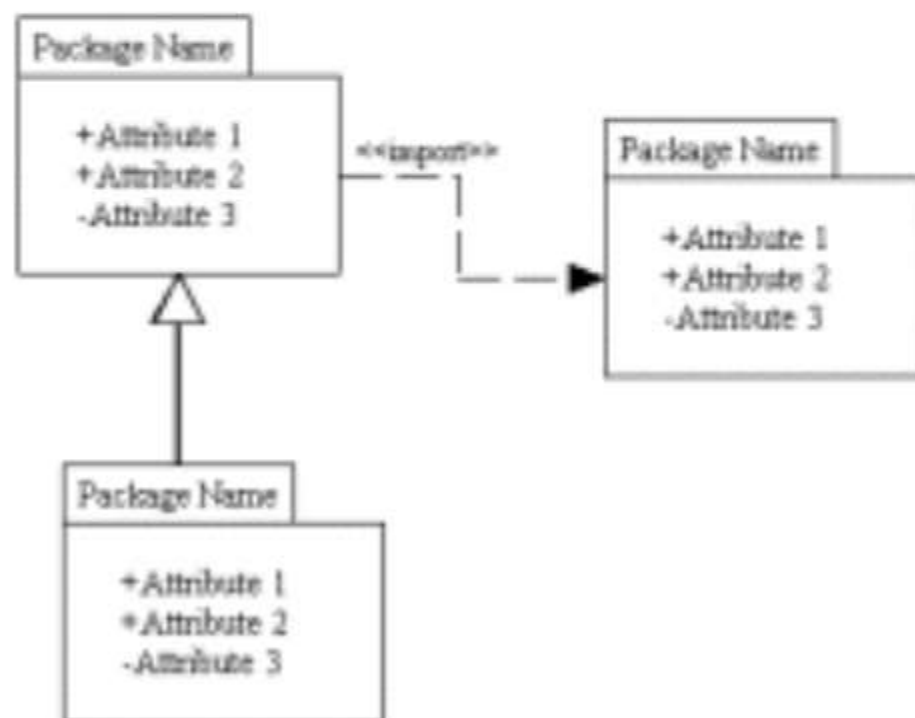
UML – Secuencia y Colaboración

- Comportamiento dinámico del sistema
- Muestra las interacciones entre los objetos para cumplir con cierta funcionalidad (normalmente un Caso de Uso)



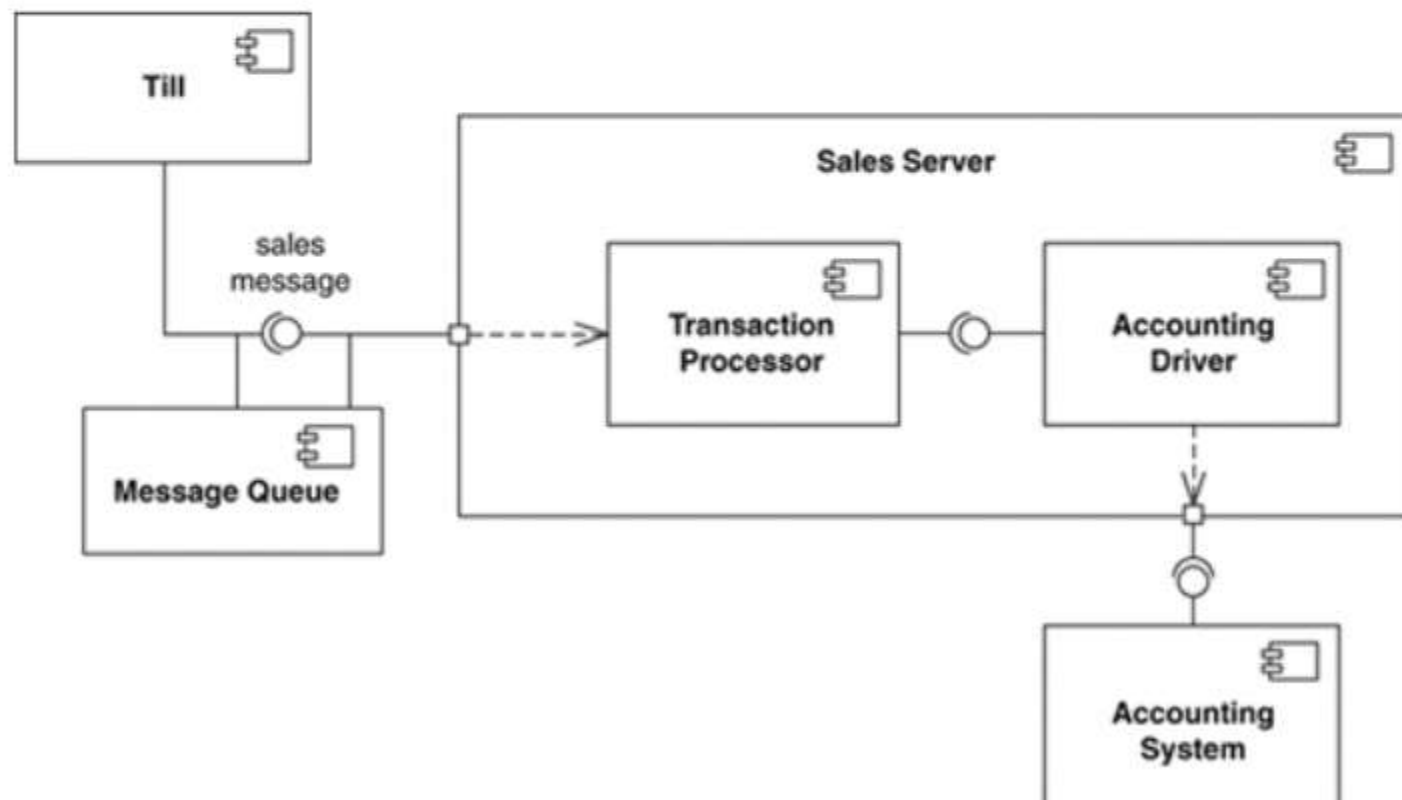
UML – Diagrama de Paquetes

- Permite agrupar otros elementos de UML
- Mecanismo de agrupación



UML - Componentes

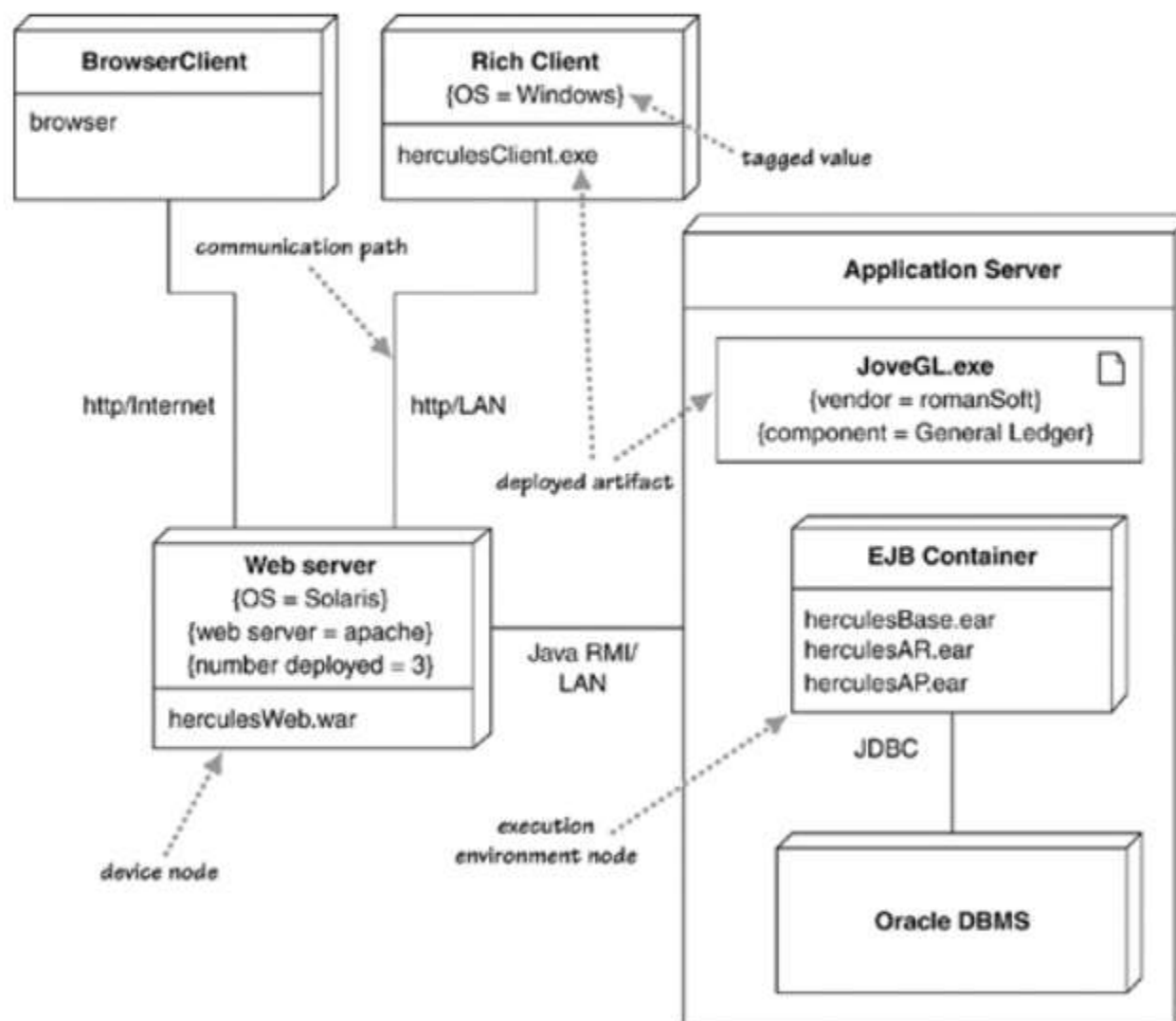
- Piezas independientes
- Estas piezas es bueno que sean actualizables (upgradeable)



UML - Despliegue

- Qué hardware usan los distintos elementos de software
- Nodo – Algo que puede alojar (host) software
 - Se identifica con un cubo
 - Dos tipos
 - Dispositivo – Representa hardware
 - Ambiente de ejecución – Representa software que aloja otro software
 - Dentro se pone lo que se despliega en ese nodo
 - Nodos que se comunican se representa mediante líneas

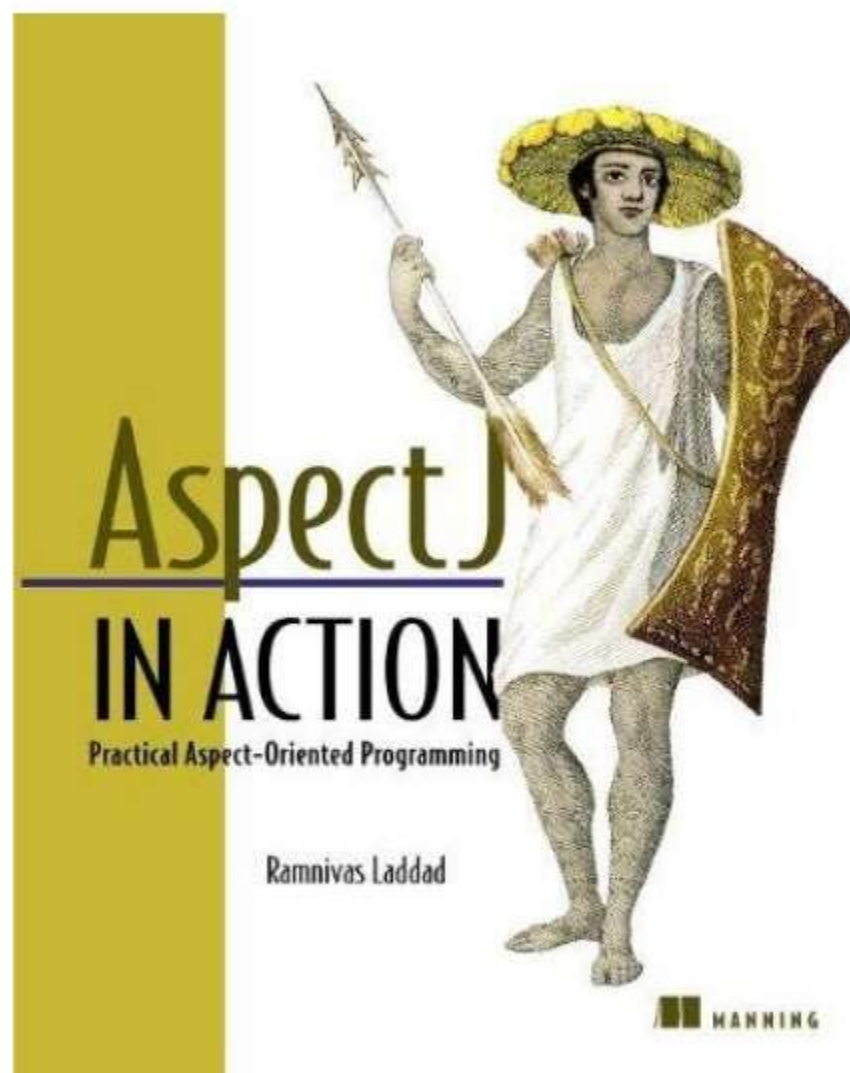
UML – Despliegue (2)



Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Bibliografía consultada



- R. Laddad, AspectJ in action. Greenwich, CT: Manning, 2004.
- G. Kiczales et al., "Aspect-oriented programming", 1997. Available: <https://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf>. [Accessed 10 October 2020].
- I. Sommerville, "Ingeniería de software", 9na Edición, Cap.21

Motivación

- En la mayoría de los grandes sistemas, las *relaciones entre los requerimientos y componentes del programa son **complejas***.
 - *Un solo requerimiento puede implementarse mediante algunos componentes, y*
 - *Cada componente puede incluir elementos de varios requerimientos.*
- En la práctica, esto significa que *implementar un cambio a los requerimientos implica comprender y modificar varios componentes.*

La separación de intereses

- La separación de competencias o intereses (*concerns*) es un principio clave del diseño e implementación de software.
 - Significa que usted debe organizar su software de modo que cada elemento en el programa (clase, método, procedimiento, etcétera) realice una función y sólo una función.
- En la realidad, resulta muy difícil identificar con exactitud lo que se entiende realmente por competencia.
 - Sin duda, las competencias son más que simplemente elementos funcionales, pero una definición más general es tan vaga, que en realidad resulta inútil.

Tipos de competencias o intereses

- **Funcionales**: funcionalidad específica a incluir en un sistema.
 - Por ejemplo, en un sistema de control ferroviario, una competencia funcional específica consiste en el frenado del tren.
- **De calidad del servicio**: comportamiento no funcional de un sistema.
 - Éstas incluyen características como rendimiento, fiabilidad y disponibilidad.
- **De política**: políticas generales que rigen el uso de un sistema.
 - Las limitaciones políticas incluyen competencias de seguridad y protección, y las competencias relacionadas con las reglas de negocio.
- **De sistema**: atributos del sistema como un todo, tales como su mantenibilidad o configurabilidad.
- **Organizacionales**: metas y prioridades de la organización.
 - Entre ellas se incluyen producir un sistema dentro de presupuesto, usar los activos de software existentes y mantener la imagen de la organización.

Competencias centrales y secundarias de un **gran** sistema

- **Centrales:** competencias funcionales relacionadas con su *propósito primario*.
 - Ejemplo: en un sistema de información de pacientes en un hospital, las competencias funcionales centrales son la creación, edición, recuperación y gestión de registros de pacientes.
- **Secundarias:** pueden incluir funcionalidad que comparte información con las competencias centrales, o que se requiere para que el sistema pueda cumplir con sus *requerimientos no funcionales*.
 - Ejemplo: Una competencia central de mantener un buffer compartido (*venta y compra de productos*) al agregar y eliminar elementos del mismo requiere de una competencia secundaria esencial de *sincronización* para garantizar que los procesos productor y consumidor no interfieran entre sí.
 - El sistema debe diseñarse de forma que el proceso productor no pueda sobrescribir datos que no se hayan consumido, ni el proceso consumidor pueda tomar datos de un buffer vacío.

Competencias transversales

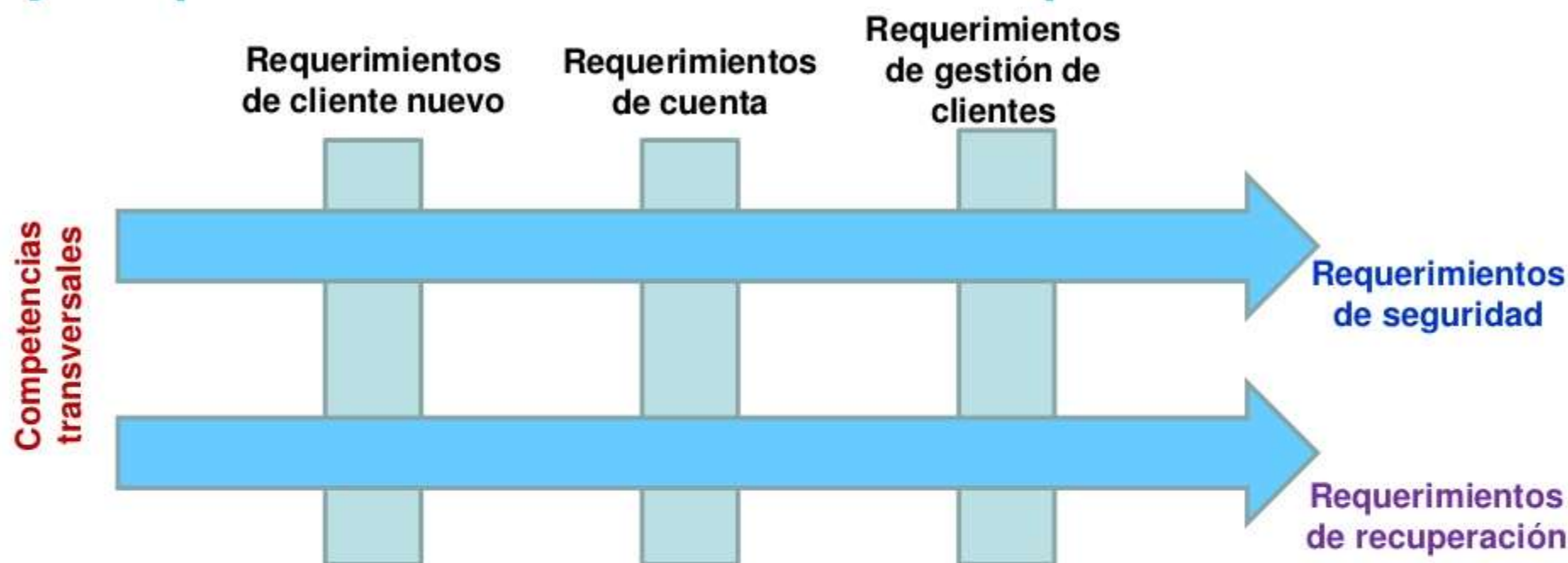
- Aparte de las competencias secundarias, otras competencias como **calidad de servicio** y **políticas organizacionales** reflejan requerimientos esenciales del sistema.
 - se aplican al sistema como un todo en lugar de implementarse a requerimientos individuales o a la realización de dichos requerimientos en un programa.
- Son llamadas **competencias transversales** (*cross-cutting*), para distinguirlas de las competencias centrales.
 - Las competencias funcionales secundarias también pueden ser transversales, aunque no siempre atraviesan todo el sistema; en vez de ello, se asocian con agrupamientos de competencias centrales que ofrecen funcionalidad relacionada.

Ejemplo: sistema de banca por Internet



- Este sistema tiene requerimientos relacionados con **nuevos clientes**, como **comprobación de crédito** y **verificación de dirección**.
- Asimismo, posee requerimientos relacionados con la **administración de los clientes existentes** y la **gestión de cuentas de los clientes**.
- Todas ellas son competencias centrales que se asocian con el propósito primario del sistema: *proveer un servicio de banca por Internet*.

Ejemplo: sistema de banca por Internet



- Sin embargo, el sistema también cuenta con:
 - requerimientos de seguridad con base en la política de seguridad del banco, y
 - requerimientos de recuperación para garantizar que los datos no se pierdan en caso de una falla del sistema.
- Se trata de **competencias transversales**, ya que pueden influir en la implementación de todos los otros requerimientos del sistema.

Competencias transversales

- Las abstracciones de lenguaje de programación, tales como **procedimientos y clases**, son el mecanismo que se usa normalmente para organizar y estructurar las **competencias centrales de un sistema**.
- No obstante, la implementación de las competencias centrales en lenguajes de programación convencionales incluye por lo general **código adicional para implementar las competencias transversales, funcionales, de calidad de servicio y de políticas**.
- Esto conduce a dos fenómenos indeseables: enredos (*tangling*) y dispersión (*scattering*).

Enredos (*tangling*)

Ocurre cuando un módulo en un sistema incluye *código que implementa diferentes requerimientos de sistema*.

El código que agrega un registro al buffer está enredado con código que implementa **sincronización**.

```
synchronized void put (SensorRecord rec )
{
    // Comprueba que haya espacio en el buffer; espera si no
    if ( numberOfEntries == bufsize)
        wait () ;
    // Agrega registro al final del buffer
    store [back] = new SensorRecord (rec.sensorId, rec.sensorVal) ;
    back = back + 1 ;
    // Si está al final del buffer, la siguiente entrada está al
    principio
    if (back == bufsize)
        back = 0 ;
    numberOfEntries = numberOfEntries + 1 ;
    // indica que el buffer está disponible
    notify () ;
} // put
```

El código de sincronización, que se asocia con la competencia secundaria de **asegurar exclusión mutua**, debe incluirse en **TODOS** los métodos que acceden al buffer compartido.

Dispersión (*scattering*)

Ocurre cuando la implementación de una competencia individual (un requerimiento lógico o un conjunto de requerimientos) se *dispersa a través de varios componentes del programa*

Dispersión de métodos que implementan competencias secundarias

Paciente	Imagen	Consulta
<decls atributo>	<decls atributo>	<decls atributo>
getName () editName () getAddress () editAddress () ... anonymize () ...	getModality () archive () getDate () editDate () ... saveDiagnosis () saveType () ...	makeAppoint () cancelAppoint () assignNurse () bookEquip () ... anonymize () saveConsult () ...

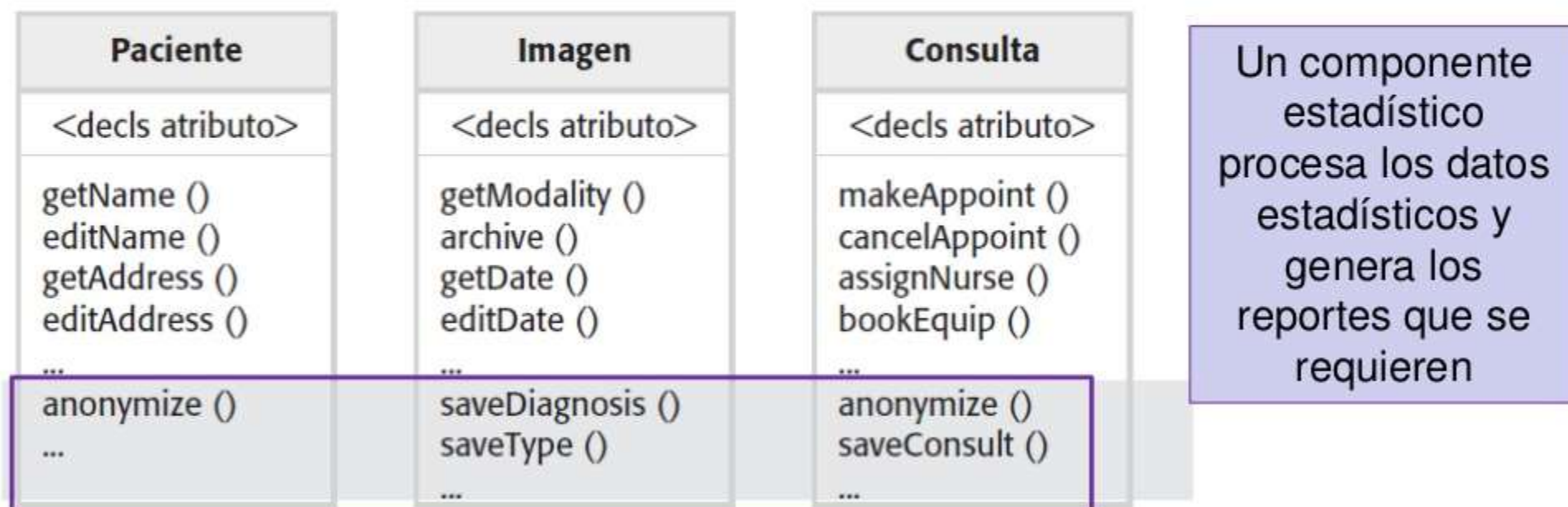
El proveedor de código de salud quiere registrar detalles de cuántos pacientes se admiten y se dan de alta cada mes, cuántos pacientes mueren, qué medicamentos se prescriben, las razones de las consultas, etcétera.

Dichos requerimientos tienen que implementarse agregando código que vuelva anónimos los datos (para mantener la privacidad de los pacientes) y los escriba en una base de datos estadística

Dispersión (*scattering*)

Un componente estadístico procesa los datos estadísticos y genera los reportes que se requieren.

Dispersión de métodos que implementan competencias secundarias



Como se observa, esta competencia estadística se dispersa a lo largo de las otras competencias centrales.

Enredos (*tangling*) y dispersión (*scattering*)

- Los problemas con la dispersión y el enredo ocurren cuando cambian los requerimientos iniciales del sistema.
 - Por ejemplo, suponga que deben recopilarse nuevos datos estadísticos en el sistema de registro de pacientes.
 - Los cambios al sistema no se ubican todos en un lugar y, por lo tanto, uno tiene que emplear tiempo buscando los componentes en el sistema que deban cambiarse.
- Para ello, es preciso modificar cada uno de estos componentes para incorporar los cambios requeridos. Esto puede ser costoso debido al tiempo que se necesita para analizar los componentes y, luego, para realizar y probar los cambios.
 - Siempre existe la posibilidad de que se pierda algo de código que se debe cambiar y, por consiguiente, las estadísticas serán incorrectas.
- Más aún, cuanto más severos sean los cambios que deban realizarse, aumentará la probabilidad de que se cometa una falla y se introduzcan errores en el software.

Desventajas de los diseños tradicionales

(Diseño Funcional y Orientado a Objetos)

- Unidades de descomposición no administran de buena forma aspectos como:
 - Gestión de memoria
 - Coordinación
 - Distribución
 - Ejecución en tiempo real
 - Sincronización
 - Manejo de errores
 - Gestión de la seguridad

Consecuencia:

- Nos encontramos con problemas de programación en los cuales ni las técnicas funcionales, ni las orientadas a objeto son suficientes para capturar todas las decisiones de diseño que el programa debe implementar.
- Las técnicas tradicionales no soportan bien la separación de competencias para *aspectos distintos de la funcionalidad básica de un sistema*, y esta situación claramente tiene un impacto negativo en la calidad del software.



1ª Generación:
Código espagueti

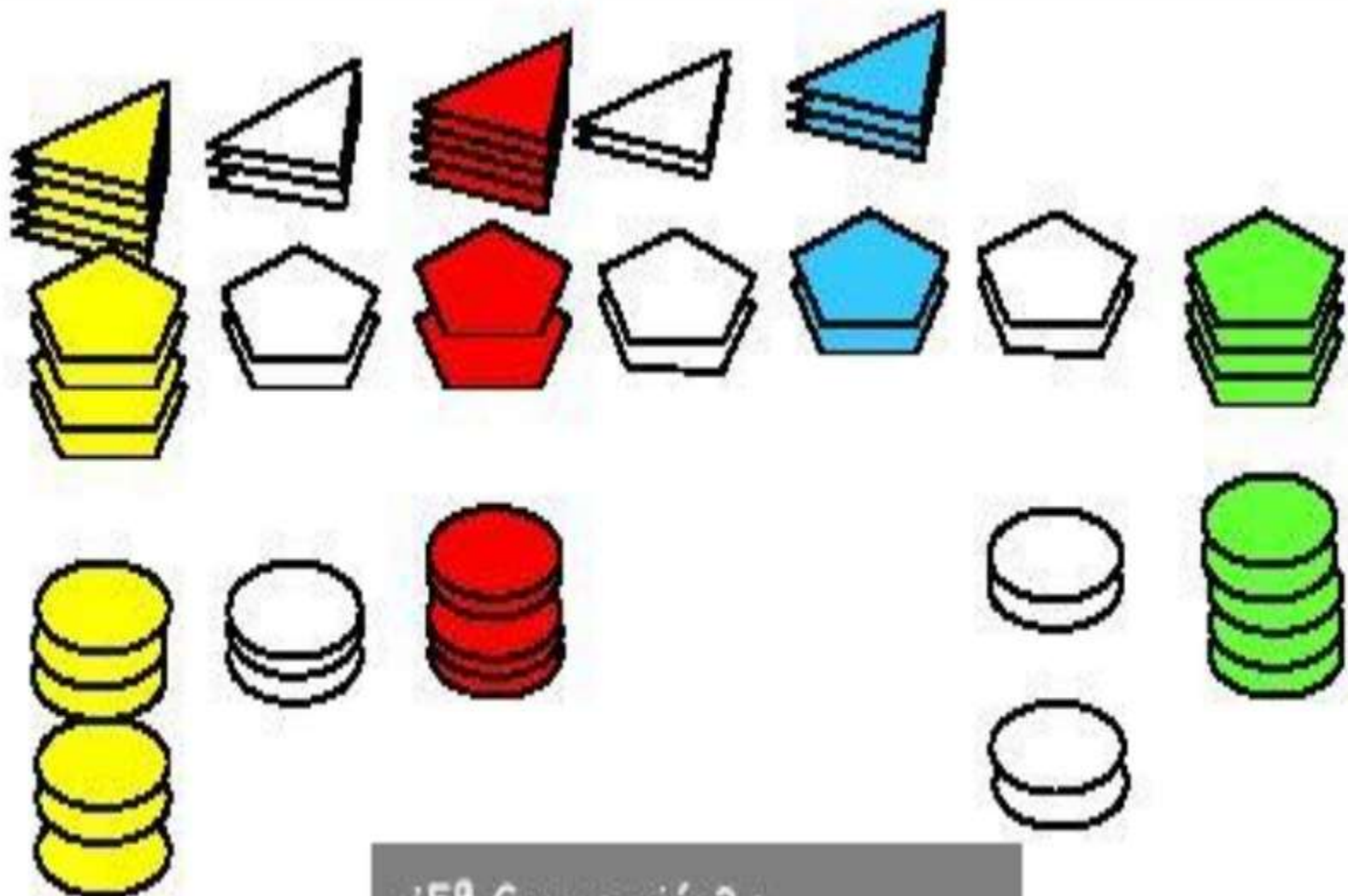


2ª y 3ª Generación:
Descomposición
funcional

Software = Datos (formas)
+ Funciones (colores)



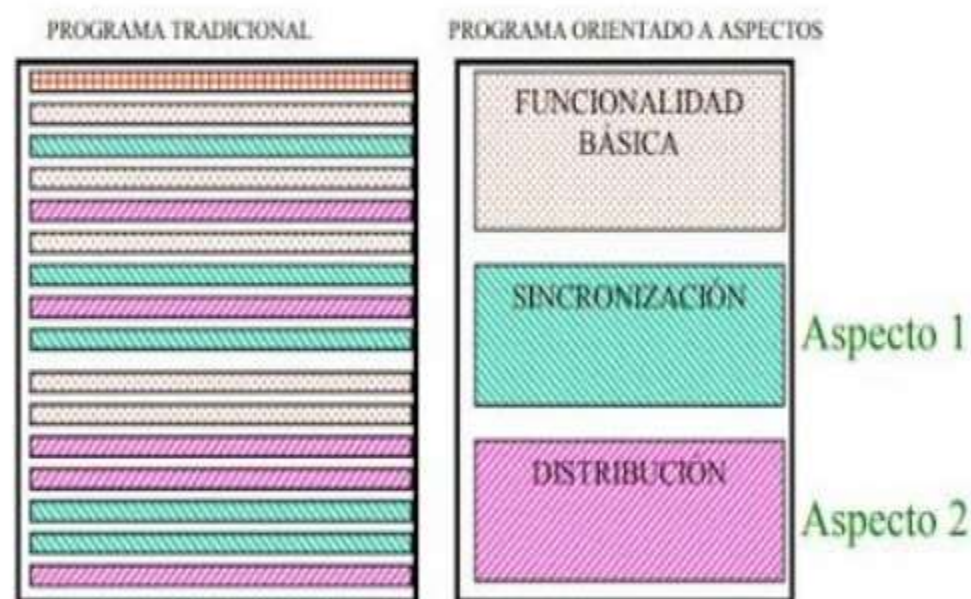
4ª Generación:
Descomposición en
objetos



¿5ª Generación? :
Descomposición de aspectos

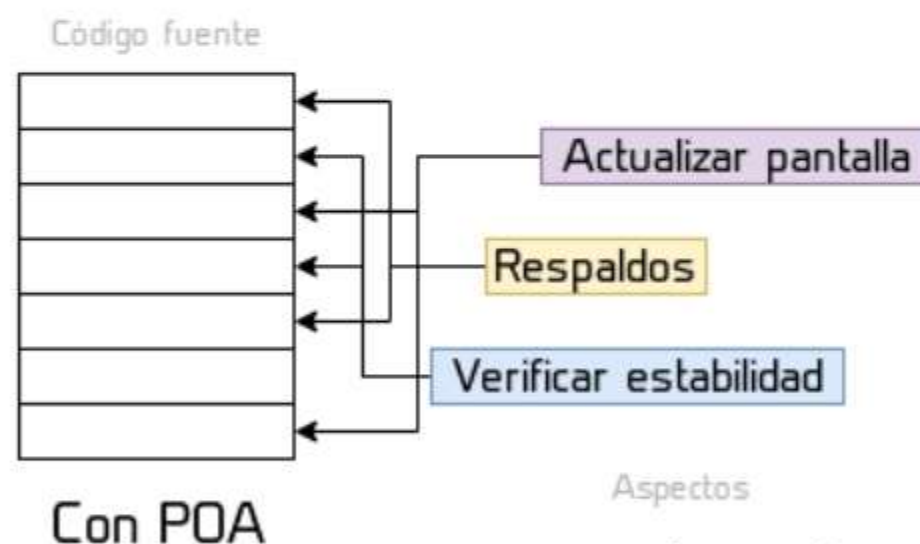
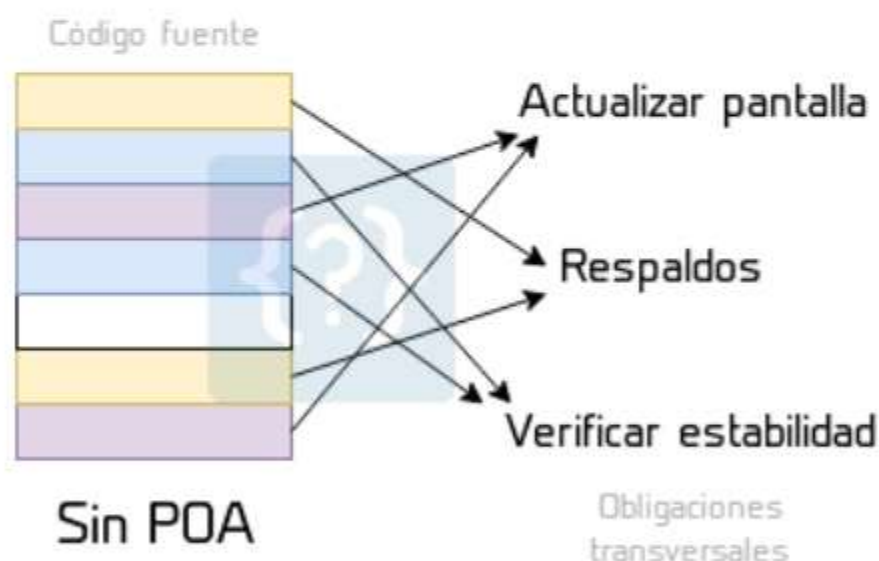
P.O.A.

- La programación orientada a aspectos (POA) aspira a soportar la *separación de competencias* para los aspectos antes mencionados.
- Intenta separar los componentes y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema.



Ejemplo de P.O.A.

Ejemplo de funcionamiento de la programación orientada a aspectos (POA)



codingornot.com

Tarea: Comprensión lectora

- Leer el artículo y redactar un resumen de máximo 1 página con los aspectos más importantes del DSOA.
- El resumen deberá ser fruto de vuestra propia síntesis.

El Desarrollo de Software Orientado a Aspectos: Un Caso Práctico para un Sistema de Ayuda en Línea

Aspect-Oriented Software Development: A Practical Case for an On-line Help Desk System

Marta S. Tabares B. Ph.d.,¹ Germán H. Alferez Salinas MSc.,² Edward M. Alferez Salinas MSc.³

1. Escuela de Ingeniería de Antioquia, Colombia

2. Universidad de Morelos, México

3. Universidade Nova de Lisboa, Portugal

pfmstabares@eia.edu.co; harveyalferez@um.edu.mx; mauricio.alferez@di.fct.unl.pt

Recibido para revisión 15 de Marzo de 2008, Aceptado 19 de Mayo de 2008, Versión final 21 de Mayo de 2008

Resumen—El Desarrollo de Software Orientado a Aspectos (DSOA) provee un conjunto de enfoques para identificar, modularizar e implementar intereses o propiedades del sistema que pueden cruzar otros intereses del sistema. También busca mejorar el entendimiento de cada interés del sistema de forma clara y separada desde las primeras etapas del ciclo de vida de software. Este se orienta a la obtención de productos de software de calidad con partes más reutilizables y que evolucionen fácilmente en el tiempo. En este artículo, se presenta un caso de estudio para ilustrar la aplicación del DSOA desde etapas tempranas del desarrollo de software hasta la implementación. Diferentes enfoques orientados por aspectos se aplican para facilitar el manejo separado de intereses dentro de

1. INTRODUCCIÓN

La Orientación a Aspectos (OA) surgió de necesidades muy concretas desde el nivel de programación. Define un mecanismo que ayuda a resolver problemas complementarios de código disperso (scattered) y enmarañado (tangled) que no se resuelven fácilmente usando paradigmas de desarrollo de software tradicionales como la orientación a objetos. Provee una unidad modular llamada Aspecto y un mecanismo de composición que permite entremezclar unidades modulares de comportamiento común con otras unidades modulares básicas

Objetivo principal de la POA

Brindar un contexto al programador que permita **separar claramente componentes y aspectos**, separando componentes entre sí, aspectos entre sí, y aspectos de componentes, a través de mecanismos que hagan posible abstraerlos y componerlos para producir el sistema completo

<i>Tecnología</i>	<i>Conceptos Claves</i>	<i>Constructores</i>
Programación estructurada	Constructores de control explícitos	Do, while, y otros iteradores
Programación modular	Ocultamiento de Información	Módulos con interfaces bien definidas
Abstracción de datos	Ocultar la representación del dato	Tipo de dato abstracto
Programación orientada a objetos	Objetos con clasificación y especialización	Clases, objetos, polimorfismo
Programación orientada a aspectos.	"6 C" y "4 S"	Aspectos

Objetivos específicos

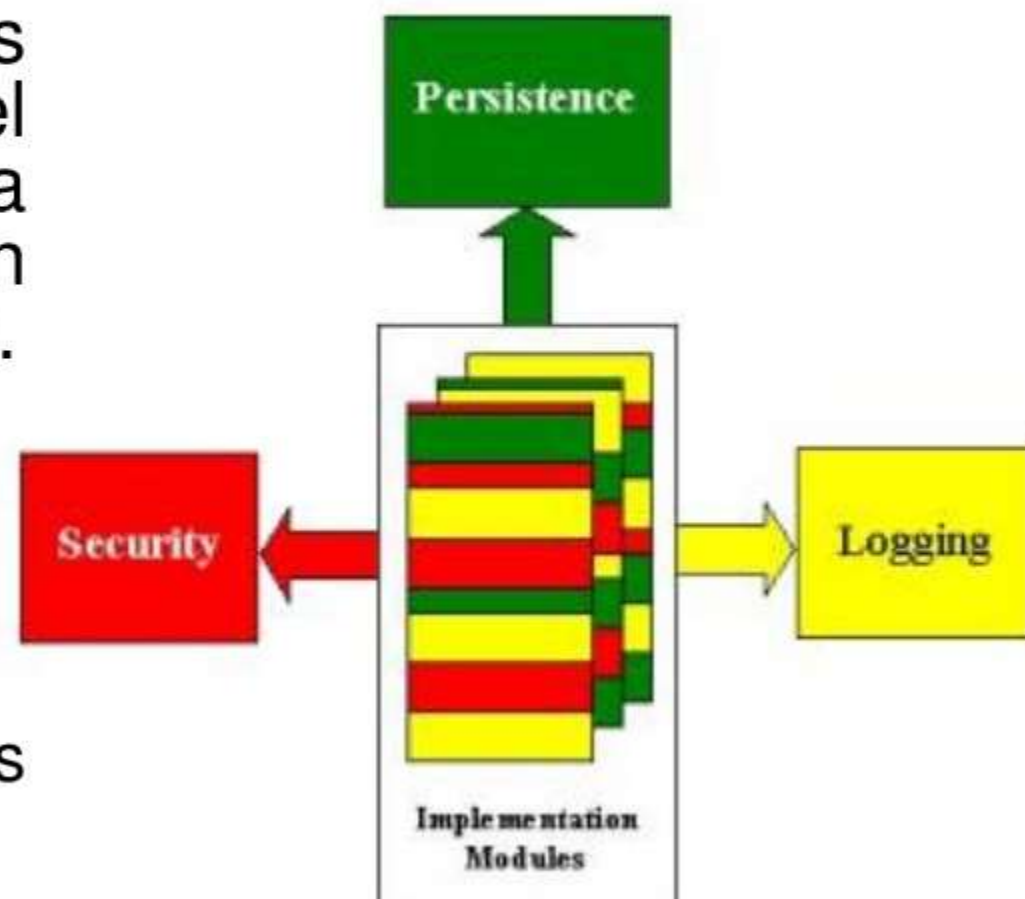
- Dentro de los objetivos específicos se tienen los siguientes:
 - Tener un código **menos enmarañado**, más natural y más reducido.
 - Una mayor facilidad para razonar sobre las materias, ya que están **separadas** y tienen una **dependencia mínima**.
 - Tener más facilidad para depurar y hacer modificaciones en el código.
 - Conseguir que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.
 - Tener un código más reusable y que se puede acoplar y desacoplar cuando sea necesario.

Conceptos claves (1)

- **CONCEPTO (Concern):** Es un requerimiento específico que debe ser implementado para satisfacer las necesidades del sistema.
 - Así como en la ingeniería de software se propone una clasificación para los requerimientos (funcionales y no funcionales), la POA también propone que los conceptos o *concerns* se pueden clasificar en **componentes** y **aspectos**.
- **COMPONENTE (Component):** Son las partes del código que pueden ser encapsuladas claramente.
 - Estas partes suelen estar relacionadas con las funcionalidades del sistema.
- **ASPECTO (Aspect):** “es una **unidad modular que se disemina por la estructura de otras unidades funcionales**. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. (G. Kiczales).”

Conceptos claves (2)

- Por lo general, los **aspectos** no suelen hacer parte de las funcionalidades del sistema, sino más bien a propiedades que afectan el desempeño del mismo.
- **Ejemplos de aspectos:**
 - Uso de memoria
 - Persistencia de datos
 - Seguridad
 - Sistemas de registros (logger)
 - Manejo de errores



Puntos de corte (pointcut) vs puntos de enlace (joinpoints)

Cuando sales a comer a un restaurante, estando en él, miras el menú y hay varias opciones que se pueden escoger de platos fuertes, ensaladas, postres, etc. Después de ver que hay en el menú escoges uno o mas de estos platos, pero hasta que estos no sean ordenados y el mesero no te los traiga son solo "oportunidades para cenar".



Los **joinpoints** son las opciones que hay en el menu y los **pointcuts** son los platos que se ordenan.

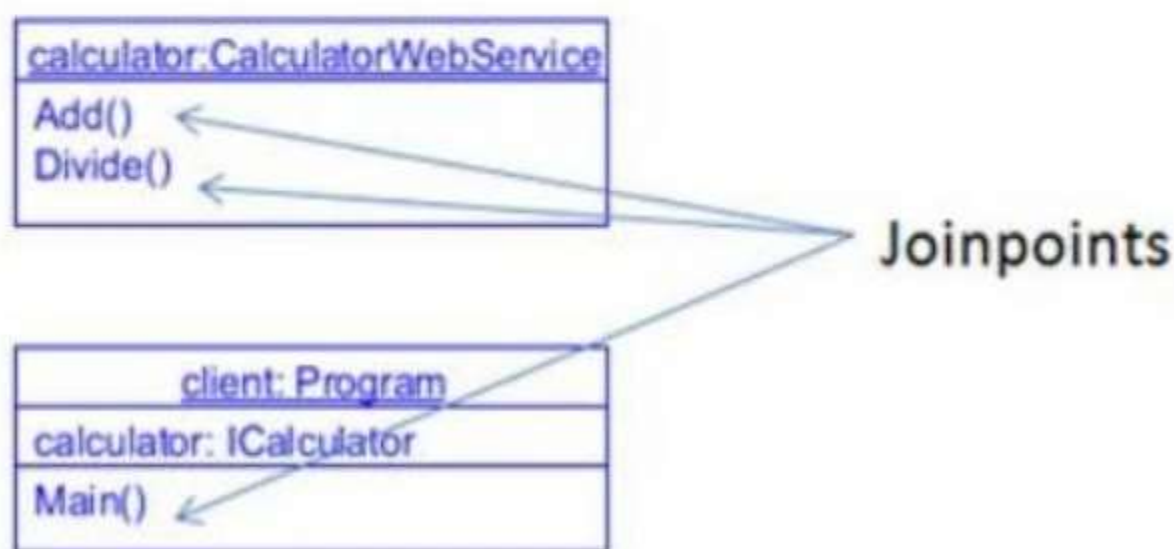
Hablando dentro del ámbito de la programación, un **joinpoint** es una oportunidad dentro del código para la aplicación de un ASPECTO; Una vez se toma esa oportunidad y se selecciona uno o mas joinpoints, al aplicar un ASPECTO de ellos se tendrá un **pointcut**.

Conceptos claves (3)

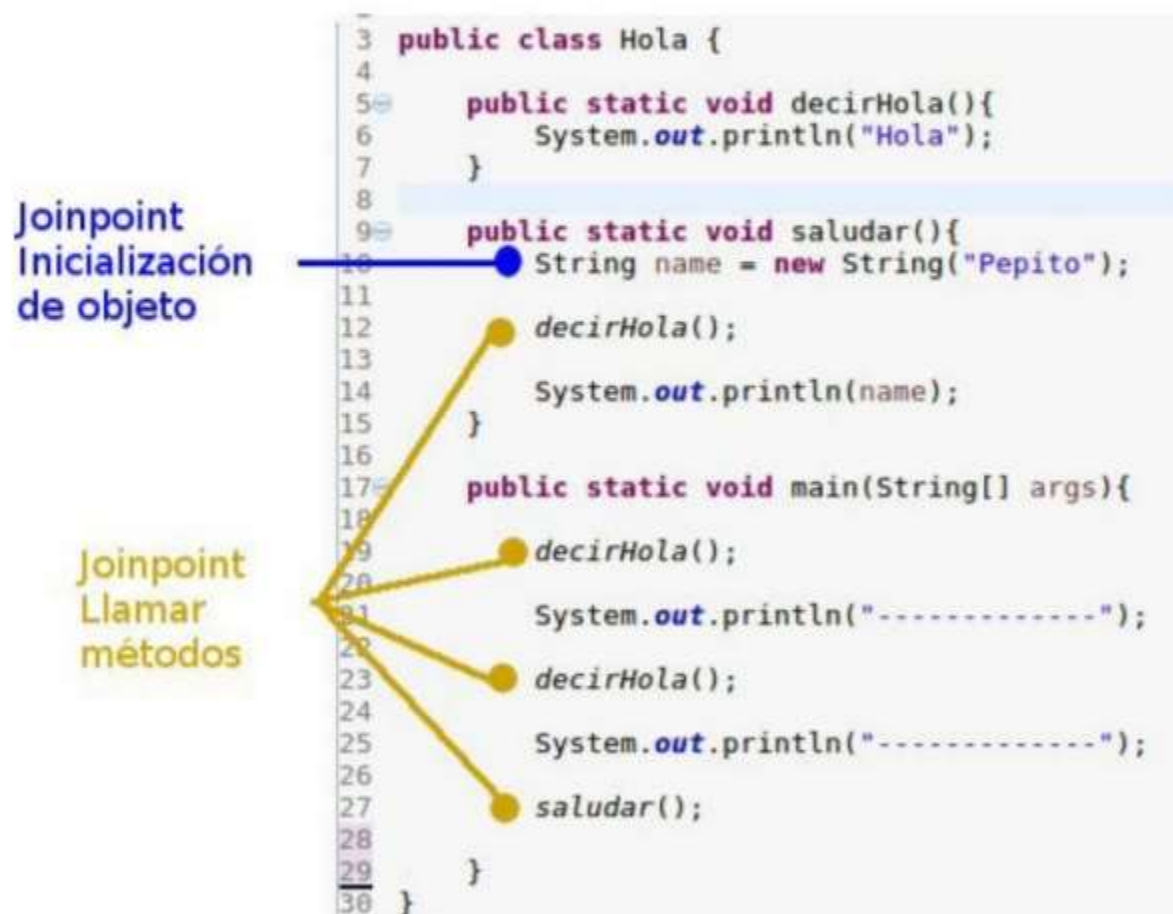
- De manera resumida:
 - Un **Joinpoint** es un lugar individual en donde se puede ejecutar el código. P.ej. "Cuando un método arroja una excepción".
 - Un **punto de corte** es una colección o conjunto de Joinpoints. P.ej. "Cuando un método en la clase Foo arroja una excepción".

Conceptos claves (4)

- **PUNTO DE UNIÓN/ENLACE (Joint point):** Punto de ejecución dentro del sistema donde un aspecto puede ser conectado:
 - Llamada a un método
 - Lanzamiento de una excepción
 - Modificación de un campo.



Conceptos claves (5)

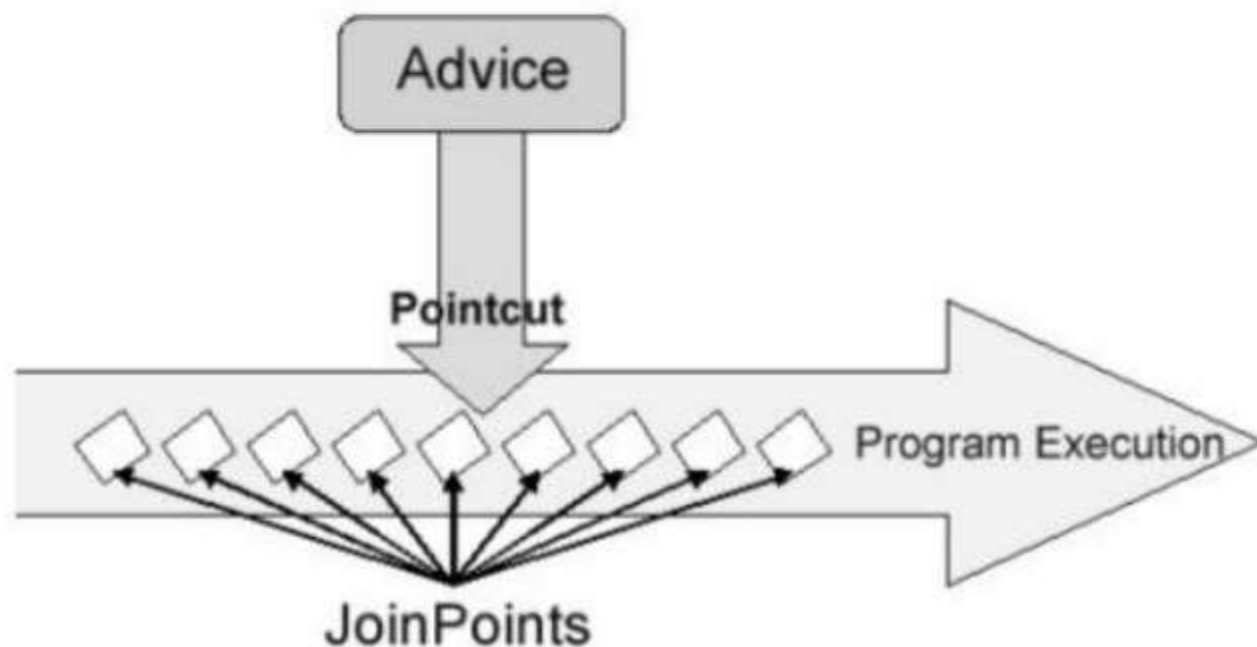


Joint points por Categoría

Categoría de Join Point	Sintaxis
Ejecución de Método	execution(FirmaMetodo)
Llamada a Método	call(FirmaMetodo)
Ejecución de Constructor	execution(FirmaConstructor)
Llamada a Constructor	call(FirmaConstructor)
Inicialización de Clase	staticinitialization(FirmaTipo)
Lectura de un atributo	get(FirmaAtributo)
Escritura de un atributo	set(FirmaCampo)
Ejecución de manejo de Excepción	handler(FirmaTipo
Inicialización de Objeto	initialization(FirmaConstructor)
Pre-Inicialización de Objetos	preinitialization(FirmaConstructor)
Ejecución del Advice	adviceexecution()

Conceptos claves (6)

- **PUNTOS DE CORTE (Pointcut):** Define los Consejos que se aplicarán a cada Punto de Cruce. Un pointcut es un **predicado o condición para la aplicación de un aspecto**.



Algunos Pointcut importantes

Pointcut	Descripción
within(PatronTipo)	Cualquier join point dentro del scope de la clase indicado o uno de sus subtipo.
withincode(PatronTipo)	Cualquier join point dentro del scope de la clase indicada, o clases anidadas.
this(FirmaTipo)	Todos los join points donde la referencia "this" es del tipo indicado, o sus subclases.
target(FirmaTipo)	Todos los join points donde el objeto en el que se invoca el método es del tipo indicado o sus subclases.
args(PatronTipo)	Los join points en todos los métodos donde se reciban los argumentos indicados.
if(condiciones)	Todos los join points donde la condición indicada se cumple.

Conceptos claves (7)

- **CONSEJO(Advice):** Es la implementación del aspecto, es decir, contiene el **código que implementa la nueva funcionalidad**. Se insertan en la aplicación en los Puntos de unión (Joint point).
- Dentro de ellos podemos encontrar:
 - **before** : Esta anotación ejecuta un **advice** antes de la ejecución del punto de corte especificado. Ej: justo antes de entrar al getter, o al constructor etc.
 - **after returning**: Esta anotación ejecuta un **advice** después de la ejecución del punto de corte especificado, siempre que el método del punto de corte retorne de forma correcta (sin generar errores).
 - **after throwing**: Esta anotación ejecuta un **advice** después de la ejecución del punto de corte especificado, si el método del punto de corte genera una excepción. Podemos tanto acceder a la excepción generada como restringir el tipo de las excepciones que nos interesan.
 - **after**: Esta anotación ejecuta un **advice** tras la ejecución de un método, sin importar si fue por exception o por flujo normal (return)
 - **around**: Cuando el flujo llega a ejecutar el join-point, permite que nuestra lógica pueda determinar si proceder o no mediante el llamado al método proceed().

Conceptos claves (8)

Otro
Jointpoint

```
3 public class Hola {  
4  
5 public static void decirHola(){  
6     System.out.println("Hola");  
7 }  
8  
9 public static void saludar(){  
10     String name = new String("Pepito");  
11  
12     decirHola();  
13  
14     System.out.println(name);  
15 }  
16  
17 public static void main(String[] args){  
18  
19     decirHola();  
20  
21     System.out.println("-----");  
22  
23     decirHola();  
24  
25     System.out.println("-----");  
26  
27     saludar();  
28 }  
29 }  
30 }
```

Joinpoint(s)
del PointCut
callDecirHola

Advice

```
1 public aspect AspectoHola {  
2  
3  
4     pointcut callDecirHola(): call(* Hola.decirHola());  
5  
6     before() : callDecirHola(){  
7         System.out.println("Anter de llamar decirHola()");  
8     }  
9  
10    after() : callDecirHola(){  
11        System.out.println("Despues de llamar decirHola()");  
12    }  
13 }  
14 }  
15 }
```

Ejemplo 1: Manejo de empleados en tienda

http://ferestrepoca.github.io/paradigmas-de-programacion/poa/poa_teor%C3%ADa/Pages/ejemplos.html

El administrador de la tienda va a tener todo el manejo de los empleados, incluyendo todas las operaciones CRUD (Create, Read, Update, Delete), hasta cómo será el inicio de sesión a la aplicación por parte de los empleados

Este inicio de sesión o Login será el Aspecto que tendrá nuestra aplicación.

```
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Component;

@Component
public class EmployeeManager {

    public EmployeeDTO getEmployeeById(Integer employeeId) {
        System.out.println("Method getEmployeeById() called");
        return new EmployeeDTO();
    }

    public List<EmployeeDTO> getAllEmployee() {
        System.out.println("Method getAllEmployee() called");
        return new ArrayList<EmployeeDTO>();
    }

    public void createEmployee(EmployeeDTO employee) {
        System.out.println("Method createEmployee() called");
    }

    public void deleteEmployee(Integer employeeId) {
        System.out.println("Method deleteEmployee() called");
    }

    public void updateEmployee(EmployeeDTO employee) {
        System.out.println("Method updateEmployee() called");
    }
}
```

Join points

Aspect

```
@Aspect
public class EmployeeCRUDAspect {

    @Before("execution(* EmployeeManager.getEmployeeById(..))")
    public void logBeforeV1(JoinPoint joinPoint) {
        System.out.println("EmployeeCRUDAspect.logBeforeV1() : " + joinPoint.getSignature().getName());
    }

    @Before("execution(* EmployeeManager.*(..))")
    public void logBeforeV2(JoinPoint joinPoint) {
        System.out.println("EmployeeCRUDAspect.logBeforeV2() : " + joinPoint.getSignature().getName());
    }

    @After("execution(* EmployeeManager.getEmployeeById(..))")
    public void logAfterV1(JoinPoint joinPoint) {
        System.out.println("EmployeeCRUDAspect.logAfterV1() : " + joinPoint.getSignature().getName());
    }

    @After("execution(* EmployeeManager.*(..))")
    public void logAfterV2(JoinPoint joinPoint) {
        System.out.println("EmployeeCRUDAspect.logAfterV2() : " + joinPoint.getSignature().getName());
    }
}
```

Pointcut

Advice

Spring AOP + AspectJ

Ejemplo 2: Autenticación en sistema clínico

- Imagine que ocurre una violación de la seguridad y se modifica maliciosamente información del paciente.
 - Tal vez alguien dejó por accidente su computadora conectada al sistema y una persona no autorizada entró al mismo.
 - Alternativamente, una persona autorizada puede tener acceso y modificar maliciosamente la información del paciente.
- Para reducir la probabilidad de que esto ocurra nuevamente, se introduce una nueva política de seguridad.
 - Antes de realizar cualquier cambio a la base de datos del paciente, la persona que solicita el cambio de nuevo debe autenticarse en el sistema.
 - Los detalles de quién hace el cambio también se registran en un archivo aparte.
 - Esto ayudará a rastrear los problemas si volvieran a ocurrir.

Ejemplo 2: Autenticación en sistema clínico

```
aspect authentication {  
  before: call (public void update* (..)) // éste es un punto de corte  
  {  
    // éste es el consejo -advice- que debe ejecutarse cuando se teje  
    // -weaving- en el sistema en ejecución  
    int tries = 0 ;  
    string userPassword = Password.Get ( tries ) ;  
    while (tries < 3 && userPassword != thisUser.password ( ) ) {  
      // permite 3 intentos para ingresar la contraseña correcta  
      tries = tries + 1 ;  
      userPassword = Password.Get ( tries ) ;  
    }  
    if (userPassword != thisUser.password ( )) then  
      //si la contraseña es equivocada, supone que el usuario olvidó  
      //su contraseña  
      System.Logout (thisUser.uid) ;  
  }  
} // authentication
```

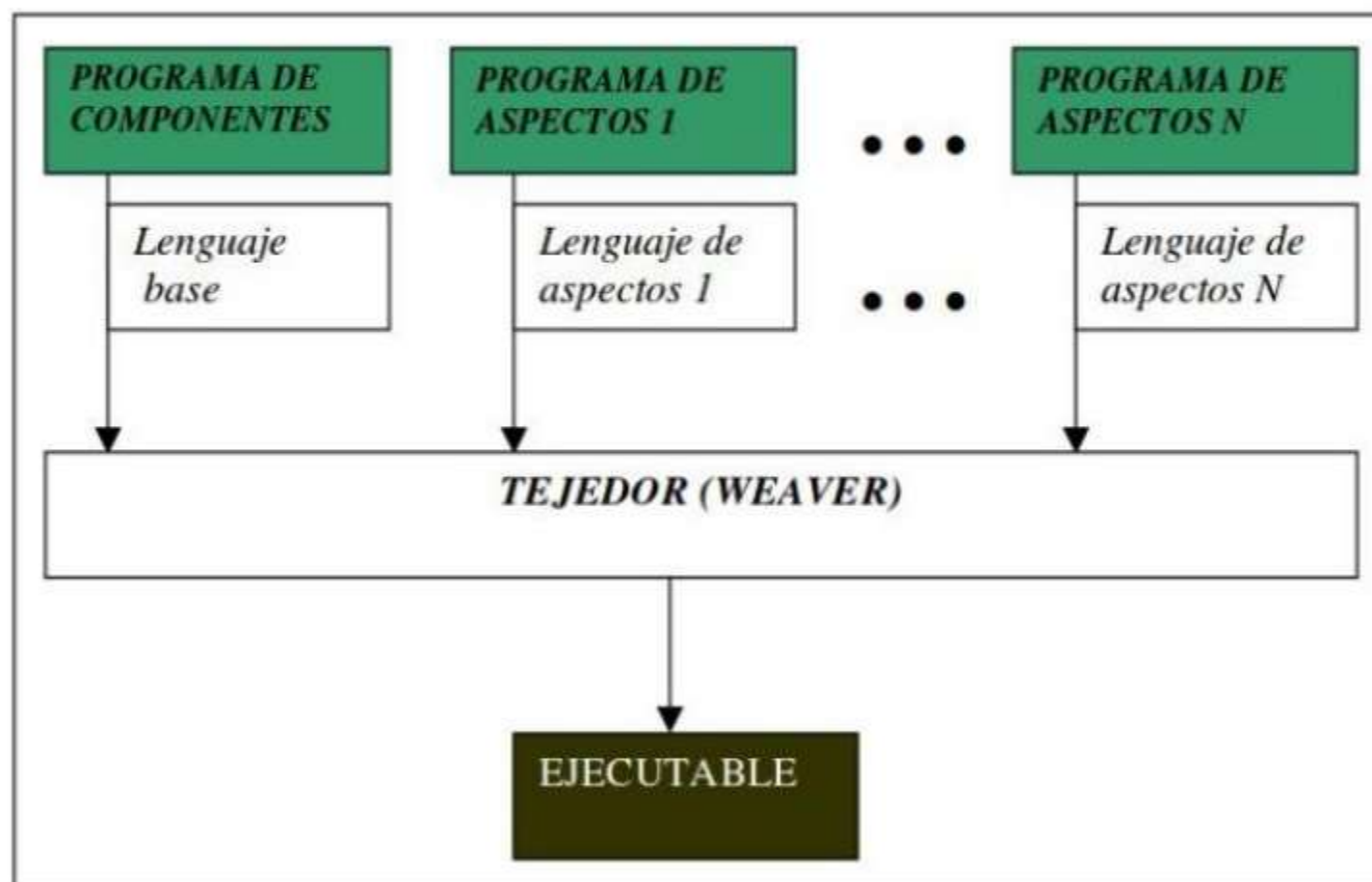
```
public void updateXid(string id, string newName, etc...)
```

Joint-point

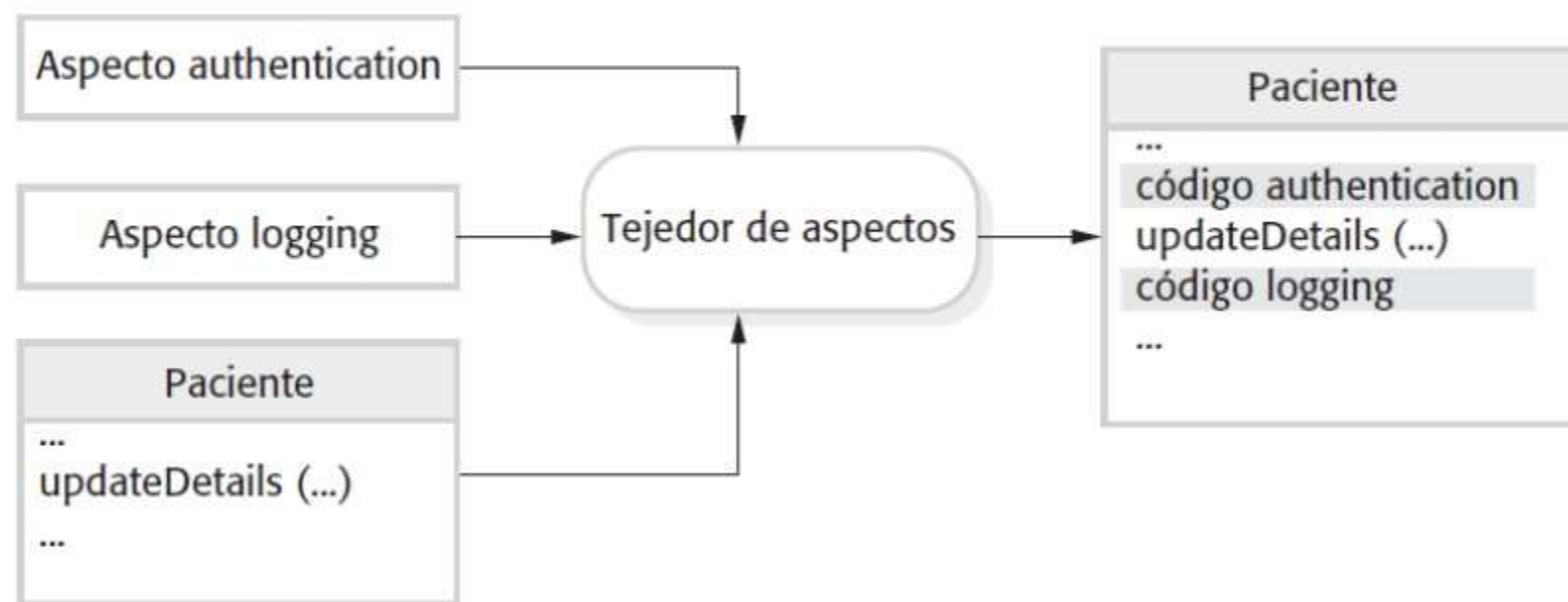
Conceptos claves (9)

- **TEJEDOR (Weaver):** El tejedor tiene un papel similar al del compilador en otros paradigmas, se encarga de **mezclar los diferentes componentes con los aspectos**, ayudándose de las reglas de recomposición que proporcionan los puntos de enlace.
 - El resultado de este proceso de tejido (o weaving) es un código ejecutable de todo el sistema.
 - Este proceso puede ocurrir a lo largo del ciclo de vida del programa:
 - Aspectos en Tiempo de Compilación.
 - Aspectos en Tiempo de Carga, los Aspectos se implementan cuando el Objeto Destinatario es cargado.
 - Aspectos en Tiempo de Ejecución.

Conceptos claves (10)



Ejemplo: Autenticación en sistema clínico



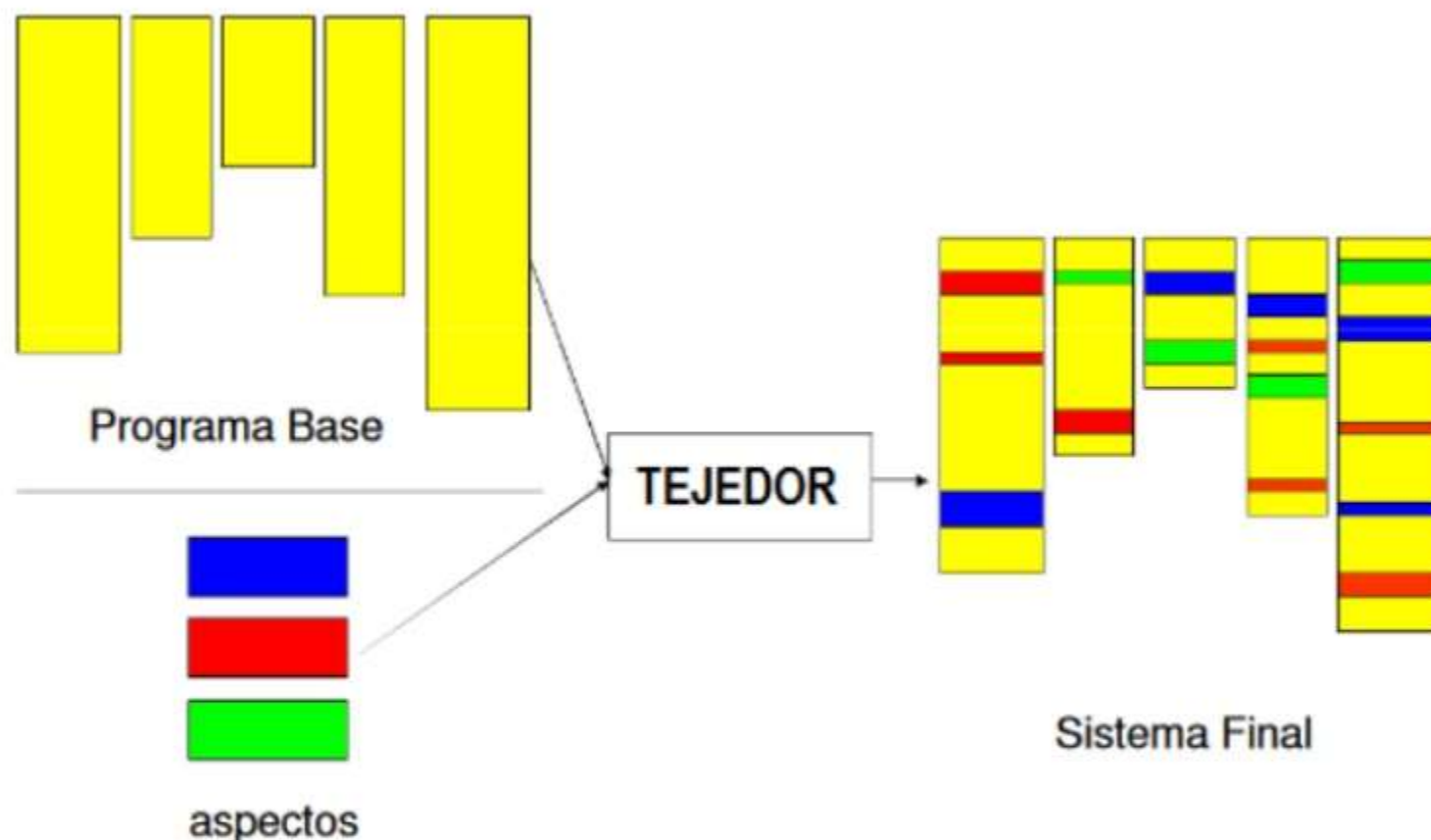
Tipos de Entrelazado (1/2)

- **Preprocesamiento de código fuente**, en el que un tejedor toma entrada de código fuente y genera nuevo código fuente en un lenguaje como Java o C++, que luego pueden compilarse usando el compilador de lenguaje estándar.
 - Este enfoque se adoptó para el lenguaje AspectX con su asociado XWeaver (Birrer *et al.*, 2005).
- **Tejido de tiempo de vinculación**, en el que el compilador se modifica para incluir un tejedor de aspectos. Un lenguaje orientado a aspectos, como **AspectJ**, se procesa y genera bytecode Java estándar.
 - Entonces éste puede ejecutarse directamente mediante un intérprete Java o procesarse aún más para generar código de máquina nativo.
- **Tejido dinámico en tiempo de ejecución**. En este caso, se monitorizan los puntos de enlace y, cuando ocurre un evento referenciado en un punto de corte, se integra el consejo correspondiente con el programa en ejecución.

Tipos de Entrelazado (2/2)

- El enfoque usado **más comúnmente** para el tejido de aspectos es el **tejido de tiempo de vinculación**, pues esto permite la implementación eficiente de los aspectos sin una gran carga de tiempo de ejecución.
- El **tejido dinámico** es el enfoque **más flexible**, pero puede incurrir en significativas **penalizaciones de rendimiento durante la ejecución del programa**.
- Actualmente el **preprocesamiento de código fuente** se usa poco.

Tejedor

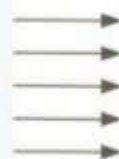


Como se ve, el concepto del **tejedor** se vuelve fundamental en la programación orientada a aspectos. Además que la implementación deja de ser una implementación secuencial y pasa a ser una implementación modularizada.

Tejedor

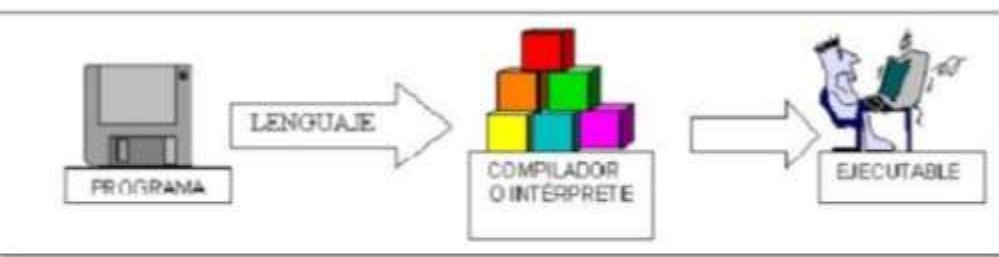
En la anterior imagen se puede observar gráficamente como funciona el tejedor. El tejedor tendría las siguientes repercusiones dentro del código final:

```
aspect Logger {  
    pointcut method() : execution(* *(...));  
    before() : method() {  
        System.out.println("Entering " +  
            thisJoinPoint.getSignature().toString());  
    }  
    after() : method() {  
        System.out.println("Leaving " +  
            thisJoinPoint.getSignature().toString());  
    }  
}  
  
public class Foo {  
    public void bar() {  
        System.out.println("Executing Foo.bar()");  
    }  
    public void baz() {  
        System.out.println("Executing Foo.baz()");  
    }  
}
```

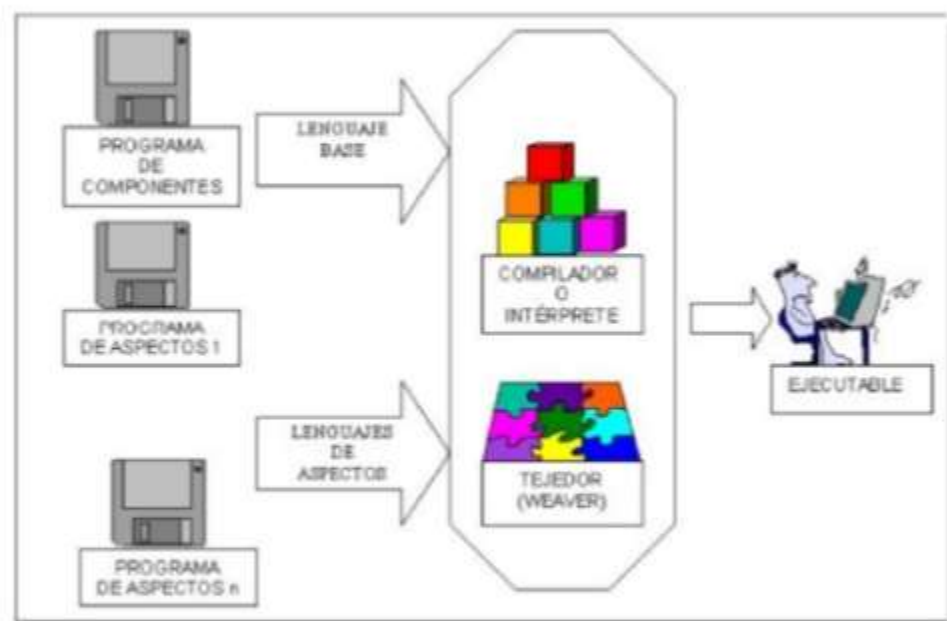


```
public class Foo {  
    public void bar() {  
        System.out.println("Entering Foo.bar()");  
        System.out.println("Executing Foo.bar()");  
        System.out.println("Leaving Foo.bar()");  
    }  
    public void baz() {  
        System.out.println("Entering Foo.baz()");  
        System.out.println("Executing Foo.baz()");  
        System.out.println("Leaving Foo.baz()");  
    }  
}
```

Diferencia en la estructura de la implementación



Estructura de una implementación en los lenguajes tradicionales.



Estructura de una implementación en los lenguajes de aspectos.

Librerías

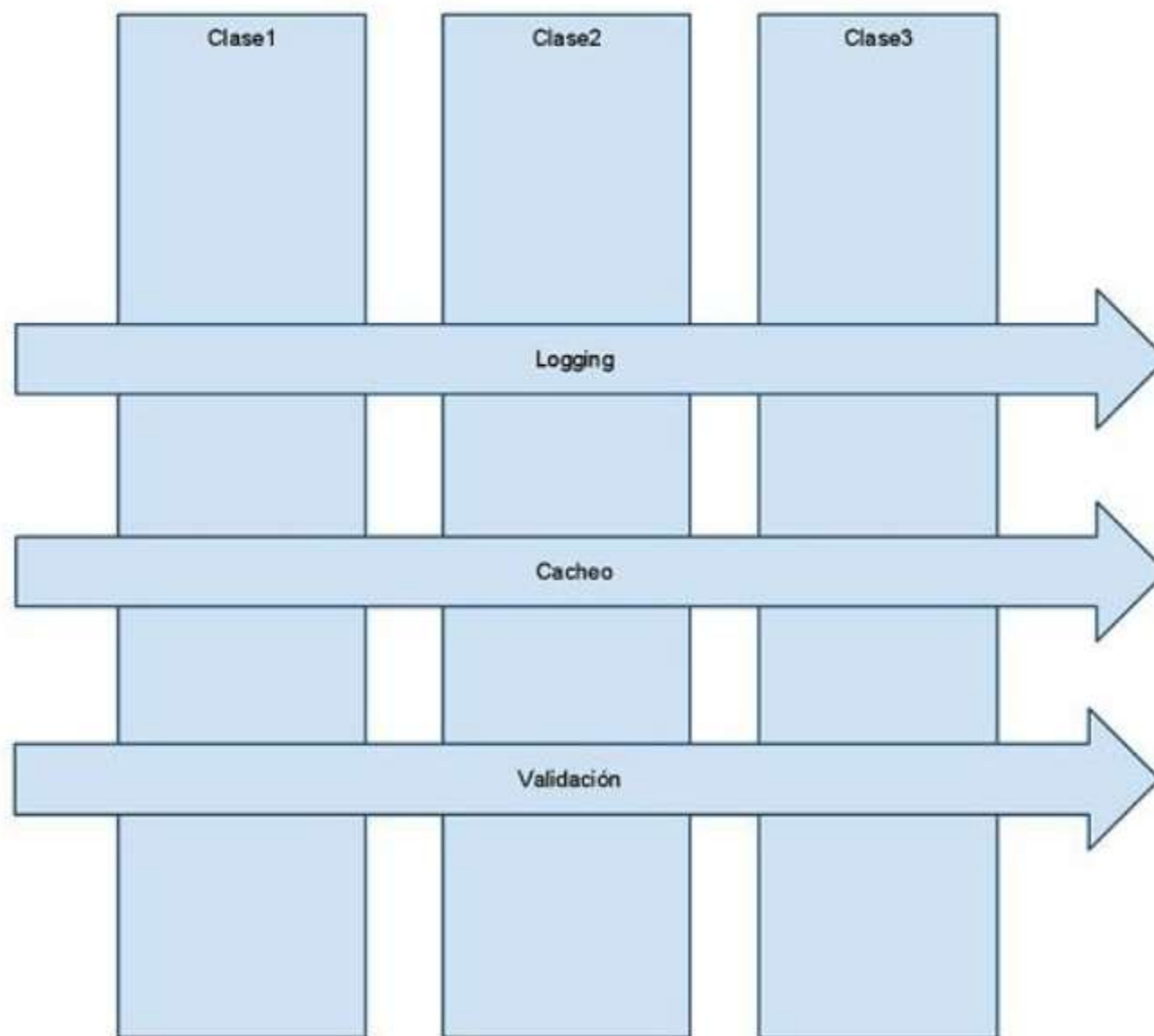
LOA	Lenguaje Base	Tejido	Propósito	Características Salientes
Cool	Java	Estático	Específico	Describe la sincronización de hilos concurrentes. Visibilidad limitada del aspecto
RIDL	Cualquier lenguaje orientado a objetos	Estático	Específico	Modulariza la interacción remota. Visibilidad limitada del aspecto
MALAJ	Java	Dinámico	Específico	Modulariza los aspectos de sincronización y relocación Su objetivo es eliminar los conflictos entre POA y POO.
AspectJ	Java	Dinámico - Estático	General	Los aspectos son extensiones del concepto de clase.
AspectC	C	Estático	General	Usado en la implementación orientada a aspectos de sistemas operativos
AspectC++	C++	Estático	General	Los aspectos son extensiones del concepto de clase.

Las "6 C's" de Mehmet Aksit

Conjunto de conceptos propuestos por Mehmet Aksit y son las bases a la hora de generar software con programación orientada a aspectos.

1. **Entrecruzado** (**C**rosscutting): encapsula los conceptos transversales que existen en el código tradicional, para luego entrecruzarlo con la funcionalidad principal que por ende también queda encapsulada.
 - Éste entrecruzado hace que ambos, funcionalidad y aspectos queden totalmente funcionales sin necesidad de repetición de código.

Ejemplo de entrecruzado



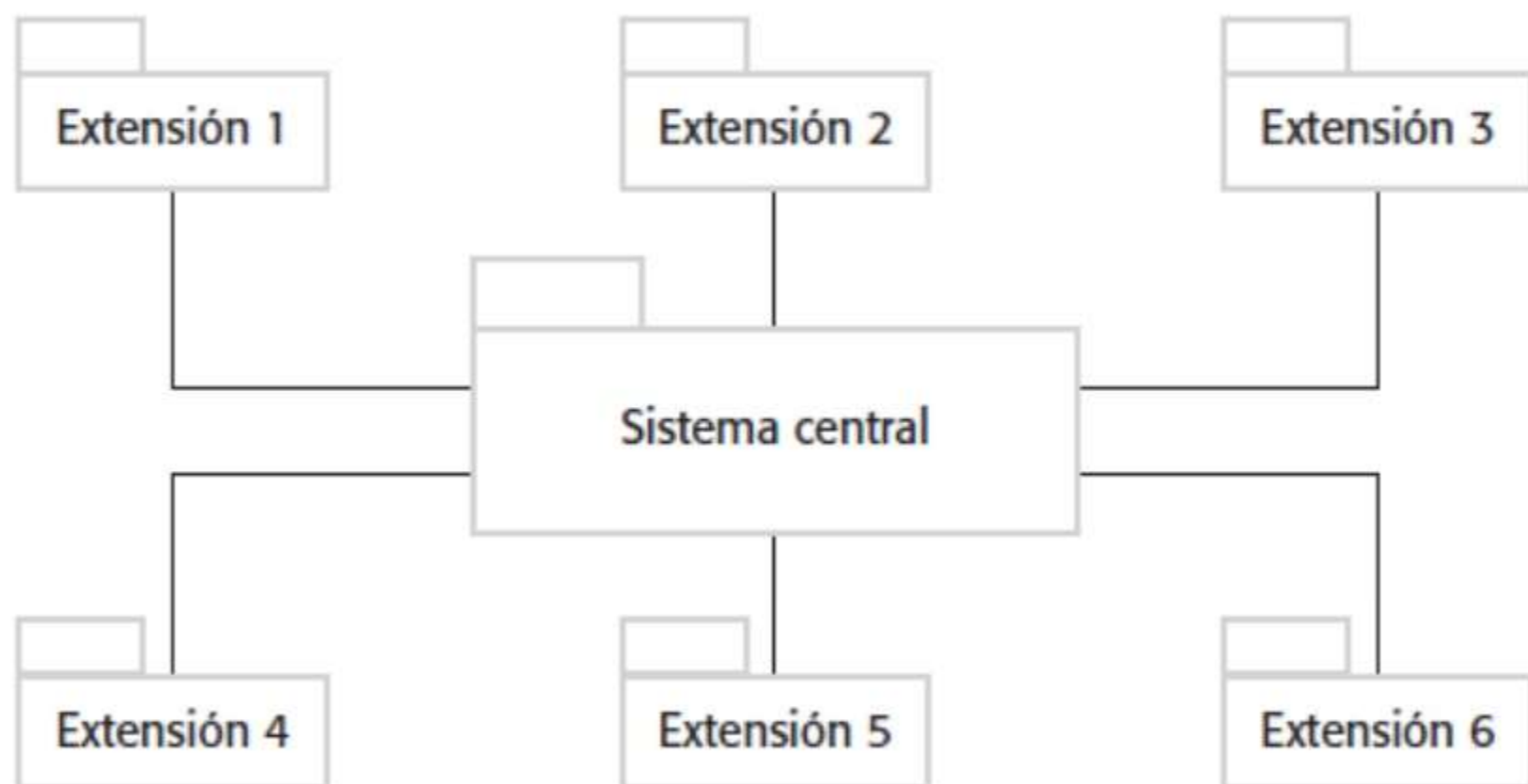
Las "6 C's" de Mehmet Aksit

2. **Canónico:** Los conceptos transversales deben ser implementados de manera estable y completa, totalmente independientes entre ellos.
3. **Composición:** La implementación del modelo debe proveer factores de calidad, como adaptabilidad, reusabilidad y extensibilidad.
4. **Clausura:** La implementación debe mantener la totalidad de los factores de calidad del diseño en la etapa de implementación, para evitar problemas con los conceptos de funcionalidad y configuración.
5. **Computabilidad:** El software generado por la implementación orientada a aspectos debe ser funcional y ejecutable.
6. **Certificabilidad:** Los modelos de diseño e implementación deben ser evaluables en cada parte del proceso de desarrollo e implementación, además de ser controlable para aumentar o mantener la calidad del modelo.

Las "4 S's" de Harold Ossher

- Se refieren a separación exitosa de conceptos por Harold Ossher y son las siguientes:
 1. **Simultáneo**: Los conceptos y composiciones importantes deben poder coexistir dentro del modelo completo sin interrumpirse.
 2. **Auto-contenido (Self-contained)**: Cada módulo debe declarar sus dependencias, con el objetivo de poder entender cada módulo por separado.
 3. **Simétrico**: Todos los conceptos deben ser encapsulados de la misma manera en cada módulo diferente, todo esto con el fin de obtener una mayor confiabilidad en la composición.
 4. **Espontáneo (Spontaneous)**: Al agregar nuevos conceptos, la encapsulación, integración e identificación debe ser sencilla, incluyendo los conceptos que surjan en la etapa de implementación.

Sistema central con extensiones



Puntos de vista y competencias



Ejemplo DOA: Aplicación web para cadena de cines

- Una cadena de cines solicita un sistema que proporcione el uso masivo de funcionalidades para el acceso público.
- Debe proporcionar también la facilidad para promover los diversos cines de la cadena.
- Principalmente se deben considerar en el sistema:
 - reservación de boleto,
 - pago, consulta y cancelación en línea,
 - registro del usuario,
 - transacciones estadísticas.

Ejemplo DOA: Aplicación web para cadena de cines

http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S0798-40652010000300007

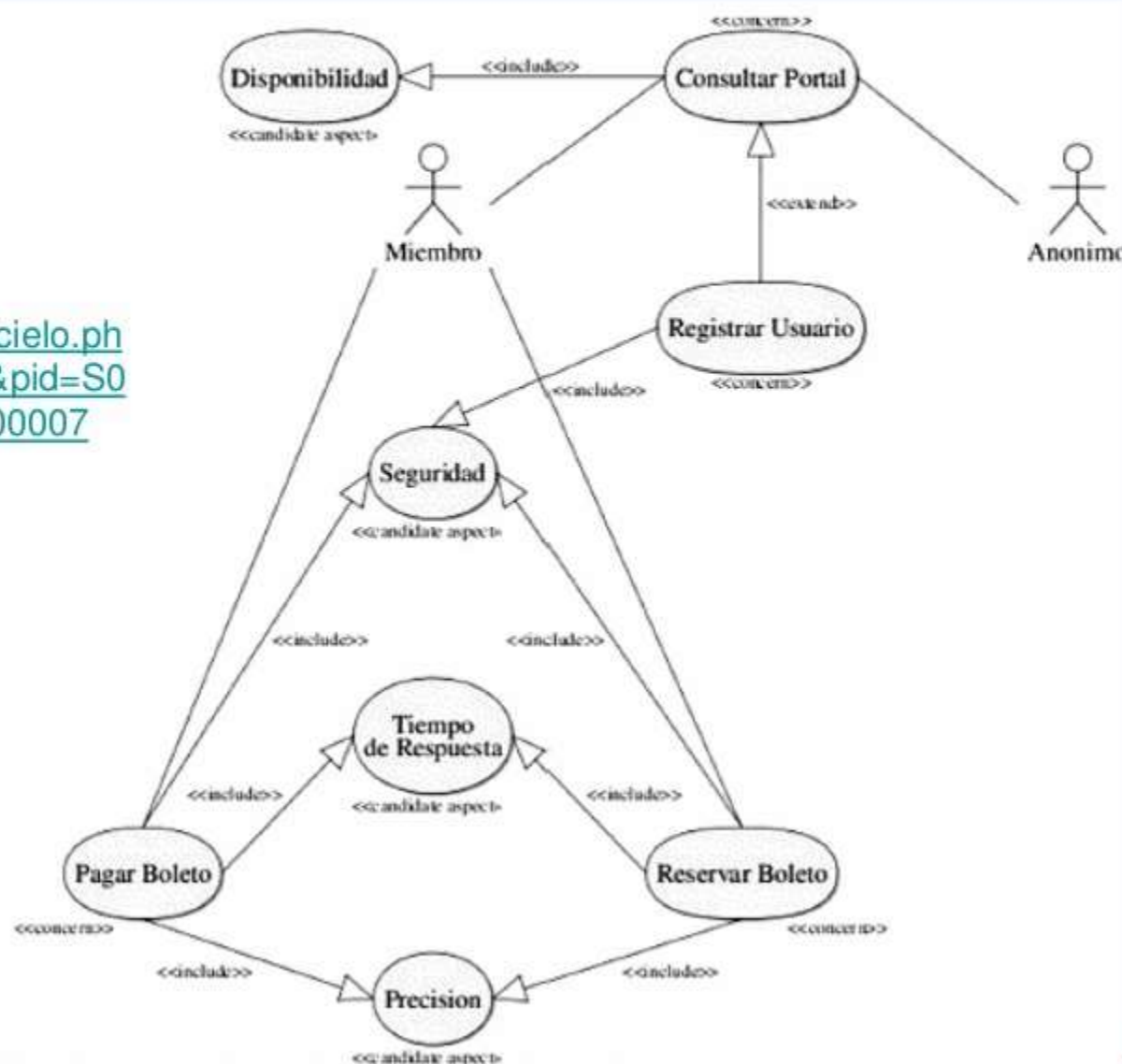


Figura 2. Diagrama de caso de uso para el portal cadena de cines.

Ejemplo DOA: Aplicación web para cadena de cines

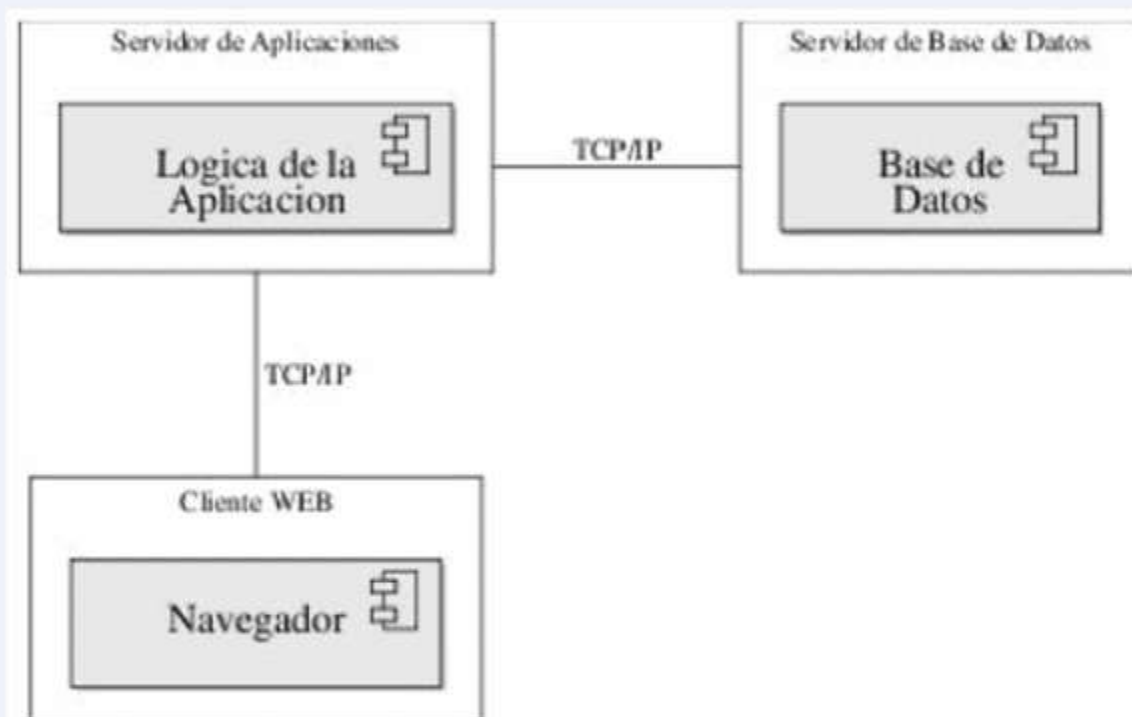


Figura 3. Arquitectura inicial de la aplicación portal cadena de cines: diagrama de despliegue.

http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S0798-40652010000300007

Ejemplo DOA: Aplicación web para cadena de cines

Tabla 2. Identificación de puntos de composición.

Aspecto candidato	Consultar portal	Registrar usuario	Reservar boleto	Pagar boleto
Tiempo de respuesta			X	X
Precisión			X	X
Seguridad		X	X	X
Disponibilidad	X			

http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S0798-40652010000300007

Ejemplo DOA: Aplicación web para cadena de cines

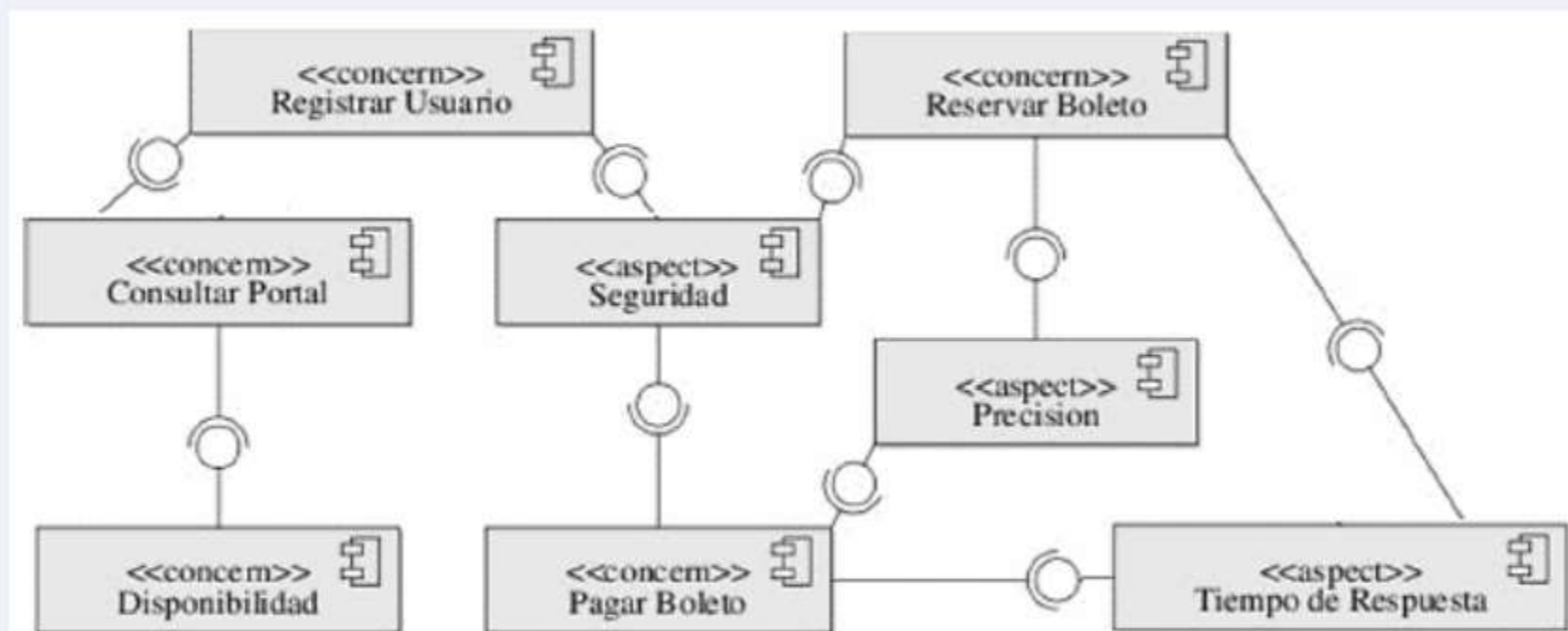


Figura 4. Refinamiento de la arquitectura, componente lógico: diagrama de componentes con incumbencias y aspectos.

http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S0798-40652010000300007

Ejemplo DOA: Aplicación web para cadena de cines

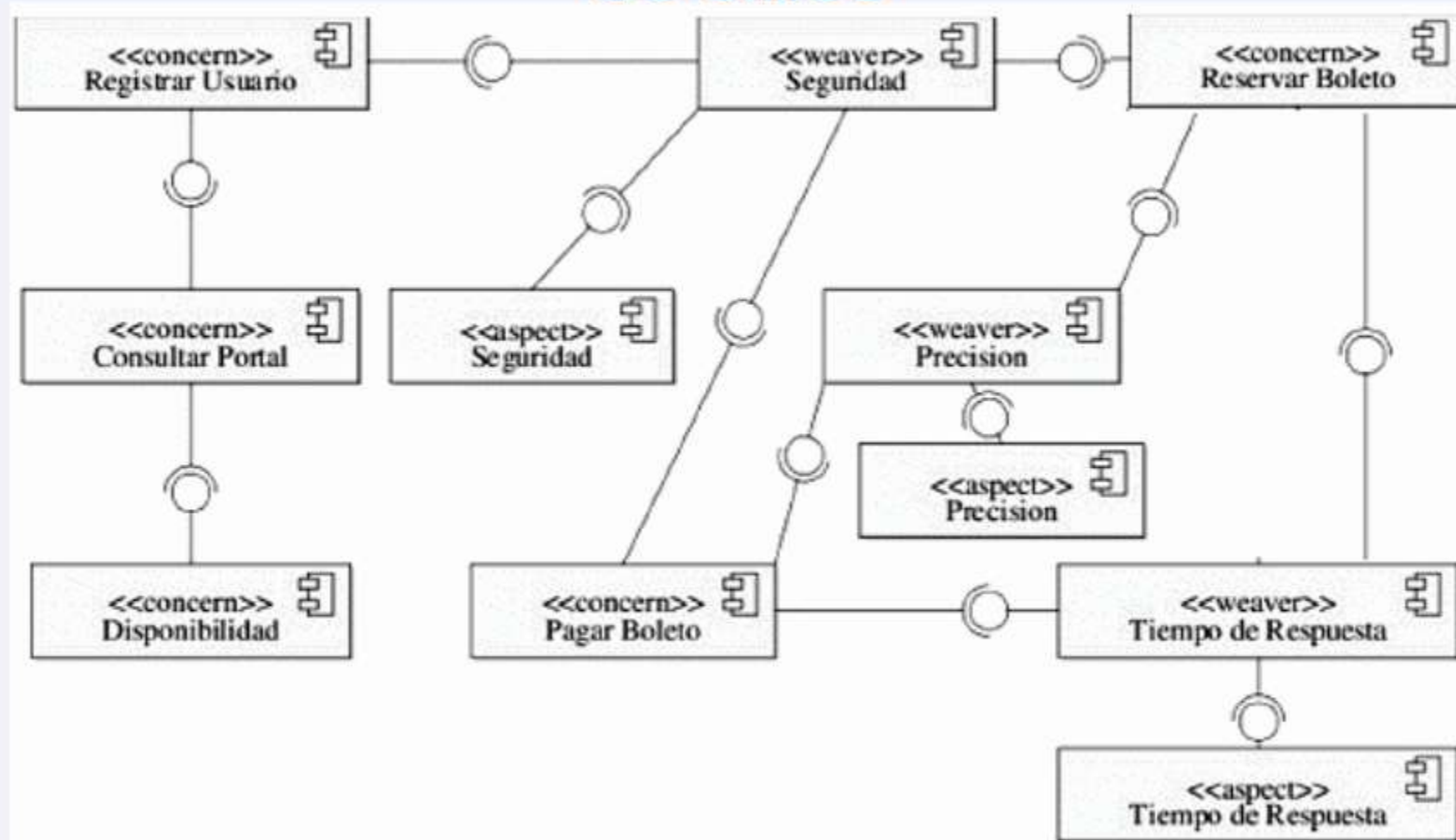
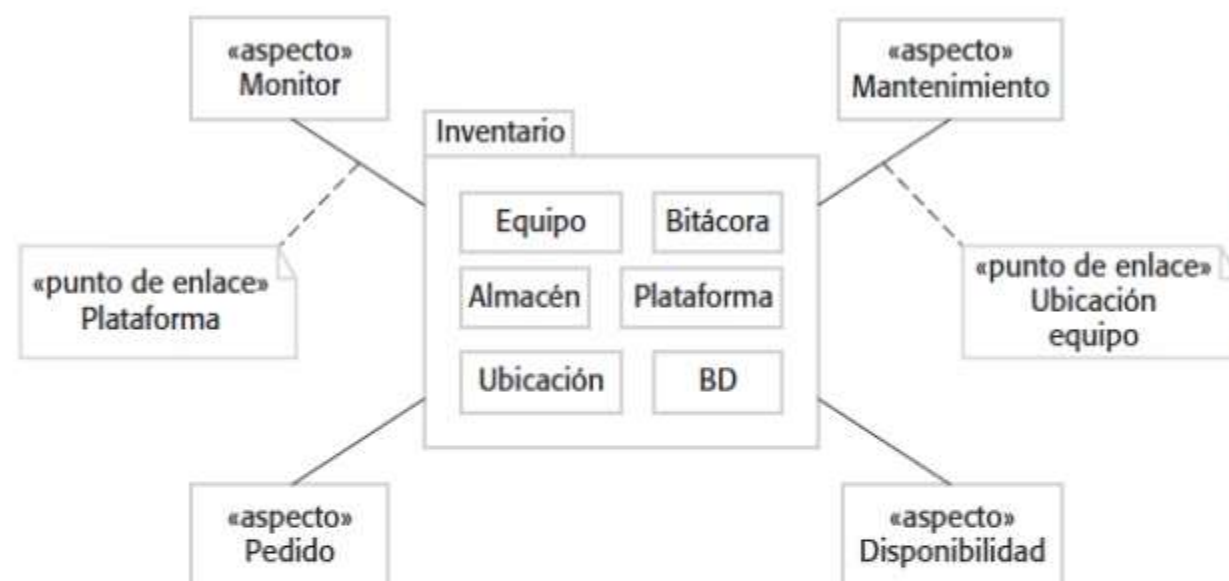


Figura 5. Refinamiento de la arquitectura, componente lógico: diagrama de tejido de componentes.

http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S0798-40652010000300007

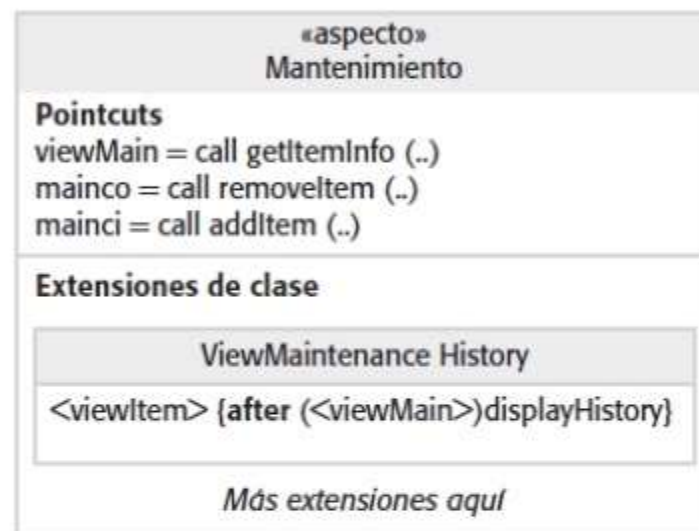
Modelo de diseño orientado a aspectos



La figura presenta un *sistema central para un inventario de servicios de emergencia* más algunos *aspectos* que pueden combinarse con dicho núcleo.

- En la figura se exponen algunas clases del sistema central y algunos aspectos.
- Ésta es una imagen simplificada; un modelo completo incluiría más clases y aspectos.
- En la figura se usan las *notas UML* para ofrecer información adicional acerca de las clases que son atravesadas (*cross-cut*) por algunos aspectos.

Parte de un modelo de un aspecto



- La figura es un modelo más detallado de un aspecto. Desde luego, *antes de diseñar aspectos, hay que tener un diseño del sistema central.*
- La primera sección del aspecto establece los puntos de corte que especifican dónde se combinarán con el sistema central.

–Por ejemplo, el primer punto de corte especifica que el aspecto puede combinarse en el punto de enlace call getItemInfo(..).

–La siguiente sección define las extensiones que implementa el aspecto. En este ejemplo, el enunciado de la extensión puede leerse como:

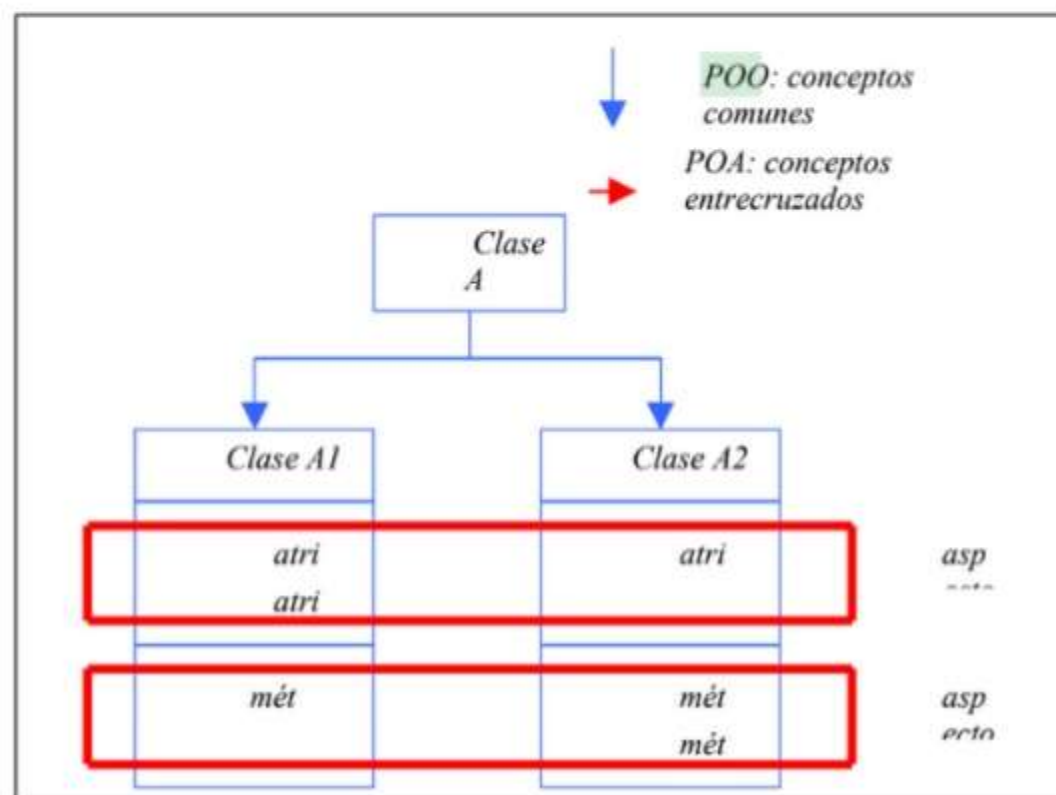
- *En el método viewItem, después de llamar al método getItemInfo, debe incluirse un llamado al método displayHistory para mostrar el registro de mantenimiento.*

POA y POO

- A primera vista daría la impresión que la POA y la POO son en realidad el mismo paradigma.
- Sin embargo, esta primera impresión es errónea. Un análisis más profundo revela las diferencias entre los paradigmas:
 - Tanto la POA como la POO crean implementaciones modularizadas y con mínimo acoplamiento.
 - La diferencia radica en que mientras *la POA se enfoca en los conceptos que se entrecruzan*, la *POO se enfoca en los conceptos comunes y los ubica en un árbol de herencia*.

POA y POO

- En POA la implementación de los conceptos son independientes. Esta independencia la distingue de las técnicas inherentes a la POO.
 - En POA, el flujo de composición va desde los conceptos que se entrecruzan al concepto principal.
 - En POO el flujo va en dirección opuesta.



POA y POO

- La POA **no** está limitada a utilizar como base a la POO.
 - POA puede utilizar como metodología base cualquier paradigma de programación, manteniendo intactos los beneficios de ese paradigma de programación.
 - Se elige POO como el sistema base para obtener los beneficios de una mejor implementación de los conceptos comunes.
 - Bajo esta implementación los conceptos individuales pueden emplear técnicas orientadas a objetos.
 - Esto es análogo a la forma en que actúan los lenguajes procedurales como lenguaje base a muchos lenguajes orientados a objetos.

DOA: problemas con las pruebas (1/3)

- Las pruebas de caja blanca o pruebas estructurales son un enfoque sistemático a las pruebas donde se usa el conocimiento del código fuente del programa para diseñar pruebas de defecto.
- La meta es diseñar pruebas que proporcionen algún nivel de cobertura de programa.
 - Esto es, el conjunto de pruebas debe garantizar que se ejecute toda ruta lógica a través del programa, con la consecuencia de que cada enunciado de programa se efectúe al menos una vez.
- Pueden usarse analizadores de ejecución de programa para demostrar que se logra este nivel de cobertura de pruebas.

INTRODUCCIÓN A LA INGENIERÍA DE SOFTWARE

Se considera dado y conocido

DOA: problemas con las pruebas (2/3)

- Para diseñar pruebas en un programa estructurado (por ejemplo, pruebas del código de un método) sin ramificaciones incondicionales, **se tiene que derivar un gráfico de flujo de programa**, que revele toda la ruta de ejecución lógica a través de dicho programa.
 - Entonces se examina el código y, para cada ruta a través del gráfico de flujo, se eligen valores de entrada que harán que la ruta se ejecute.
- Sin embargo, un **programa orientado a aspectos no es un programa estructurado**. El flujo de control se interrumpe mediante enunciados “viene de” (Constantinos *et al.*, 2004).
 - En algún punto de enlace en la ejecución del código base, puede ejecutarse un aspecto.
- Así pues, no se puede estar seguro de que en tal situación sea posible construir un diagrama de flujo estructurado.

DOA: problemas con las pruebas (3/3)

- Los problemas ocurren porque los aspectos están estrechamente integrados con el código base de un sistema.
 - Por lo tanto, son difíciles de probar en aislamiento.
- Puesto que están tejidos en un programa en muchos lugares diferentes, no podemos estar seguros de que un aspecto que funciona con éxito en un punto de enlace funcionará necesariamente en todos los puntos de enlace.
- Todos éstos siguen siendo problemas de investigación para el desarrollo de software orientado a aspectos

DOA en las empresas

Soarian[®]
Clinicals

The screenshot displays the Siemens Soarian Clinicals interface. It features a top navigation bar with tabs for Patient Record, Clinical Summary, QIS, Charting, Plan of Care, and Orders. The main content area is divided into several sections:

- Problems:** A table listing active problems for a patient.

Problem	Date	Status	Field	Assigned Date
Infection Risk	03/03/2013	Active		03/03/2013
Strength Impairment	03/04/2013	Active		03/04/2013
Respiratory Impairment	03/04/2013	Active		03/04/2013
ROM Impairment	03/04/2013	Active		03/04/2013
- Expected Outcomes:** A table listing expected outcomes for a patient.

Expected Outcome	Target Completion	Charting Status	Note	Last Charted
PTB (T1) - Patient's ADL/ROM will increase to the right knee with 5-15 degrees, active	Not	Not		03/03/13 09:09
PTB (T1) - Patient's motor control will increase to the right knee with 5-15 degrees, active	Not	Not		03/03/13 09:09
PTB (T1) - Patient's pain level will decrease to 0-2 pain score to improve functional mobility	Not	Not		03/03/13 09:09
Discharge of Pain	Discharge	Progressing		03/03/13 09:09
Control of Acute Pain to Patient Satisfaction	Discharge	Not Met		03/03/13 09:09
PTB (T1) - To complete ADL's and functional mobility, patient will improve standing balance to good	Discharge	Progressing		03/03/13 09:09
- Orders:** A section on the right side of the interface showing various orders, including Laboratory, Medication, and Nursing orders.

- **Siemens Soarian** es un sistema de gestión de información de salud que admite un acceso transparente a los registros médicos de los pacientes y la definición de flujos de trabajo para las organizaciones de proveedores de salud.
- Soarian utiliza **AspectJ** para integrar funciones transversales como el seguimiento, la auditoría y la supervisión del rendimiento en el contexto de un proceso de desarrollo ágil.

DOA en las empresas



- **Motorola wi4** es un sistema de infraestructura celular que brinda soporte para el estándar de banda ancha inalámbrica WiMAX.
- El software de control wi4 se desarrolla utilizando una extensión orientada a aspectos del estándar UML 2.0 llamada **WEAVR**. WEAVR se utiliza durante el desarrollo con fines de depuración y prueba.

DOA en las empresas

ASML



- **ASML** es un proveedor de sistemas de litografía para la industria de semiconductores.
- ASML utiliza una extensión orientada a aspectos de C llamada **Mirjam** para modularizar las preocupaciones de rastreo y creación de perfiles.

Unidad 1: Definición y diseño de software en su ciclo de vida de desarrollo

- Introducción a la arquitectura y diseño del software.
 - Definición de diseño y Arquitectura.
 - Problemas fundamentales de diseño
 - Interacciones entre el diseño y los requisitos.
 - Atributos de calidad del Diseño.
- Estrategias de Diseño.
 - Diseño orientado a funciones.
 - Diseño orientado a objetos.
 - Diseño orientado a aspectos.
 - Diseño Arquitectónico.

Bibliografía consultada



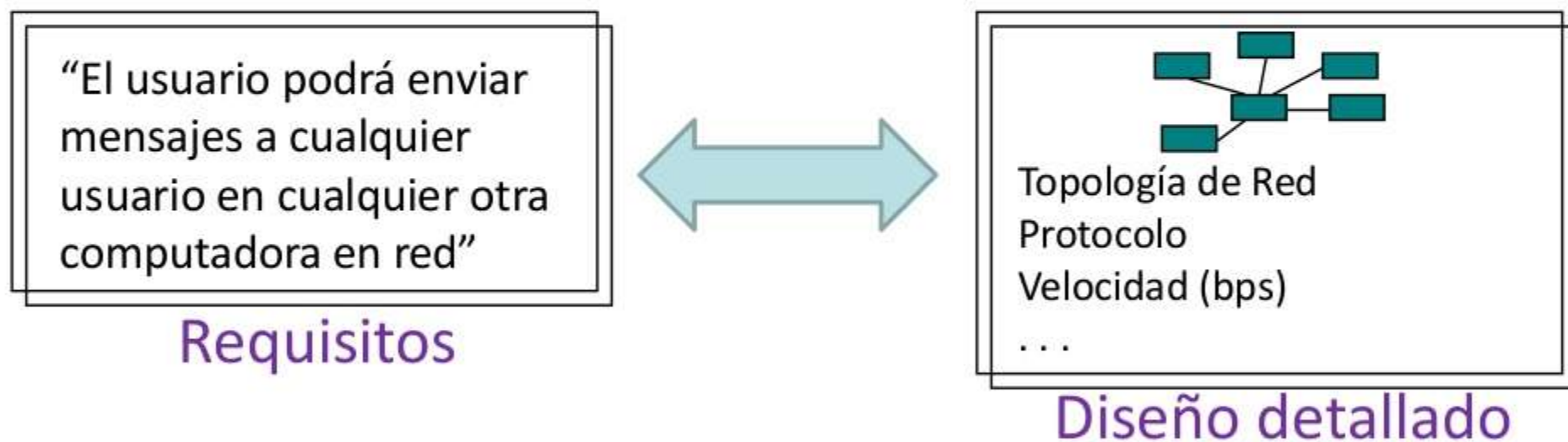
- I. Sommerville, “Ingeniería de software”, 9na Edición, Cap.6
- I. Sommerville, “Ingeniería de software”, 7ma Edición, Cap.11
- Richard N. Taylor, Nenad Medvidovic, Eric Dashofy. “Software Architecture: Foundations, Theory, and Practice”, 1st Edición

Motivación

- Usted raramente empieza desde cero cuando toma decisiones sobre la organización del software, sino que basa su diseño sobre **experiencias de diseño anteriores**.
 - De allí el motivo que usted esté capacitándose en los diversos estilos arquitectónicos a lo largo de este curso ;)
- Usted realmente aprende cómo diseñar al **observar ejemplos** de diseños existentes y discutir su diseño con otros
 - De allí, que el trabajo grupal fomente aquello, la discusión y aprendizaje colaborativo ;)

Diseño arquitectónico

- Constituye la etapa preliminar del diseño.
- Representa el **vínculo** entre los requisitos y el diseño detallado.



- Involucra la **identificación** de los componentes que conforman el sistema y la **forma en que estos se comunicarán**.

Diseño arquitectónico

- El proceso de diseño para identificar los subsistemas que componen un sistema y el marco para el control y la comunicación del subsistema es el diseño arquitectónico.
- El resultado de este proceso de diseño es una descripción de la arquitectura del software.

Niveles de abstracción (Sommerville)

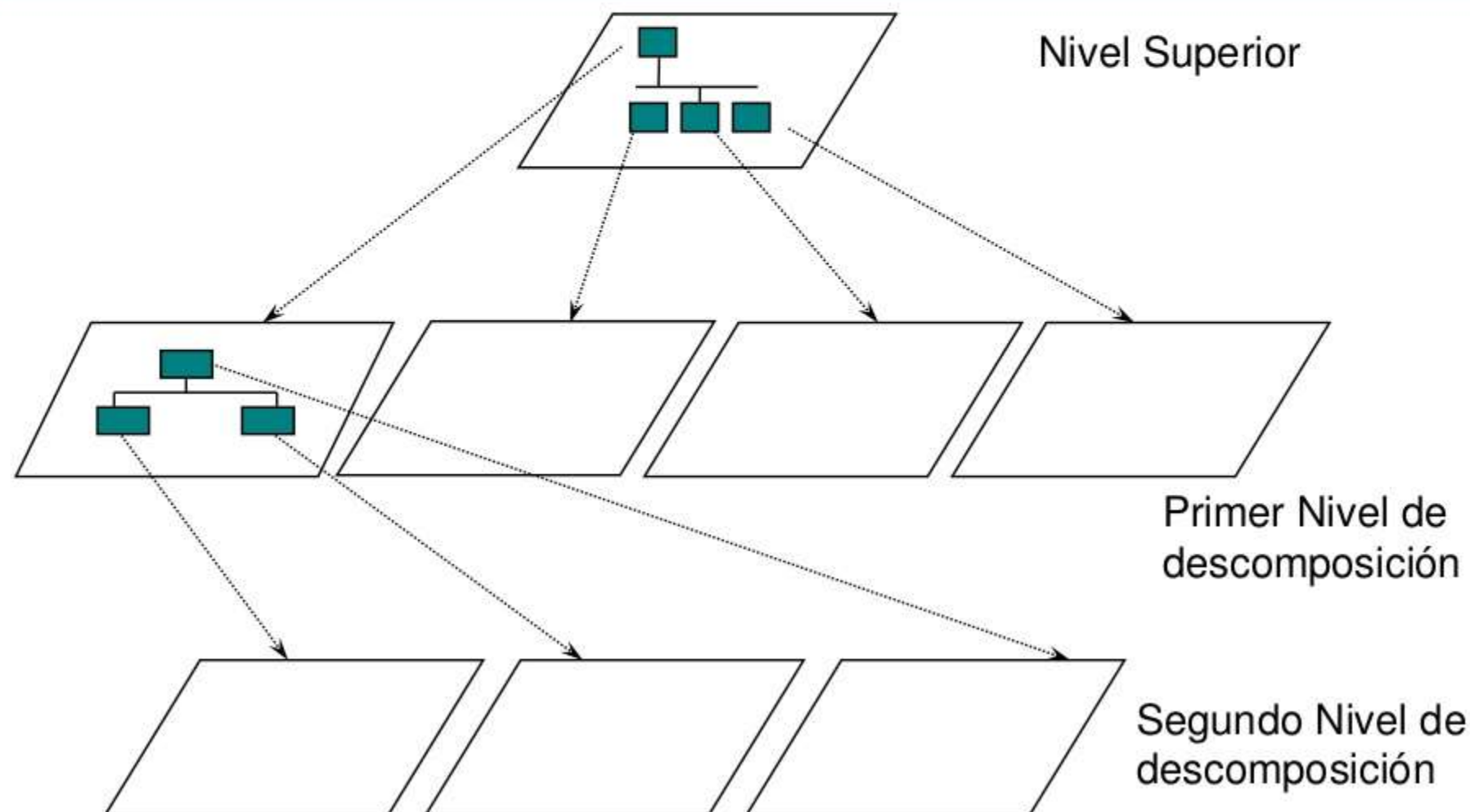
- Las arquitecturas de software se diseñan en dos niveles de abstracción
- **La arquitectura en pequeño** se interesa por la arquitectura de programas individuales.
 - Se centra principalmente en arquitecturas de programa.
 - En este nivel, uno se preocupa por la forma en que el programa individual se separa en componentes.
- **La arquitectura en grande** se interesa por la arquitectura de sistemas empresariales complejos que incluyen otros sistemas, programas y componentes de programa.
 - Tales sistemas empresariales se distribuyen a través de diferentes computadoras, que diferentes compañías administran y poseen.
 - En este curso se cubrirán las arquitecturas grandes cuando abordemos las arquitecturas de los sistemas distribuidos y arquitecturas orientadas a servicios.

Descomposición y Modularidad (1)

- Determinar un conjunto de componentes e interfaces entre ellos, que satisfacen un conjunto especificado de requerimientos (De Marco 1982)
- Métodos de descomposición (Wasserman 1995)
 - **Modular** (a partir de las **funciones**)
 - A partir de los **Datos**
 - A partir de **Eventos** (y transiciones de **Estados**)
 - A partir de las **Entradas** (de afuera hacia adentro)
 - Orientado a **Objetos**
- Sistema Modular: cuando cada una de las actividades la realiza exactamente un único componente donde además están bien definidas c/u de sus entradas y salidas.

Descomposición y Modularidad (2)

Todo método de diseño abarca un tipo de descomposición a partir de una descripción de alto nivel de los elementos claves del sistema, y creando visiones a niveles inferiores para ver como las características y funciones del sistema se acomodaran en conjunto.



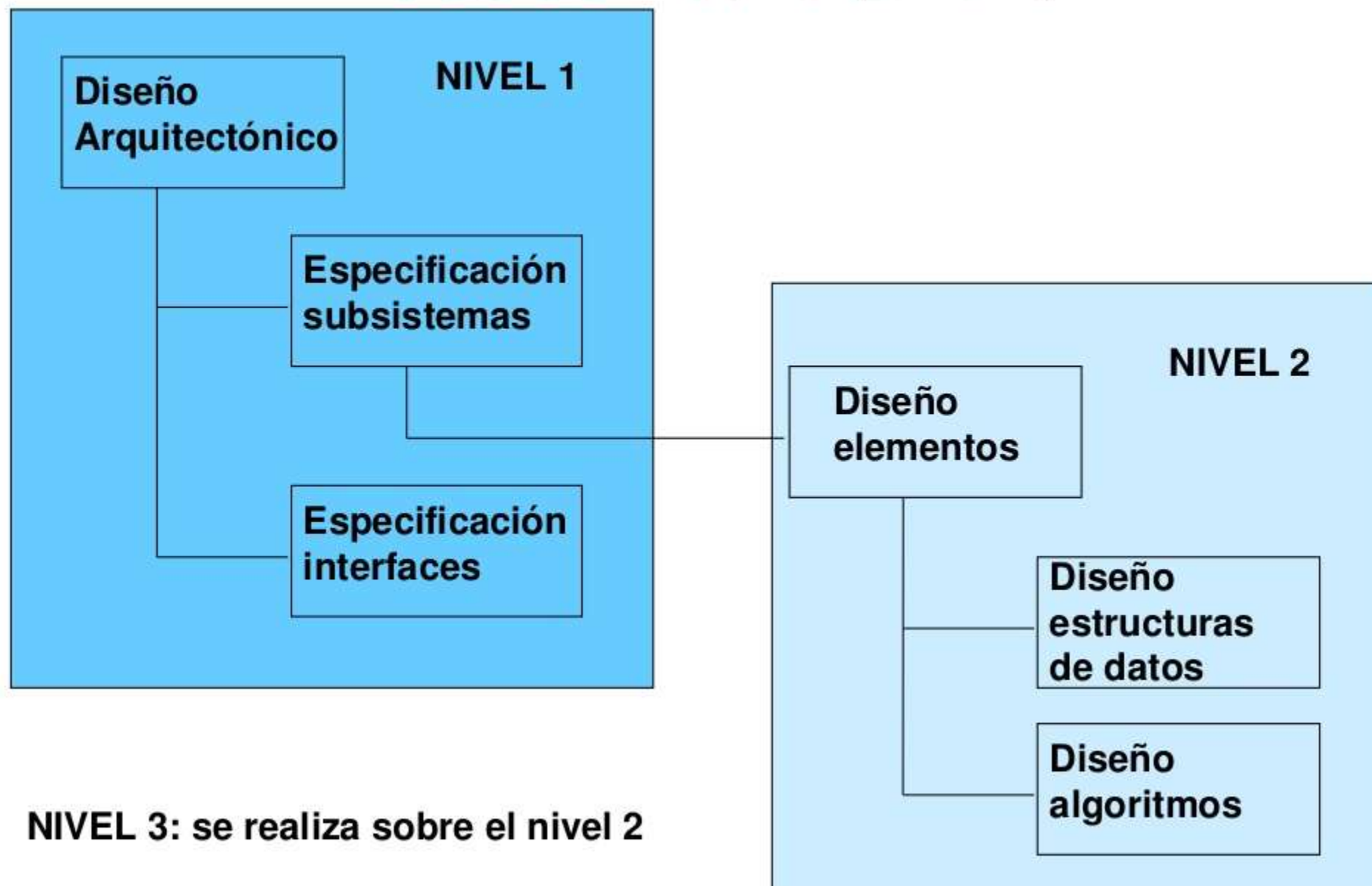
Descomposición y Modularidad (3)

- La descomposición separa el diseño en partes componibles denominadas **módulos o componentes**.
- Se dice que un modulo está “bien definido” cuando todas las entradas en él son esenciales para su función y todas las salidas son generadas por una de sus acciones.

Niveles de Diseño

- Shaw y Garlar (1996) sugieren tres niveles de diseño:
- **Arquitectura**
 - Este nivel está asociado a las capacidades del sistema identificadas en la especificación de requisitos, con los componentes del sistema que la implementarán.
 - Por lo general estos componentes son módulos y la arquitectura también describe las interconexiones entre ellos.
 - La arquitectura define además operadores que crean sistemas a partir de los subsistemas.
- **Diseño de Código**
 - Comprende algoritmos, estructuras de datos, y los componentes que son primitivas del lenguaje de programación, tales como números, caracteres, punteros, etc.
- **Diseño ejecutables**
 - Se remite al diseño de código en un nivel de detalle todavía más bajo.
 - Trata entre otros aspectos la asignación de memoria, los formatos de datos y la configuración de bits.

Niveles de Diseño (Sommerville)



Decisiones estructurales del arquitecto (1/2)



- Durante el proceso de diseño arquitectónico, los arquitectos del sistema deben tomar algunas **decisiones estructurales** que **afectarán profundamente el sistema y su proceso de desarrollo.**
- Con base en su **conocimiento y experiencia**, deben considerar las siguientes preguntas fundamentales sobre el sistema:

Decisiones estructurales del arquitecto (2/2)

- ¿Existe alguna **arquitectura de aplicación genérica** que actúe como plantilla para el sistema que se está diseñando?
- ¿Cómo se **distribuirá el sistema** a través de algunos núcleos o procesadores?
- ¿Qué patrones o **estilos arquitectónicos** pueden usarse?
- ¿Cuál será el **enfoque** fundamental usado para **estructurar el sistema**?
- ¿Cómo los componentes estructurales en el sistema se separarán en **subcomponentes**?
- ¿Qué estrategia se usará para **controlar la operación de los componentes** en el sistema?
- ¿Cuál **organización arquitectónica** es mejor para entregar los **requerimientos no funcionales** del sistema?
- ¿Cómo se **evaluará** el diseño arquitectónico?
- ¿Cómo se **documentará** la arquitectura del sistema?

Evaluación de diseño arquitectónico

- Evaluar un diseño arquitectónico es difícil porque la verdadera prueba de una arquitectura es qué tan bien el sistema cubre sus requerimientos funcionales y no funcionales cuando está en uso.
- Sin embargo, es posible hacer cierta evaluación al comparar el diseño contra arquitecturas de referencia o patrones arquitectónicos genéricos, como las que revisaremos en este curso.

El módulo de aterrizaje lunar: un ejemplo para todo el curso



El módulo de aterrizaje lunar: un ejemplo para todo el curso

- Un simple juego de computadora que apareció por primera vez en la década de 1960.
- Concepto simple:
 - Tú (el piloto) controlas la velocidad de descenso del módulo de aterrizaje lunar de la era Apolo
 - El ajuste del **acelerador** controla el motor de **descenso**
 - **Combustible** limitado
 - Preajuste inicial de **altitud** y **velocidad**
 - Si aterriza con una velocidad de descenso de <5 fps: gana (ya sea que quede combustible o no)
 - Versión "avanzada": el joystick controla la actitud y el movimiento horizontal

Etapas del proceso de diseño arquitectónico

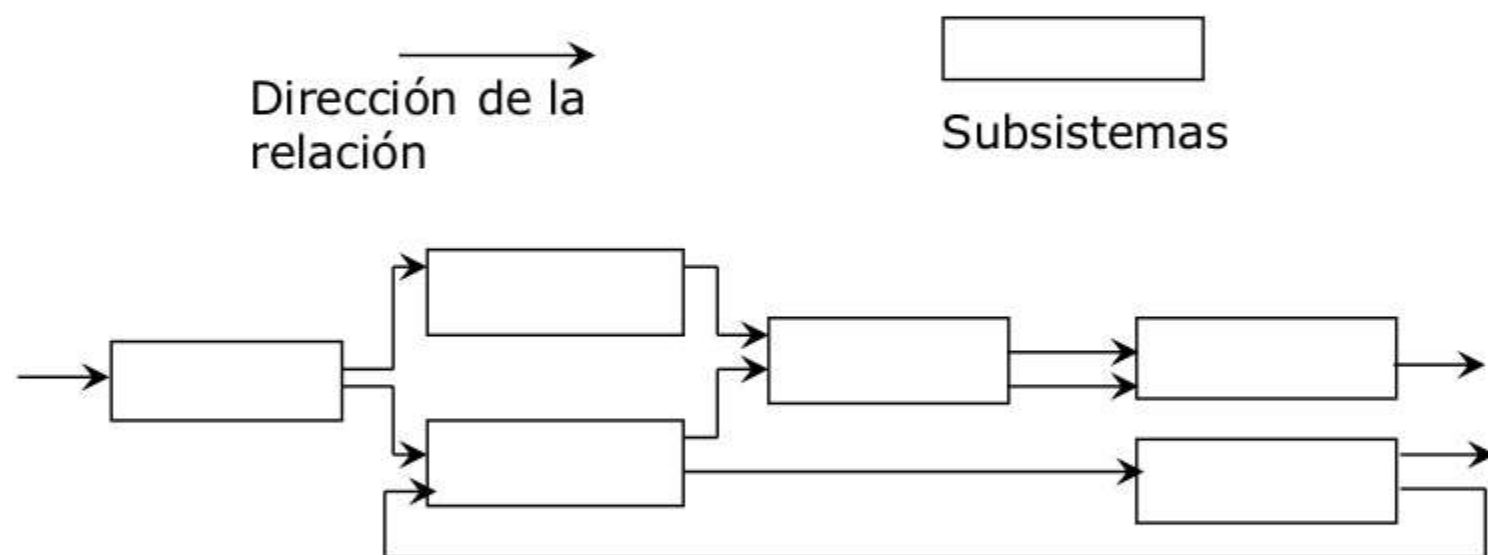
- Las siguientes etapas son comunes para todos los procesos de diseño arquitectónicos:
 - **Estructuración del sistema:** se establecen los subsistemas y sus relaciones.
 - **Modelo de control:** se establece un modelo general de las relaciones de control entre las partes del sistema.
 - **Descomposición modular:** cada subsistema se descompone en módulos.

Estructura del sistema

- Está primera etapa, es la descomposición de un sistema en un conjunto de subsistemas que interactúan.
- En el nivel más abstracto, un diseño se puede ver como un diagrama de bloques en el que cada cuadro representa un subsistema.
- Un diagrama de bloques presenta un panorama general de la estructura del sistema. Por lo general, esto es comprensible por usuarios y desarrolladores.

Estructura del sistema

Ejemplo abstracto para un diagrama de bloques.



Se pueden desarrollar modelos más específicos de la estructura, que muestren cómo los subsistemas comparten datos, cómo están distribuidos y como se conectan entre ellos.

Estructura del sistema

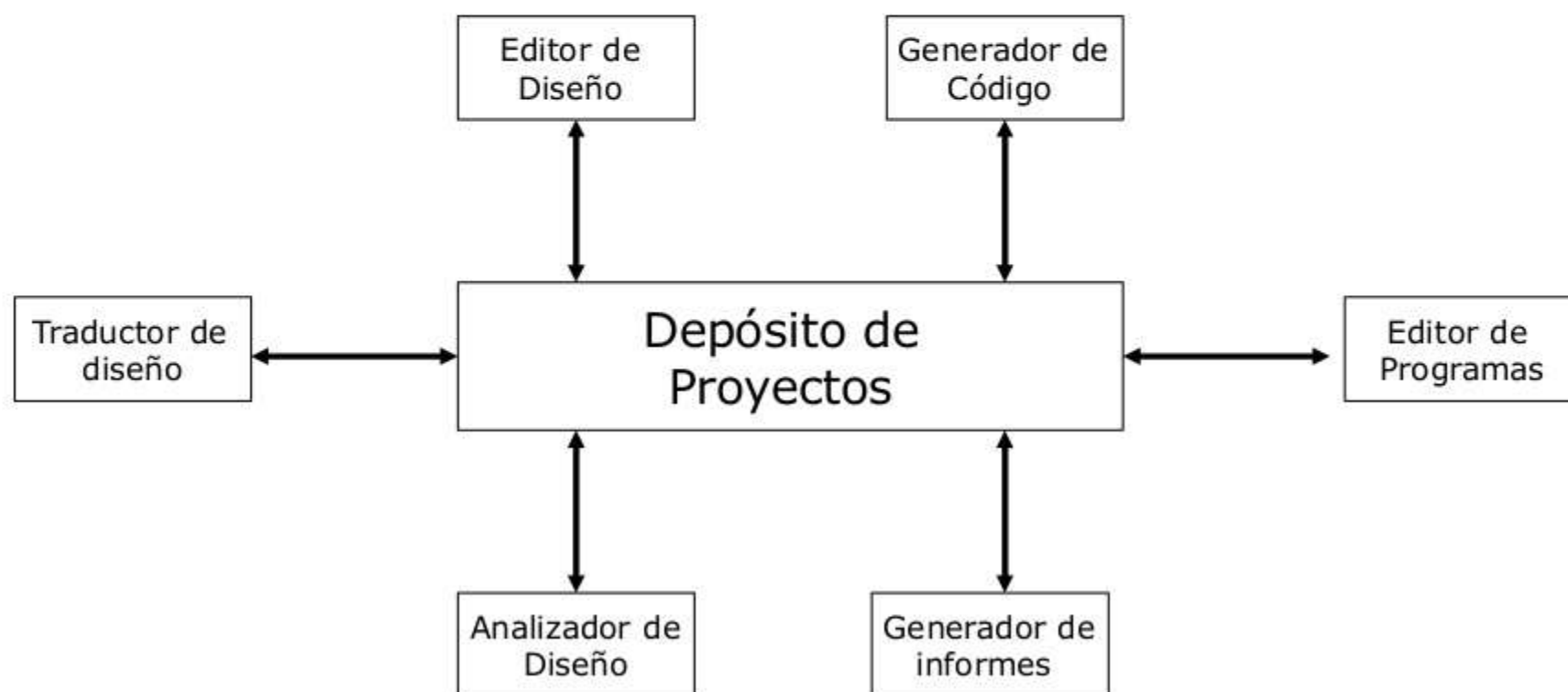
1. Modelo de Depósito (repositorio)

- Los subsistemas que componen un sistema deben intercambiar información con el fin de que puedan trabajar de forma conjunta y efectiva.
- Existen dos formas:
 - Todos los datos compartidos se ubican en una base de datos central que pueda ser accedida por todos los subsistemas (se denomina modelo de deposito).
 - Cada subsistema tiene su propia base de datos. Los datos se intercambian con otros subsistemas pasando mensajes entre ellos.

Estructura del sistema

1. Modelo de Depósito (repositorio)

Ejemplo Modelo Depósito: La arquitectura de un conjunto integrado de herramientas case.

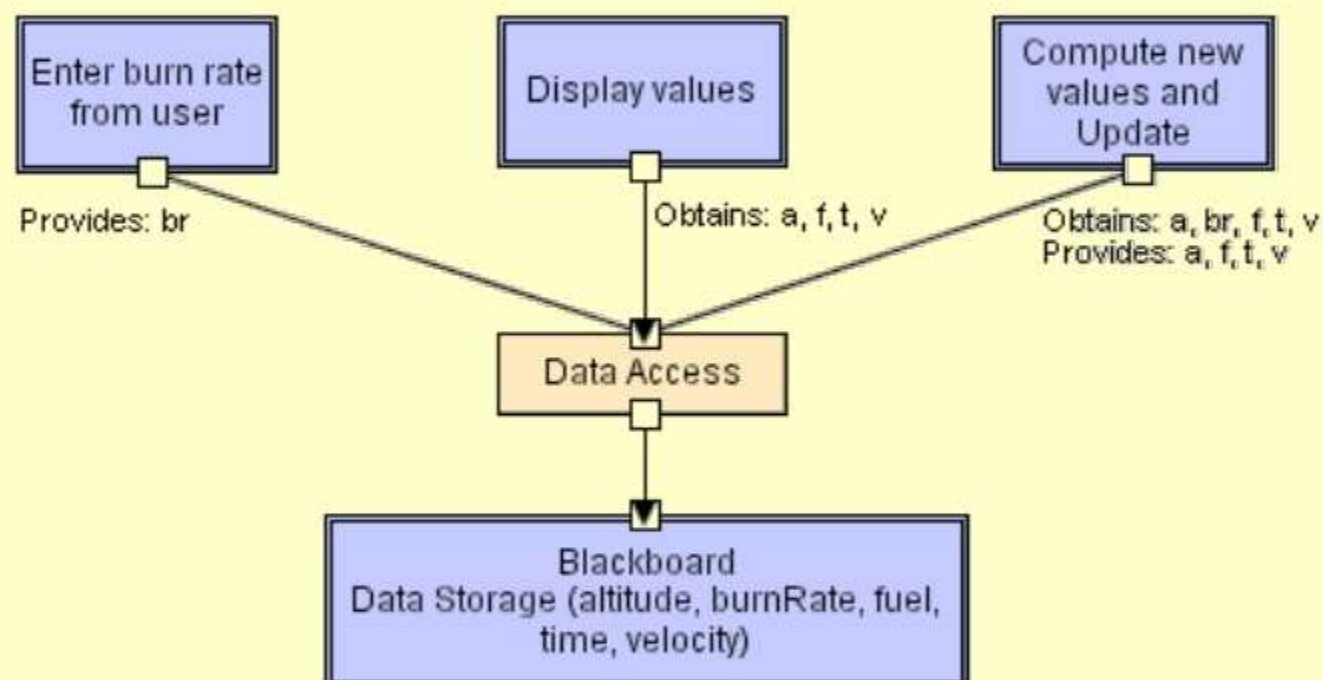


Estructura del sistema

1. Modelo de Pizarrón (Repositorio activo)

- Dos tipos de componentes
 - Estructura de datos central — pizarra
 - Componentes que operan en la pizarra
- El **control** del sistema está totalmente **impulsado por el estado de la pizarra**
- Ejemplos
 - Usado típicamente para sistemas de IA
 - Entornos de software integrados (por ejemplo, Interlisp)
 - Arquitectura del compilador

El módulo de aterrizaje lunar: estilo arquitectónico Pizarrón (Repositorio activo)



Estructura del sistema

1. Modelo de Depósito

- **Ventajas**

- Eficiente para compartir grandes cantidades de datos.
- Los subsistemas están acordes al modelo de depósito de datos.
- Las actividades de respaldo, seguridad, control de acceso y recuperación de errores están centralizadas. Son responsabilidad del deposito.

- **Desventajas**

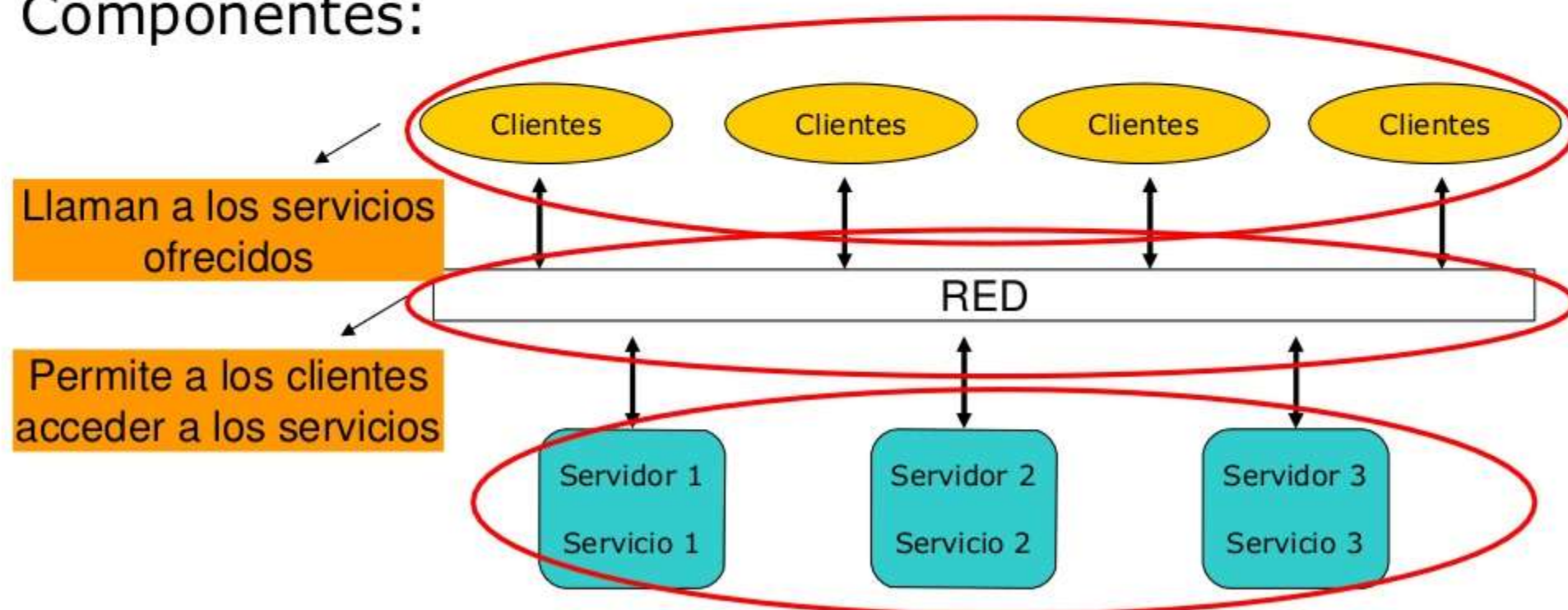
- Si se genera un gran volumen de información, será difícil evolucionar si se ha acordado un modelo de datos. Traducir esto en un nuevo modelo será costoso, puede ser difícil, e incluso imposible.
- El modelo de depósito fuerza a los subsistemas a las mismas políticas de seguridad.

Estructura del sistema

2. Modelo Cliente - Servidor

- Es un modelo de sistemas distribuidos que muestra como los datos y el procesamiento se distribuyen a lo largo de varios procesadores.

Componentes:

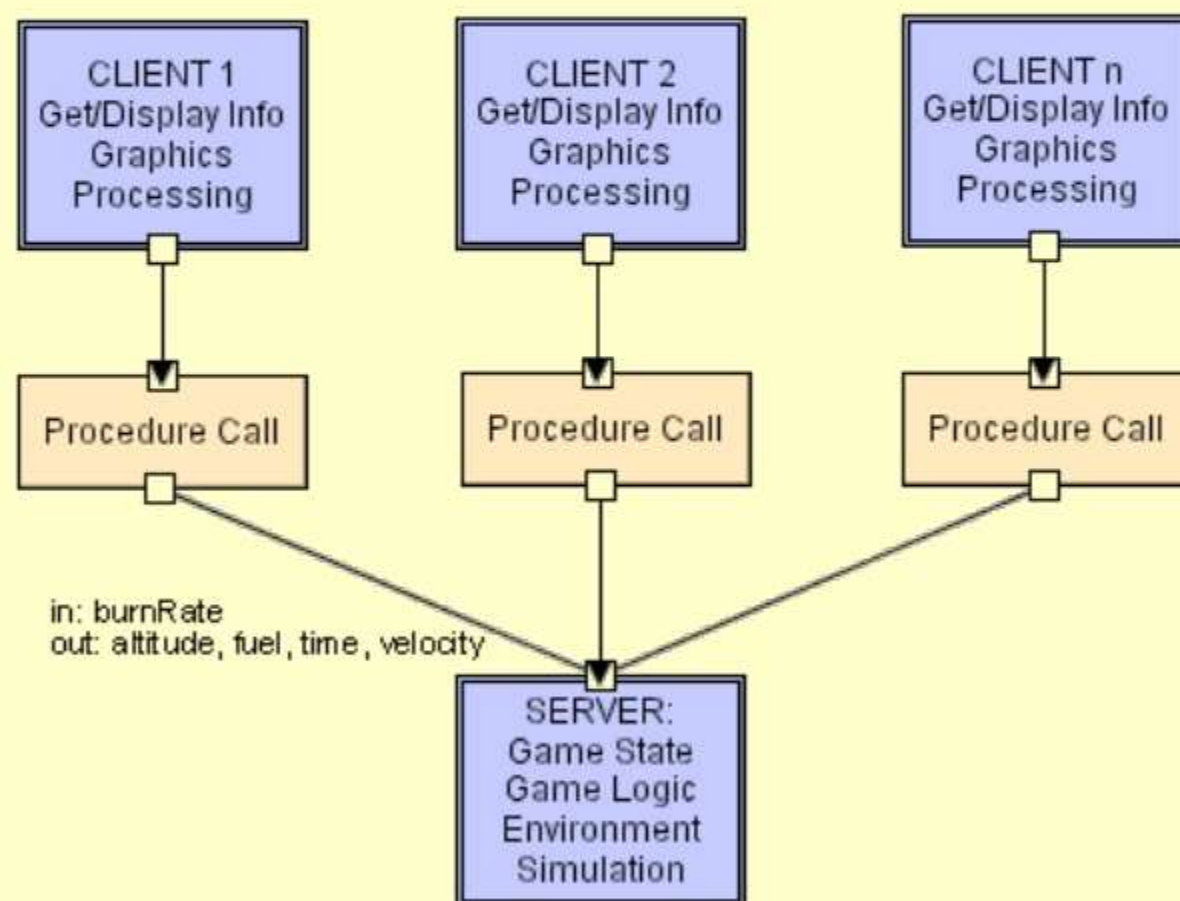


Estructura del sistema

2. Modelo Cliente - Servidor

- Los clientes necesitan conocer los nombres de los servidores y los servicios que suministran. En cambio, los servidores no requieren conocer la identidad de los clientes o cuantos clientes existen.
- La ventaja más importante del modelo cliente – servidor, es su arquitectura distribuida. Ya que, es fácil agregar o modificar un nuevo servidor sin afectar a otras partes del sistema.
- El lado negativo es que por lo general necesitan seguridad más elaborada, administración del sistema y desarrollo de aplicaciones. Así que necesitan de más recursos para ser implementados y darles soporte.

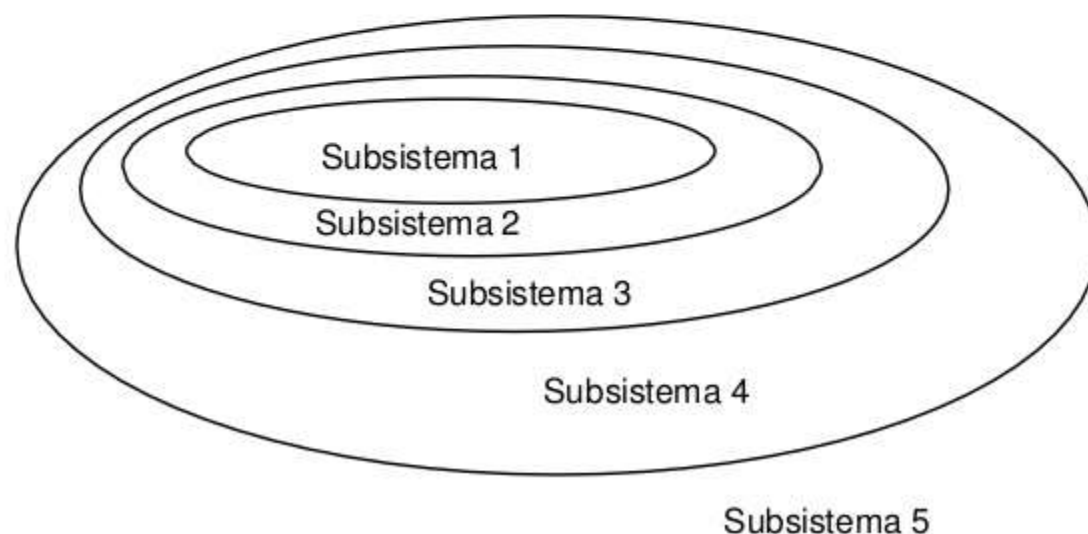
El módulo de aterrizaje lunar: estilo arquitectónico cliente-servidor



Estructura del sistema

3. Modelo de Capas

- Modela la interacción entre los subsistemas.
- Organiza el sistema en una serie de capas, cada una de las cuales suministra un conjunto de servicio.
- Un modelo bien conocido para este enfoque es el modelo de referencia ISO/OSI.

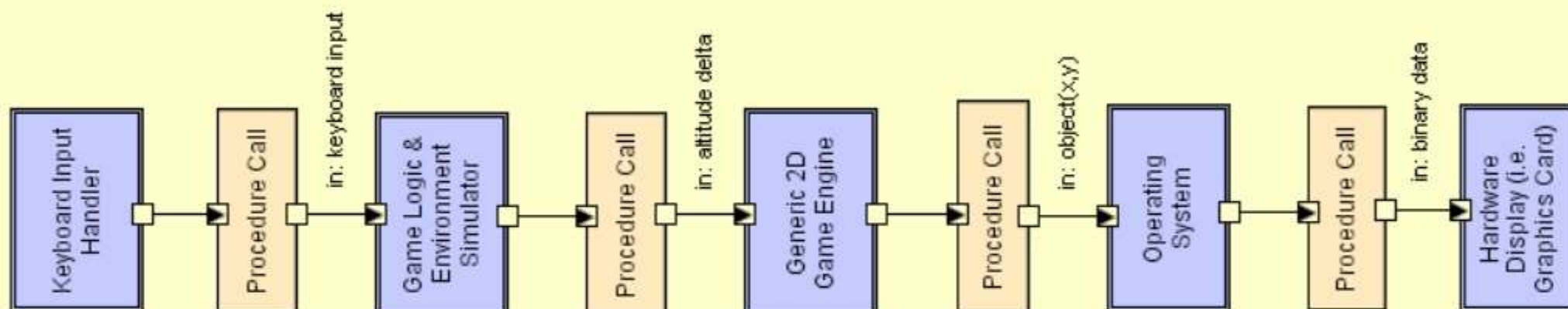


Estructura del sistema

3. Modelo de Capas

- Este modelo es una arquitectura cambiable y portable. Cuando se cambia o elimina una capa, solo se verán afectadas sus capas adyacentes.
- Una desventaja del enfoque, es que estructurar los sistemas de esta forma es difícil.
- El desempeño también puede ser un problema debido a los múltiples niveles de interpretación de ordenes que algunas veces se requieren.

El módulo de aterrizaje lunar: Capas jerárquicas



Etapas del proceso de diseño arquitectónico

- Las siguientes etapas son comunes para todos los procesos de diseño arquitectónicos:
 - **Estructuración del sistema:** se establecen los subsistemas y sus relaciones.
 - **Modelo de control:** se establece un modelo general de las relaciones de control entre las partes del sistema.
 - **Descomposición modular:** cada subsistema se descompone en módulos.

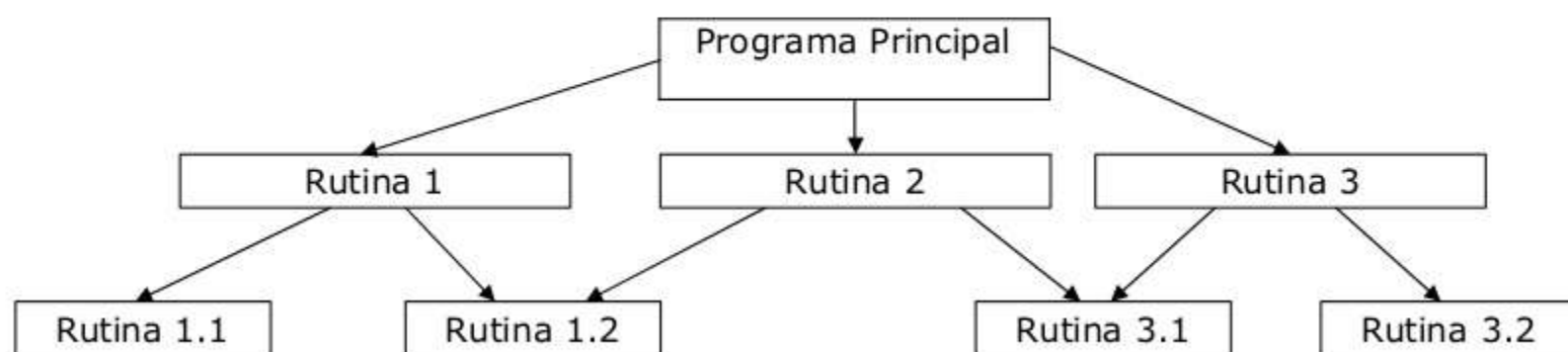
Modelos de Control

- Los **modelos para estructurar un sistema** se refieren a la manera en que un sistema se descompone en subsistemas. Esto **no incluye información de control**.
- El diseñador debe **organizar los subsistemas acorde a un modelo de control que complemente** el modelo estructural que se utiliza.
- Se identifican dos enfoques:
 - **Control Centralizado:** un subsistema tiene completa **responsabilidad** para controlar, iniciar, y detener a otros subsistemas. Puede entregar el control, pero se lo deben devolver.
 - **Control basado en eventos:** cada subsistema puede **responder a eventos** generados en el exterior (en lugar de que la información de control esté embebida en un subsistema).

Modelos de Control

1. Control Centralizado

- Un subsistema se designa como controlador del sistema y tiene la responsabilidad de administrar la ejecución de otros subsistemas.
- Existen dos enfoques:
 - **Modelo de llamada - retorno:** éste es el modelo familiar de subrutinas descendentes, donde el control se inicia en la parte superior de una jerarquía y por medio de llamadas a las subrutinas pasa a los niveles inferiores del árbol.

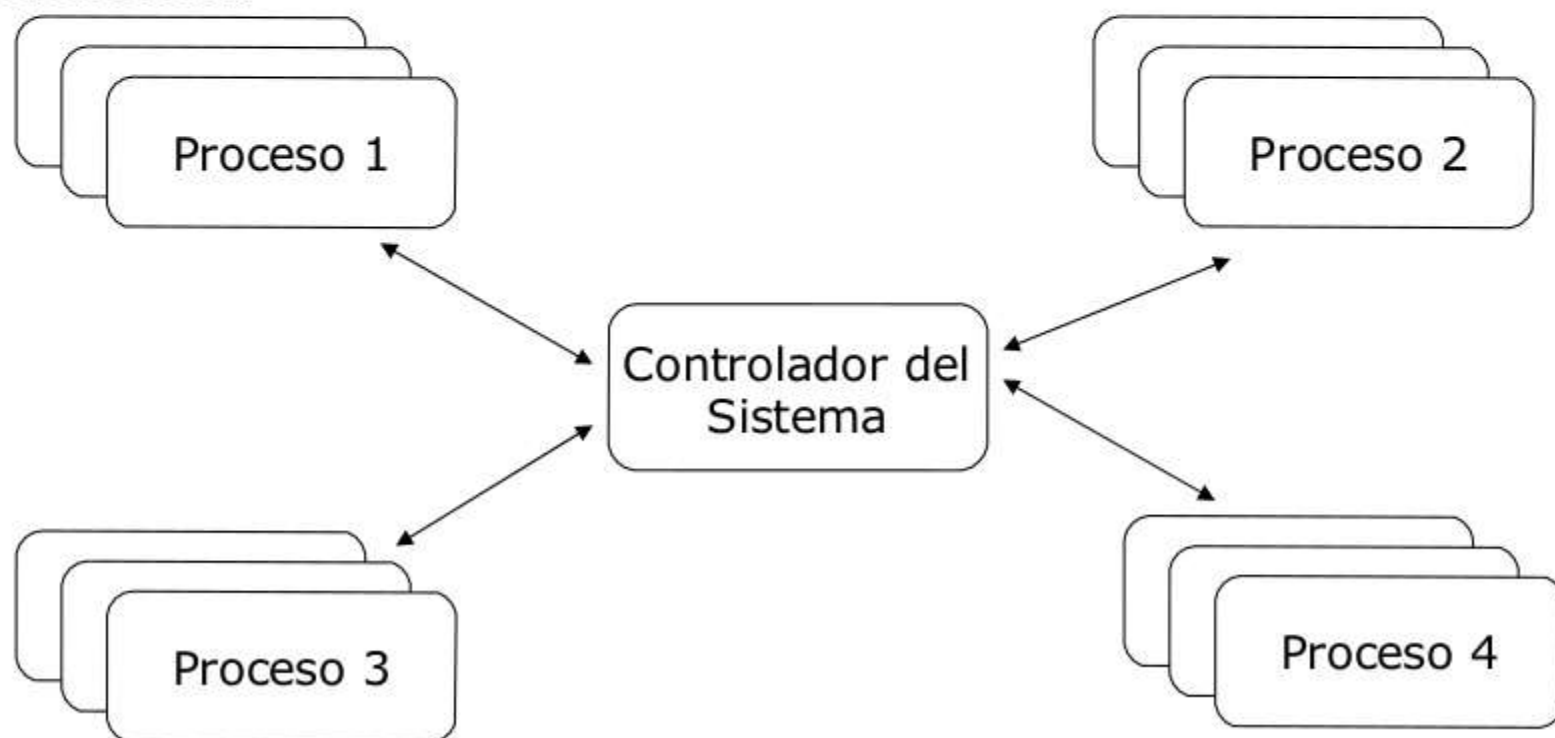


Aplicable solo a sistemas secuenciales

Modelos de Control

1. Control Centralizado

- **Modelo del administrador:** se aplica a los modelos concurrentes. Un componente del sistema se designa como un sistema administrador y controla el inicio, detención y la coordinación de otros procesos del sistema.



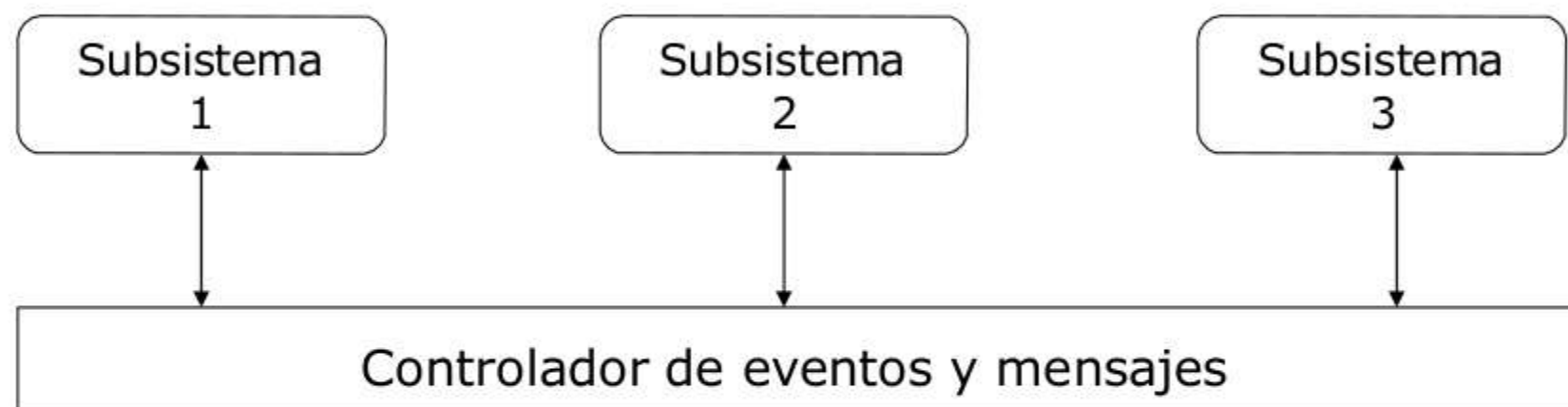
Modelos de Control

2. Dirigidos por eventos

- Los modelos de control dirigidos por eventos se rigen por eventos generados en el exterior.
- Existen varios sistemas dirigidos por eventos que se pueden desarrollar. Ej.: las hojas de calculo, en la que el valor cambiante de las celdas provoca que otras celdas se modifiquen.
- Se discuten solo dos modelos de este tipo:
 - **Modelo de Transmisión:** en estos modelos, un evento se transmite, en principio a todos los subsistemas. Cualquier subsistema que pueda manejar ese evento responde a él.

Modelos de Control

2. Dirigidos por eventos

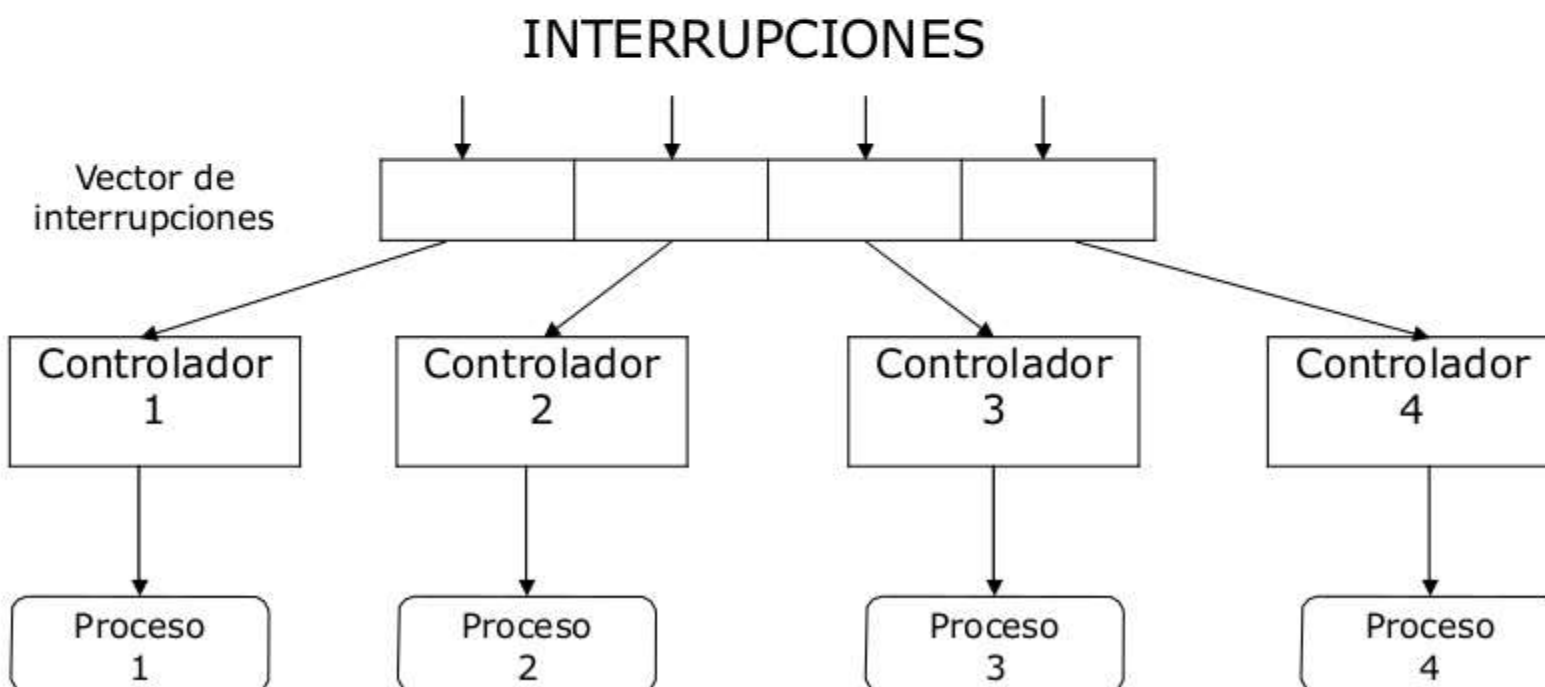


- La ventaja de este modelo, es su evolución relativamente sencilla. Un nuevo subsistema que maneje clases particulares de eventos, se puede integrar registrando sus eventos en el controlador de eventos.
- La desventaja es que los subsistemas no saben si los eventos se manejarán, ni cuando lo harán.

Modelos de Control

2. Dirigidos por eventos

- **Modelo dirigido por interrupciones:** Estos se utilizan exclusivamente en sistema de tiempo real, donde las interrupciones externas son detectadas por un controlador de interrupciones. Después éstos se pasan a algún componente para su procesamiento.

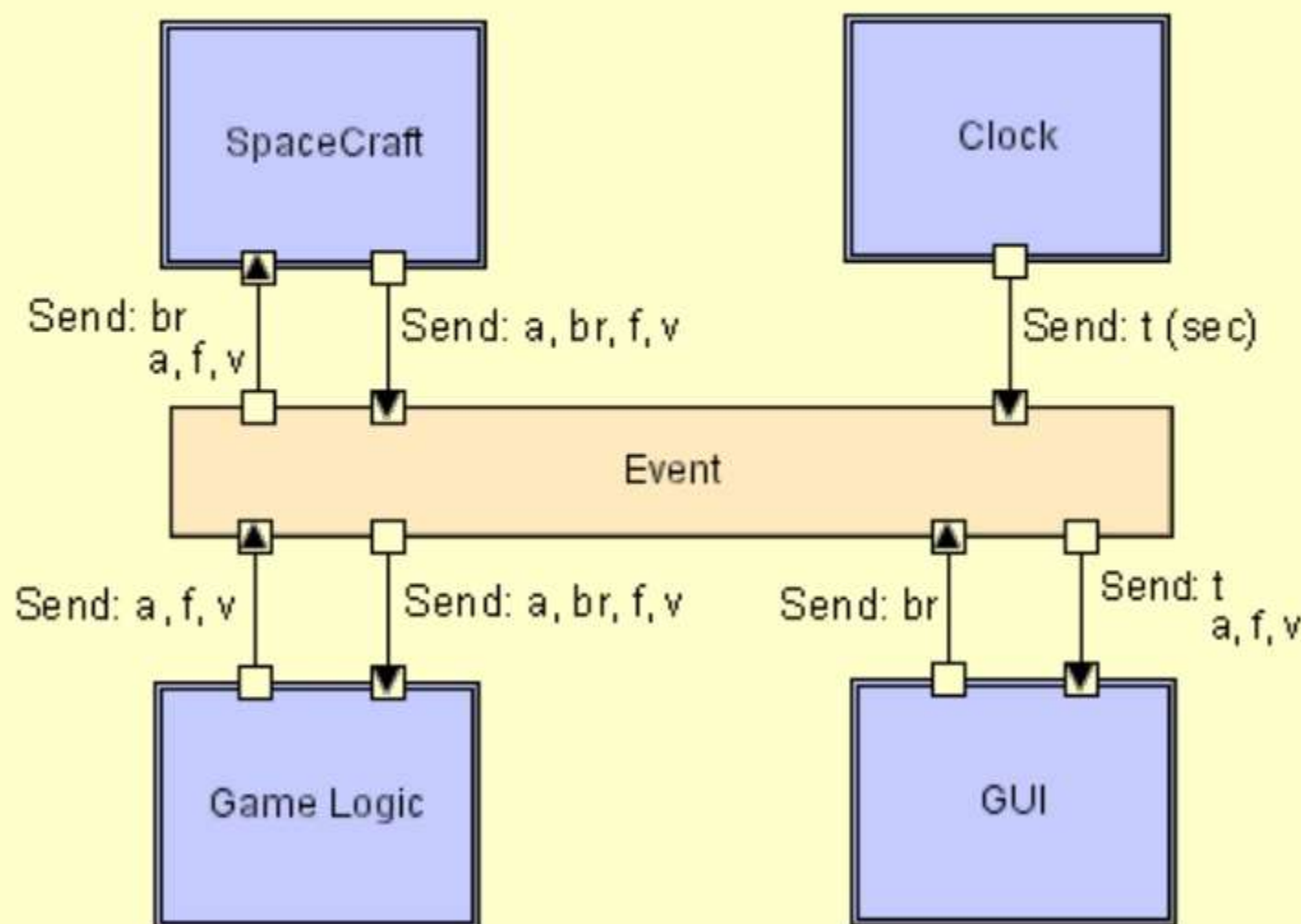


Modelos de Control

2. Dirigidos por eventos

- Los componentes independientes emiten y reciben de forma asincrónica eventos comunicados a través de buses de eventos
- **Componentes:** Consumidores y / o generadores de eventos simultáneos e independientes
- **Conectores:** buses de eventos (al menos uno) Elementos de datos: eventos: datos enviados como una entidad de primera clase a través del bus de eventos
- **Topología:** los componentes se comunican con los buses de eventos, no directamente entre sí.
- **Variantes:** la comunicación de componentes con el bus de eventos puede basarse en push o pull.
- Altamente escalable, fácil de evolucionar, efectivo para aplicaciones altamente distribuidas.

El módulo de aterrizaje lunar: Dirigidos por eventos



Etapas del proceso de diseño arquitectónico

- Las siguientes etapas son comunes para todos los procesos de diseño arquitectónicos:
 - **Estructuración del sistema:** se establecen los subsistemas y sus relaciones.
 - **Modelo de control:** se establece un modelo general de las relaciones de control entre las partes del sistema.
 - **Descomposición modular:** cada subsistema se descompone en módulos.

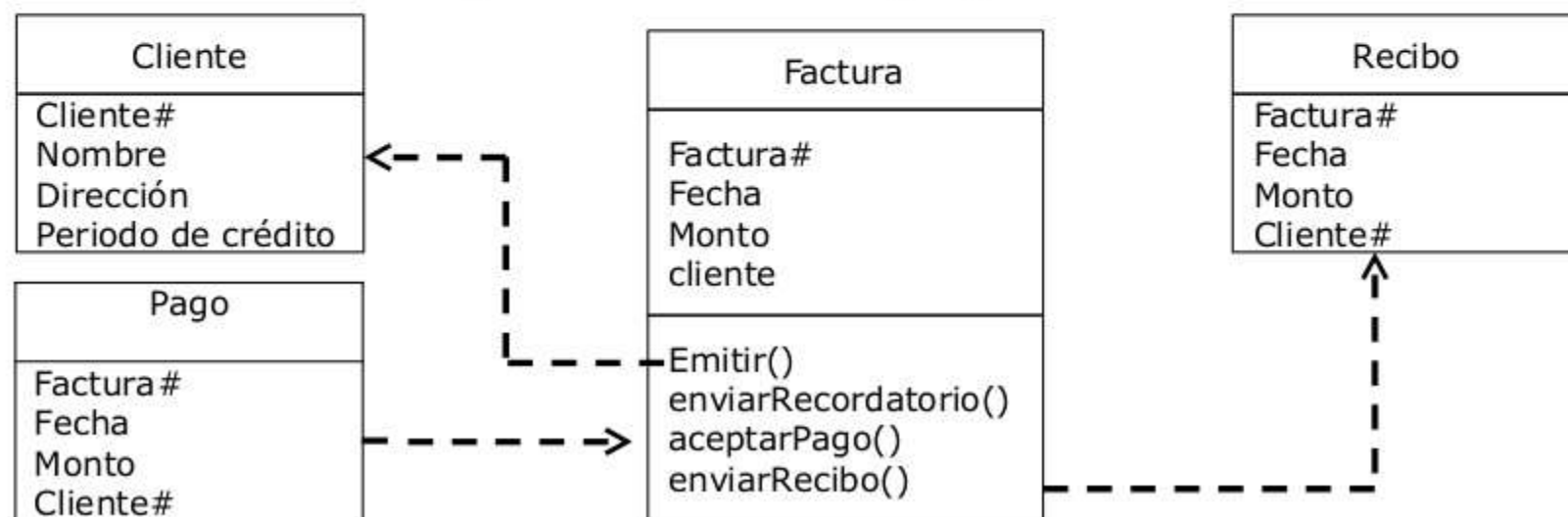
Descomposición modular

- Después de diseñar una arquitectura estructural, la siguiente etapa del proceso de diseño arquitectónico es descomponer los subsistemas en módulos.
- Se consideran dos modelos que son de utilidad cuando se descompone un subsistema en módulos:
 - **Modelo orientado a objetos:** el sistema se descompone en un conjunto de objetos que se comunican entre ellos.
 - **Modelo de flujo de datos:** el sistema se descompone en módulos funcionales que afectan entradas de datos y las transforman, de alguna manera, en datos de salida.
- En el modelo orientado a objetos, los módulos son objetos con estado privado y con operaciones definidas sobre ese estado.
- En el modelo de flujo de datos, los módulos son transformaciones funcionales.
- En ambos casos, los módulos pueden implementarse como componentes secuenciales o como procesos.

Descomposición modular

1. Modelo de Objetos

- El modelo orientado a objetos de una arquitectura de sistema, estructura el sistema en un conjunto de objetos débilmente acoplados con interfaces bien definidas.
- Los objetos llaman a los servicios ofrecidos por otros objetos.
- Una descomposición orientada a objetos, comprende clases de objetos, sus atributos y operaciones.



Descomposición modular

1. Modelo de Objetos

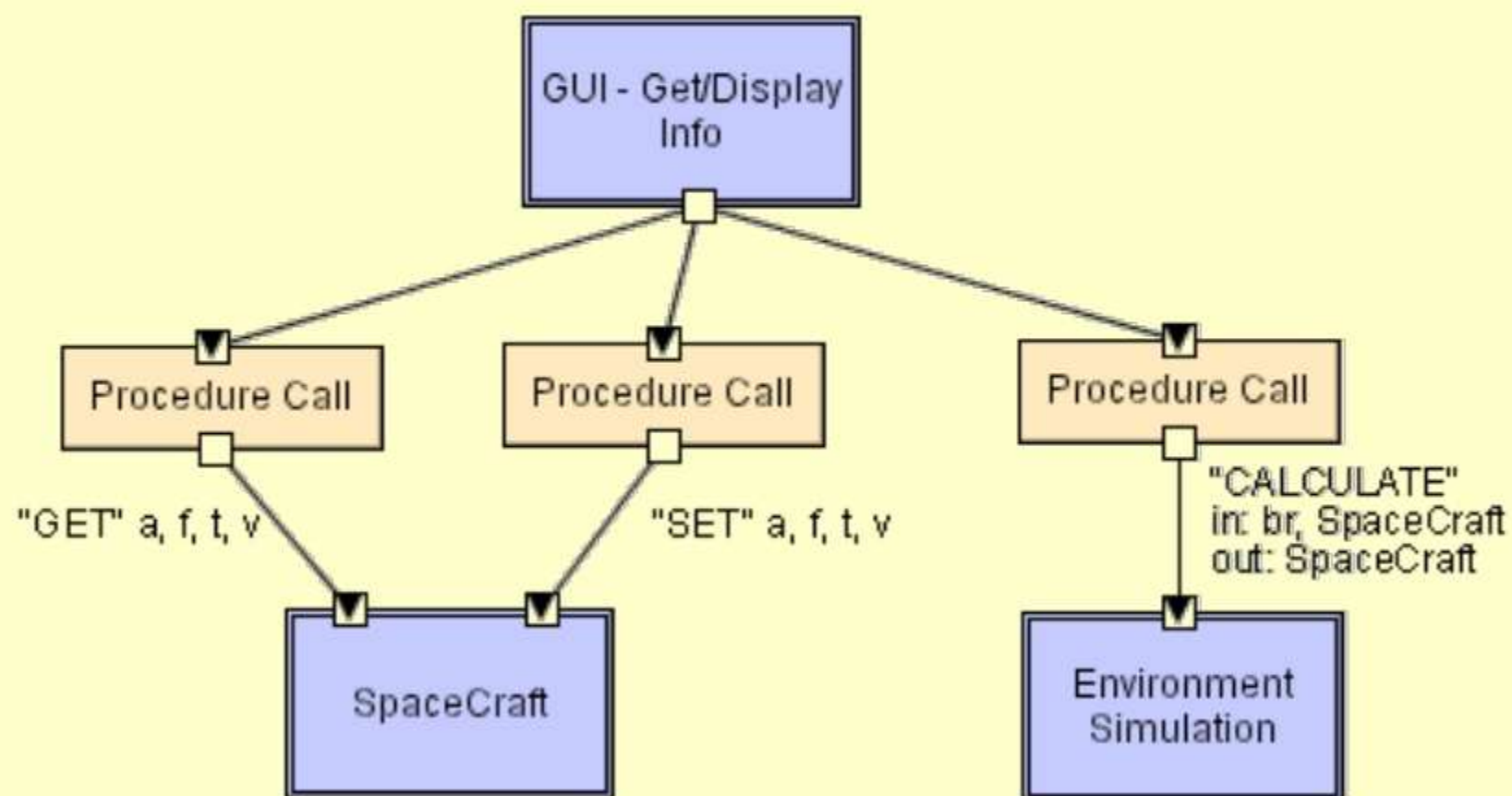
- Las ventajas de este modelo son bien conocidas:
 - Puesto que los objetos están débilmente acoplados, la implementación de objetos se puede modificar sin afectar a otros objetos.
 - Además, los objetos se pueden reutilizar.
- La desventaja, es que los objetos para utilizar los servicios deben hacer referencia explícita al nombre y la interfaz de otros objetos.

Descomposición modular

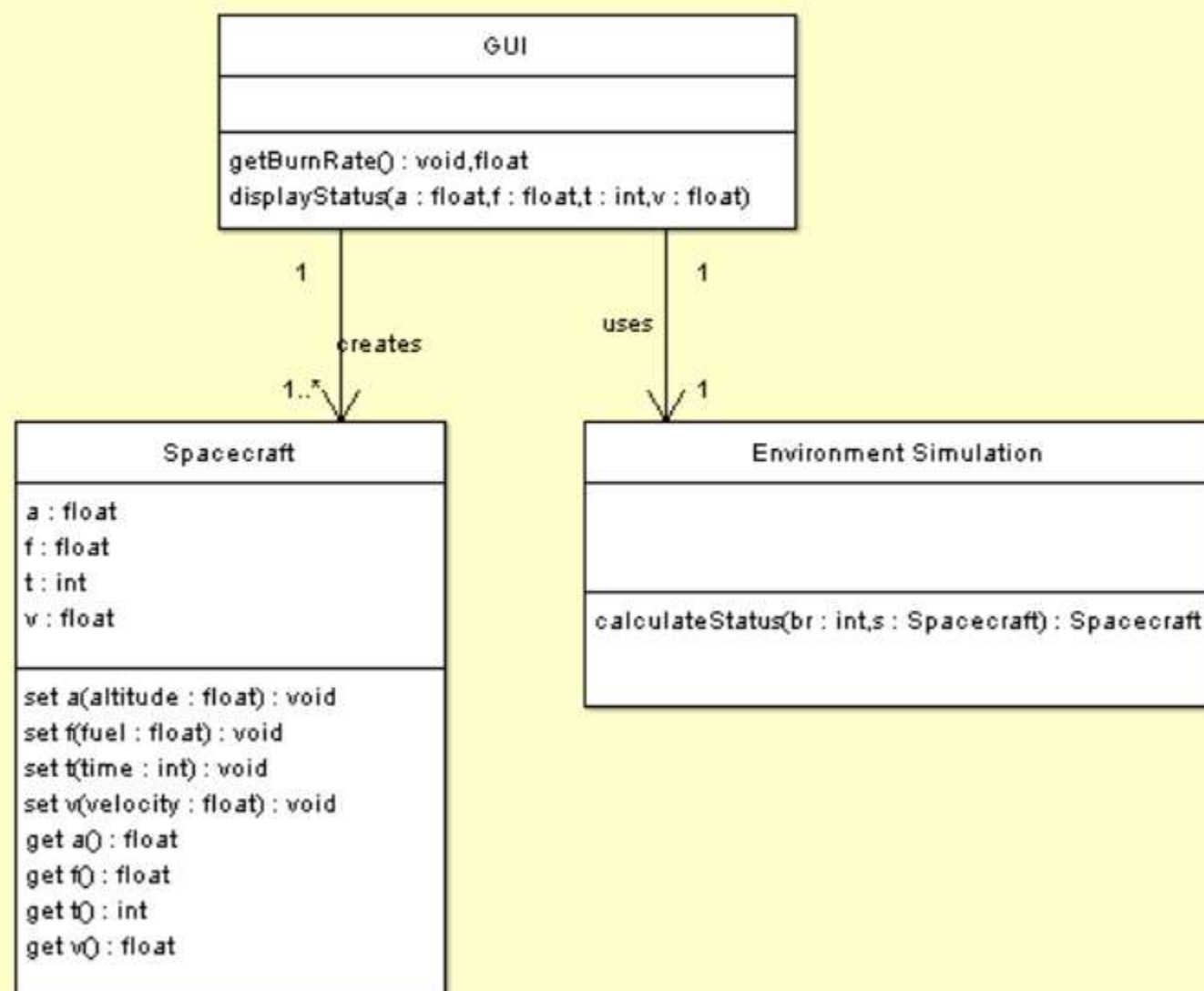
1. Modelo de Objetos

- Los **componentes son objetos**
 - Datos y operaciones asociadas
 - Los **conectores** son **mensajes e invocaciones de métodos**
- Invariantes de estilo
 - Los objetos son responsables de la integridad de su representación interna.
 - La representación interna está oculta a otros objetos.
- Ventajas
 - "Maleabilidad infinita" de los componentes internos del objeto
 - Descomposición del sistema en conjuntos de agentes que interactúan
- Desventajas
 - Los **objetos deben conocer las identidades de los servidores.**
 - Efectos secundarios en las invocaciones de métodos de objeto

El módulo de aterrizaje lunar: estilo arquitectónico orientado a objetos



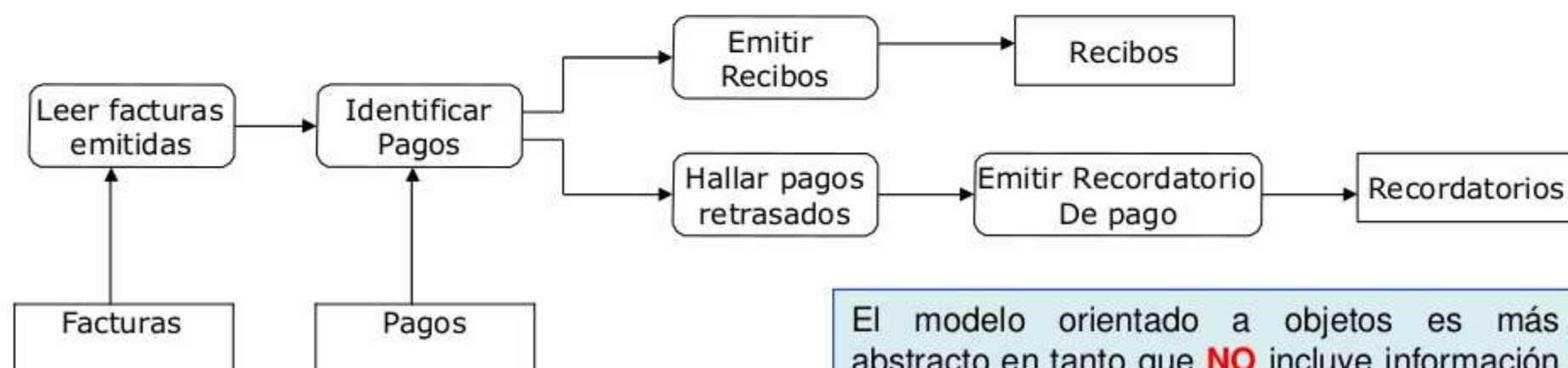
El módulo de aterrizaje lunar en UML



Descomposición modular

2. Modelo de Flujo de datos

- En un modelo de flujo de datos, las transformaciones funcionales procesan sus entradas y producen sus salidas .
- Los datos fluyen de un lugar a otro y se transforman durante este movimiento.
- Las transformaciones se pueden ejecutar en forma secuencial o en paralelo.



El modelo orientado a objetos es más abstracto en tanto que **NO** incluye información sobre la **secuencia de operaciones**

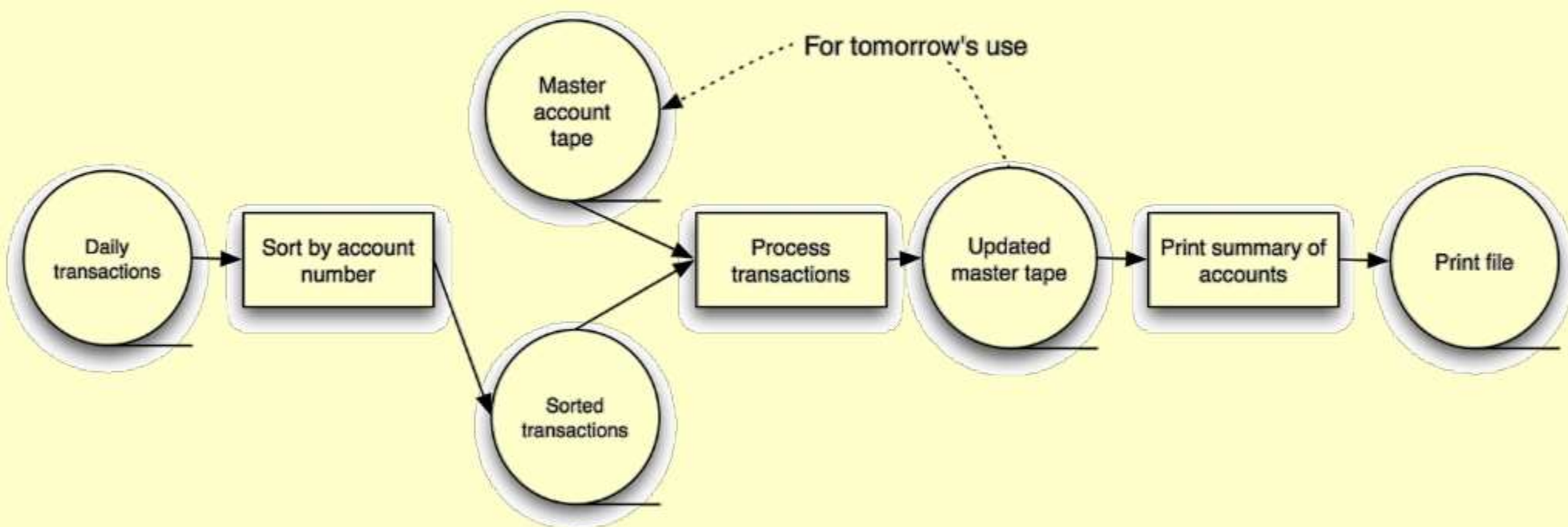
Descomposición modular

2. Modelo de Flujo de datos

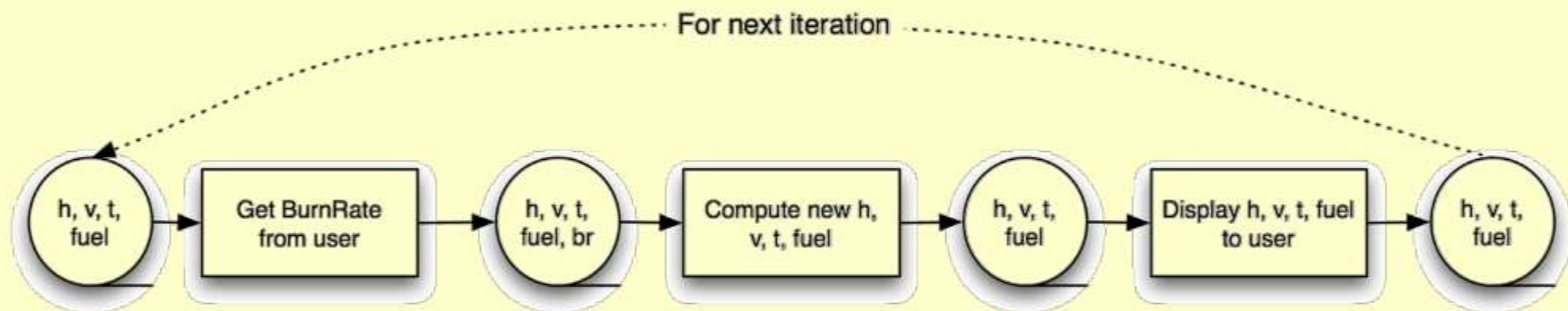
Batch Sequential (Procesamiento por lotes)

- Los programas separados se ejecutan en orden; los datos se pasan como un agregado de un programa al siguiente.
- Conectores: "La mano humana" que lleva cintas entre los programas, también conocido como "sneaker-net"
- Elementos de datos: elementos agregados explícitos que se pasan de un componente al siguiente una vez finalizada la ejecución del programa de producción..
- Usos típicos:
 - Procesamiento de transacciones en sistemas financieros. "El abuelo de los estilos"

Procesamiento por lotes: una aplicación financiera



El módulo de aterrizaje lunar: estilo arquitectónico orientado a flujo de datos



No es una receta para una misión lunar exitosa!

Descomposición modular

2. Modelo de Flujo de datos

Ventajas de esta arquitectura:

- ✓ Permite la reutilización de transformaciones.
- ✓ Es intuitivo.
- ✓ Permite que el sistema evolucione al agregar nuevas transformaciones.
- ✓ Sencilla de implementar, ya sea como un sistema concurrente o secuencial.

Desventajas

- ✗ Cada transformación debe estar acorde con las transformaciones que se comunica, en el formato de los datos a ser procesado.

Descomposición modular

2. Modelo de Flujo de datos

Desventajas

- ✗ Los sistemas interactivos son difíciles de escribir con el modelo tubería y filtro, debido a la necesidad de procesar una secuencia de datos.
 - Aunque las entradas y salidas textuales simples pueden modelarse de esta forma...
 - las interfaces gráficas de usuario tienen formatos I/O más complejos.
 - Tienen además una estrategia de control que se basa en eventos como clics del mouse o selecciones del menú.
 - Es difícil traducir esto en una forma compatible con el modelo *pipelining* (entubamiento o tubos y filtros).

Problemas en la creación del diseño

Modularidad y nivel de abstracción

- En un diseño modular, los componentes tienen entradas y salidas bien definidas y cada componente tiene un propósito previamente establecido.
- Los componentes modulares están organizados en una jerarquía, como resultado de la descomposición o abstracción.
- A medida que nos desplazamos hacia abajo encontramos mayor nivel de detalle para cada componente.
- El nivel más alto es el más abstracto.
- Los niveles más abstractos se resuelven primero y la solución se alcanza al desarrollar el detalle.

Problemas en la creación del diseño

Modularidad y nivel de abstracción

- La modularidad oculta los detalles. Una ventaja de este “**ocultamiento de información**”, es que cada componente oculta de los demás una decisión de diseño.
- Por esto, es probable que cambien las decisiones de diseño, pero el diseño como un todo puede permanecer intacto, solo cambia el diseño de componentes.



Universidad de Guayaquil



Unidad 1

Definición y diseño de software en su ciclo de vida de desarrollo

Final de la unidad



Diseño y Arquitectura de Software





Carrera de **SOFTWARE** Ciencia e innovación para la excelencia



Definición del proceso de venta de una empresa

Taller Unidad 1 – Especificación de requisitos

Ph.D. Franklin Parrales
15/12/2021
0

0

Diseño y Arquitectura de Software

Introducción

- ¿Has trabajado con sistemas de ventas?
- ¿Cuáles son los requerimientos funcionales del sistema con el que has trabajado?
- ¿Conoces otros tipos de sistemas de ventas, por ejemplo ventas por Internet, venta directa en comercios a clientes?

Ph.D. Franklin Parrales
15/12/2021
1

1

Diseño y Arquitectura de Software

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Ph.D. Franklin Parrales
15/12/2021
2

2

Diseño y Arquitectura de Software

Proceso de pedido de venta

- Antes de definir los requerimientos funcionales de un sistema de ventas, es importante obtener un entendimiento de los pasos del proceso de negocio a representar.
 - Previamente a que un sistema de ventas entre en funcionamiento, es necesario registrar los productos que se van a vender y sus precios. Estos registros se podrán modificar posteriormente.
 - Luego de esto, ante cada transacción de cliente, primero es necesario registrar sus datos, por ejemplo el nombre, número fiscal y dirección, para luego registrar lo que desea pedir.

Ph.D. Franklin Parrales
15/12/2021
3

3

Diseño y Arquitectura de Software

Proceso de pedido de venta

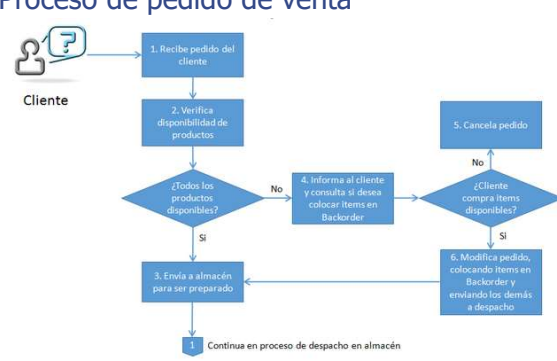
- Luego del registro del pedido, se procede a registrar un despacho de mercancía y la factura.
- Es por esto que el flujograma en la gerencia de proyectos es fundamental.
- En la siguiente figura, mostramos un flujograma general para un proceso de ventas...

Ph.D. Franklin Parrales
15/12/2021
4

4

Diseño y Arquitectura de Software

Proceso de pedido de venta



Ph.D. Franklin Parrales
15/12/2021
5

5

Proceso de pedido de venta

- Pueden existir variantes a este proceso, por ejemplo:
 - En un sistema de ventas al detal, el pedido, entrega y factura es una sola transacción.
 - Por otra parte, si lo que estamos vendiendo es un servicio, no manejamos un pedido sino un convenio, solicitud de servicio o contrato establecido previamente.

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Requisitos funcionales

- El proceso de ventas en el sistema abarcará los siguientes pasos: Ingreso de pedido de venta, emisión de orden de entrega (despacho), facturación y cobranza.
- El sistema enviará un correo electrónico cuando se registre alguna de las siguientes transacciones: pedido de venta de cliente, despacho de mercancía al cliente, emisión de factura a cliente y registro de pago de cliente.

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Requisitos funcionales: Maestro de clientes

- El sistema manejará un registro maestro de clientes.
 - Sólo los usuarios autorizados podrán ingresar nuevos clientes, modificar los datos o eliminarlos.
 - Para crear un registro maestro de cliente, se solicitará como mínimo el nombre del cliente y su número de identificación fiscal.
- Se permitirán asociar varias direcciones al registro de cliente y se le podrá asignar a cada dirección un propósito, por ejemplo dirección fiscal, dirección de despacho, entre otros.
 - Al crear el registro maestro, el sistema asignará un código de cliente.
- Se permitirá ingresar clientes sin número de identificación fiscal o dirección, sin embargo, no se podrán procesar pedidos a dichos clientes hasta que cuenten con esa información.
- El registro maestro de clientes tendrá asociado un límite de crédito.

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Diseño y Arquitectura de Software

Requisitos funcionales: Maestro de materiales (ítems) y lista de precios

- El sistema permitirá manejar un maestro de materiales y productos, en el cual se registrarán todos los ítems que se pueden vender.
 - Cada ítem debe tener al menos su nombre, unidad de medida y descripción.
- El sistema debe manejar listas de precios.
 - El usuario podrá agregar, modificar o eliminar listas de precios.
- Las listas de precios tendrán un período de vigencia (fecha inicial y fecha final).
- Al crear la lista de precios se agregarán los materiales e ítems, los cuales deben estar registrados previamente en el maestro de materiales.
 - Para cada uno deberá especificarse un precio.
- El acceso a la lista de precios estará restringido solamente a un grupo de usuarios autorizados.
- Se podrán crear listas de precios convenidos con un cliente. La misma tendrá fechas inicio y fin de vigencia del convenio y se le podrán agregar ítems del maestro de materiales y definir su precio convenido.

Ph.D. Franklin Parrales 15/12/2021 12

12

Diseño y Arquitectura de Software

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
 - Ambigüedad en lenguaje natural

Ph.D. Franklin Parrales 15/12/2021 13

13

Diseño y Arquitectura de Software

Requisitos funcionales: Pedidos de venta (1/5)

- Al ingresar un pedido de venta se deberá asignar un cliente.
 - El cliente deberá estar creado en el maestro de clientes,
 - Se permitirá realizar búsquedas por nombre de cliente o número fiscal.
- Al ingresar el pedido de venta, se podrán ingresar una o más líneas al pedido.
 - El usuario podrá seleccionar el ítem a agregar de una lista que proviene de un registro maestro de materiales y lista de precios.

Ph.D. Franklin Parrales 15/12/2021 14

14

Diseño y Arquitectura de Software

Requisitos funcionales: Pedidos de venta (2/5)

- Al seleccionar un ítem, se mostrará su descripción y su precio.
 - Se podrán realizar búsquedas por nombre o familia de material.
 - Para finalizar el registro de la línea, es obligatorio especificar la cantidad.
- Al seleccionar un ítem de pedido, el precio será determinado a partir de las listas de precio autorizado.
 - Si existe un convenio vigente para el ítem con el cliente seleccionado, se tomará el precio del convenio.
 - Si no existe un convenio se tomará el precio de la lista general de precios.
 - El precio no podrá ser modificado.

Ph.D. Franklin Parrales 15/12/2021 15

15

Diseño y Arquitectura de Software

Requisitos funcionales: Pedidos de venta (3/5)

- La dirección de facturación y despacho del pedido se obtendrá del maestro de cliente seleccionado.
 - El usuario no podrá modificar las direcciones.
- El límite de crédito se verificará al momento de ingresar un pedido.
 - En caso de existir pedidos abiertos para el cliente por montos superiores al límite de crédito, se mostrará un mensaje de alerta y no se permitirá registrar el pedido en definitivo, pero si lo podrá registrar como borrador.
- En todo momento, el pedido se podrá registrar en borrador, pudiendo ser modificado y registrado en definitivo posteriormente.

Ph.D. Franklin Parrales 15/12/2021 16

16

Diseño y Arquitectura de Software

Requisitos funcionales: Pedidos de venta (4/5)

- El sistema llevará un control de las cantidades que se están pidiendo, en relación con las cantidades existentes en inventario.
 - Al registrar un pedido, en caso que exista inventario suficiente el sistema dará de alta al pedido y registrará una reserva de ese material o producto reservada para ese pedido.
- Al ingresar el pedido en definitivo, el sistema verificará línea por línea las cantidades pedidas vs. Cantidades de materiales existentes en inventario.
 - Si la cantidad pedida excede las cantidades existentes en inventario más las cantidades que ya están reservadas para otros pedidos, la línea de pedido afectada será colocada en una cola de pedidos pendientes.

Ph.D. Franklin Parrales 15/12/2021 17

17

Diseño y Arquitectura de Software

Requisitos funcionales: Pedidos de venta (5/5)

- Mientras existan líneas de pedido en una cola, las líneas de pedidos subsecuentes del mismo material serán colocadas también en la cola.
- Un proceso automatizado revisará periódicamente las líneas de pedido que se encuentran en cola, si ya existe material o productos para poder pasar la línea a definitiva el sistema lo realizará automáticamente.
 - El sistema dará de alta al pedido y registrará una reserva de ese material o producto reservada para ese pedido.

Ph.D. Franklin Parrales 15/12/2021 18

18

Diseño y Arquitectura de Software

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Ph.D. Franklin Parrales 15/12/2021 19

19

Diseño y Arquitectura de Software

Requisitos funcionales: Ordenes de entrega (1/2)

- A cada orden se le asignará un identificador único, que será utilizado para reconocerla en todos los procesos subsecuentes que se realicen sobre esta.
- Una vez registrado el encabezado, se procederá a registrar las líneas de la orden de entrega.
- Toda línea de orden de entrega deberá estar asociada a una línea de pedido a la cual ya se le dio de alta en el sistema.
 - Esto es, que ya ha superado el proceso de verificación (descrito en los requerimientos funcionales de pedido), que garantiza que exista cantidad en inventario reservada para el mismo.

Ph.D. Franklin Parrales 15/12/2021 20

20

Diseño y Arquitectura de Software

Requisitos funcionales: Ordenes de entrega (2/2)

- Las cantidades de entrega se tomarán del pedido de venta asociado, no se podrán asignar materiales o ítems diferentes a los del pedido.
- El sistema permitirá órdenes de entrega parciales.
- Se permitirán líneas de orden de entrega asociadas a distintos pedidos, siempre y cuando estos sean del mismo cliente.
- Al registrar la orden de entrega, se realizará la rebaja de los inventarios de materiales o productos.

Ph.D. Franklin Parrales 15/12/2021 21

21

Diseño y Arquitectura de Software

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Ph.D. Franklin Parrales 15/12/2021 22

22

Diseño y Arquitectura de Software

Requisitos funcionales: Facturación de pedidos de venta (1/2)

- La facturación de pedidos de venta se realizara en lotes, por medio de una pantalla de pedidos pendientes de facturación, la cual mostrará los pedidos no facturados.
 - Una vez facturados los pedidos no se mostrarán en esta lista.
- Para factura asociada a pedido, sus líneas serán determinadas exclusivamente a partir del pedido, con las cantidades y precios definidos previamente.
 - No se podrán modificar ni cantidades ni precios.
- Se permitirá incluir varios pedidos de venta en una misma factura.
 - La línea de la factura mostrará el número de pedido al que esté asociado.

Ph.D. Franklin Parrales 15/12/2021 23

23

Diseño y Arquitectura de Software

Requisitos funcionales: Facturación de pedidos de venta (2/2)

- El sistema también permitirá el registro de facturas manuales no asociadas a pedidos, sin embargo, estas requerirán autorización por parte del grupo de Gerentes antes de ser contabilizadas.
- El libro de venta será emitido en el formato establecido por las autoridades tributarias de dicha materia.
- La contabilización de transacciones de facturas de venta podrá configurarse para realizarse de forma automatizada a su registro, o manualmente en lotes (Proceso Batch).

Ph.D. Franklin Parrales 15/12/2021 24

24

Diseño y Arquitectura de Software

Contenido

- Proceso de pedido de venta
- Requisitos funcionales
 - Maestro de clientes
 - Maestro de materiales (ítems) y lista de precios
 - Pedidos de venta
 - Ordenes de entrega (Despacho)
 - Facturación de pedidos de venta
- Ambigüedad en lenguaje natural

Ph.D. Franklin Parrales 15/12/2021 25

25

Diseño y Arquitectura de Software

Análisis de requisitos: LENGUAJE NATURAL

- Lenguaje natural:
 - Problemas de ambigüedad y escaso formalismo.
 - Los usuarios pueden establecer en sus propios términos los requisitos sin la intervención de formalismos.
 - Solución de problemas de comunicación entre usuarios y analistas.

Análisis de los requerimientos

- Analizar los requerimientos y filtrar las ambigüedades
- Dividir los enunciados en conjuntos homogéneos

Ph.D. Franklin Parrales 15/12/2021 26

26

Diseño y Arquitectura de Software

Lenguaje Natural: FILTRO DE AMBIGÜIDADES

7 REGLAS EMPÍRICAS PARA FILTRAR AMBIGÜIDADES

- R1. Elegir un nivel apropiado de abstracción para los términos
- R2. Evitar el uso de "casos" en lugar de conceptos generales
- R3. Evitar expresiones vagas o indirectas
- R4. Elegir estilo estandarizado de enunciado
- R5. Verificar sinónimos y homónimos
- R6. Hacer explícitas las referencias entre términos
- R7. Utilizar un Glosario

Ph.D. Franklin Parrales 15/12/2021 27

27

Diseño y Arquitectura de Software

- En una base de datos de una universidad, se representan
- datos sobre estudiantes y **profesores**. Para los estudiantes,
- se representa el apellido, edad, sexo, ciudad y provincia de
- nacimiento, ciudad y provincia de residencia de sus
- familias, **lugares** y provincias donde vivieron antes
- (con el **lapso** que vivieron en cada uno), cursos que han
- aprobado, con nombre, código, **profesor**, nota y fecha.
- Asimismo, se representan los cursos a los que asisten en
- la actualidad y para cada uno, **día**, **lugares** y horas de
- impartición de las clases (cada curso se imparte a lo sumo
- 11 una vez en un día). Para estudiantes graduados se
- representa el nombre del **tutor** y el número total de
- créditos en el último año. Para estudiantes de doctorado,
- se representa el título y **área de investigación** de su tesis.
- Para los **maestros**, se representa el apellido, la edad, el
- lugar** y provincia de nacimiento, nombre del departamento
- al que pertenecen, **número de teléfono**, título, **situación** y
- 18 **temas de investigación**.

Ph.D. Franklin Parrales 15/12/2021 28

28

Diseño y Arquitectura de Software

Lenguaje Natural: FILTRO DE AMBIGÜIDADES

- R1. Elegir nivel apropiado de abstracción para los términos
 - Los **términos abstractos** se usan con frecuencia en enunciados, en casos en que los **términos específicos** serían más apropiados.
 - Las categorías generales son comunes en el **lenguaje natural** porque producen una comunicación rápida y eficaz en la que, comúnmente, **la ambigüedad se resuelve por el contexto**.
 - Sin embargo, en DC se debe utilizar términos en un nivel correcto de abstracción, especialmente si el diseñador no es un experto en el dominio de la aplicación.

Términos Abstractos:	Términos Apreciados:
(lin. 5) lugares	ciudades
(lin. 6) lapso	número de años
(lin. 17) situación	estado civil

Ph.D. Franklin Parrales 15/12/2021 29

29

Diseño y Arquitectura de Software

Lenguaje Natural: FILTRO DE AMBIGÜEDADES

- R2. Evitar el uso de "casos" en lugar de conceptos generales
 - Esta regla evita la fuente opuesta de ambigüedades; los usuarios de los sistemas de información adoptan, a veces, **términos más específicos de lo necesario**.
 - Ejemplo, empresa de electrónica: «la cantidad en existencia de **chips**».
 - El término chips no describe un concepto, sino más bien un caso del concepto correcto, **componentes**.
- R3. Evitar las expresiones vagas o indirectas.
 - En el lenguaje natural se usa con frecuencia la **repetición deliberada** y las **expresiones indirectas**.
 - «mira a la persona sentada en la taquilla», en vez de: «mira al taquillero»
 - La segunda oración indica una clase específica de entidades (taquillero), mientras que la primera se refiere a la misma clase indicando una interrelación con otra clase de entidades (persona).
 - Al usar rodeos se incurre en el riesgo **expresar el significado de los conceptos** en términos de **referencias implícitas a otros conceptos**, en lugar de **referencias explícitas a los conceptos mismos**.

Ph.D. Franklin Parrales15/12/202130

30

Diseño y Arquitectura de Software

Algunas recomendaciones para especificar requisitos **NO** funcionales

Propiedad	Medida
Rapidez	Transacciones procesadas por segundo
	Tiempo de respuesta al usuario y a eventos
	Tiempo de actualización de la pantalla
Tamaño	K Bytes
Facilidad de Uso	Numero de chips de RAM
	Tiempo de entrenamiento
	Número de pantallas de ayuda
Fiabilidad	Número promedio entre fallos
	Probabilidad de no disponibilidad
	Tasa de Ocurrencia de fallos
	Disponibilidad
Robustez	Tiempo de reinicio entre fallos
	Porcentaje de eventos que provocan fallos
	Probabilidad de corrupción de los datos después de fallos
Portabilidad	Porcentaje de declaraciones dependientes de objetivos
	Número de sistemas objetivo

Ph.D. Franklin Parrales15/12/202131

31

Diseño y Arquitectura de Software

Brecha en la Comunicación (Scharer '90)

Según desarrolladores , los usuarios ...	Según usuarios , los desarrolladores ...
no saben lo que quieren	no captan las necesidades operativas
no pueden articular lo que quieren	ponen excesivo énfasis en aspectos meramente técnicos
muchas necesidades por motivos políticos	pretenden indicarnos cómo hacer nuestro trabajo
quieren todo ya	no son capaces de traducir necesidades claramente establecidas en un sistema
son incapaces de definir prioridades entre sus necesidades	siempre dicen que no
rehúsan asumir responsabilidades por el sistema	siempre están pasados del presupuesto
incapaces de dar un enunciado utilizable de sus necesidades	siempre están atrasados
no están comprometidos con los proyectos de desarrollo	nos exigen tiempo y esfuerzo aún a costa de las obligaciones esenciales
no aceptan soluciones de compromiso	establecen estándares no realistas para la definición de requisitos
no pueden mantener el cronograma	son incapaces de responder rápidamente a cambios en las necesidades

Ph.D. Franklin Parrales15/12/202132

32

Diseño y Arquitectura de Software

Lenguaje Natural: FILTRO DE AMBIGÜEDADES

R4. Elegir Estilo adecuado de enunciado

- En la libre conversación se usan muchos estilos sintácticos para lograr una comunicación más eficaz.
- Esta variedad de estilos debe evitarse en los textos que definen los requerimientos; **el uso de categorías sintácticas simples permite un modelado directo (y único) de los requerimientos**.
- Lo ideal: enunciados con un **estilo ESTÁNDAR**

Formato de las Descripciones:

<SUJETO><VERBO><ESPECIFICACIÓN>

Formato para las Operaciones:

<SI> <CONDICION> <ENTONCES> <ACCIÓN> <SI NO> <ACCIÓN>, o
<CUANDO> <CONDICION> <HACER> <ACCIÓN>

- Diseñador debe seleccionar un estilo apropiado con **EQUILIBRIO** entre **ESTANDARIZACIÓN** y **EXPRESIVIDAD**

Ph.D. Franklin Parrales15/12/202133

33

Diseño y Arquitectura de Software

Lenguaje Natural: FILTRO DE AMBIGÜEDADES

R5. Verificar **SINÓNIMOS** y **HOMÓNIMOS**

- Los requerimientos suelen resultar de las contribuciones de varios usuarios. Distintas personas pueden dar
 - un **mismo significado a diferentes palabras (sinónimos)**
 - o **diferente significado a las mismas palabras (homónimos)**.
- Riesgo de homónimos mayor
 - con vocabulario de términos es pequeño,
- Riesgo de sinónimos mayor
 - con vocabulario de términos es rico.
 - Si usuarios distintos adoptan vocabularios en diferentes niveles de abstracción
- Ejemplo:
 - maestro, profesor y tutor ... son sinónimos
 - lugares ... se usa dos veces con diferente significado (homónimos)

Ph.D. Franklin Parrales15/12/202134

34

Diseño y Arquitectura de Software

Lenguaje Natural: FILTRO DE AMBIGÜEDADES

R6. Hacer explícitas las referencias entre términos

- Si no se especifica claramente las referencias entre los términos puede surgir ambigüedad

número de teléfono

¿es una propiedad de los PROFESORES, o de los DEPARTAMENTOS?

- Los conceptos referidos pueden aparecer explícitamente en los requisitos u omitirse por completo

* PROFESOR, DEPARTAMENTO sí aparecen en los requisitos

* día ----- ¿día de la semana o día del mes?

SEMANA y MES no se mencionan en los requisitos

Ph.D. Franklin Parrales15/12/202135

35

Lenguaje Natural: FILTRO DE AMBIGÜEDADES

R7. Utilizar un Glosario de términos

- Entender significado de los términos y Eliminar ambigüedades de los requisitos
 - 1º Crear el Glosario
 - 2º Descripciones de requisitos usando sólo términos del glosario
- En el Glosario, para cada término debería aparecer:
 - NOMBRE
 - DEFINICIÓN CORTA
 - POSIBLES SINÓNIMOS (área de equivalencia)
 - POSIBLES PALABRAS CLAVE (área de influencia)



FINAL

Definición del proceso de venta de una empresa

Taller Unidad 1 – Especificación de requisitos

Diseño y Arquitectura de Software



Actores, Casos de Uso, Escenarios

Taller de la Unidad 2

Ph.D. Franklin Parrales

15/12/2021

0

0

Diseño y Arquitectura de Software

Modelado de Análisis

- Una vez completado el modelo de casos de uso (CU) se ha llegado a obtener diagramas de casos de uso en determinados niveles que ya no se pueden explotar más.
- Si se intentara explotar los CU, se pasaría a describir el comportamiento interno de las funciones con artefactos inadecuados.
- Los *casos de uso contenidos en estos diagramas se denominan casos de uso elementales*.
- Esta *situación límite* indica que *se debe pasar a trabajar con otros artefactos, que son los del modelo de análisis*:
 - Clases de análisis.
 - Asociaciones.
 - Diagramas de clases.
 - Diagramas de colaboración asociados a los diagramas de clases.

Ph.D. Franklin Parrales

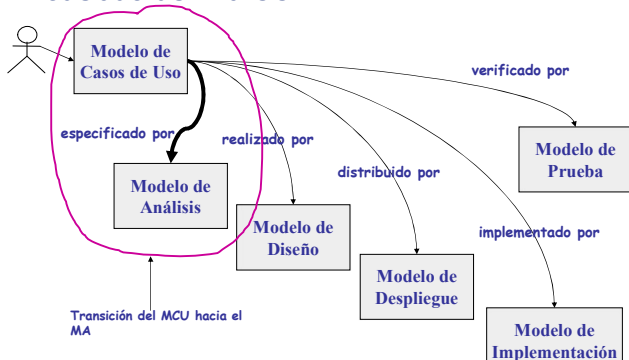
15/12/2021

1

1

Diseño y Arquitectura de Software

Modelado de Análisis



Transición del MCU hacia el MA

Ph.D. Franklin Parrales

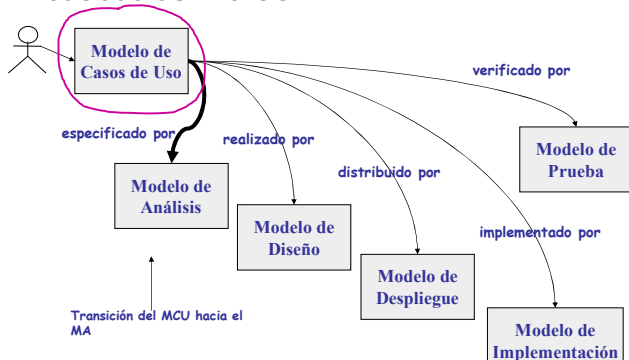
15/12/2021

2

2

Diseño y Arquitectura de Software

Modelado de Análisis



Transición del MCU hacia el MA

Ph.D. Franklin Parrales

15/12/2021

3

3

Diseño y Arquitectura de Software

Modelado de Análisis

- El Análisis en el *Proceso Unificado de Rational (RUP)*, también denominado *Proceso Unificado de Desarrollo de Software (SDUP)*, se realiza por medio de los flujos de trabajo:
 - Análisis y diseño.
- El resultado del Análisis es el siguiente:
 - Modelo de Análisis.
- El Modelo de Análisis contiene:
 - La Vista de Diseño de UML.
 - La Vista de Procesos de UML.

Ph.D. Franklin Parrales

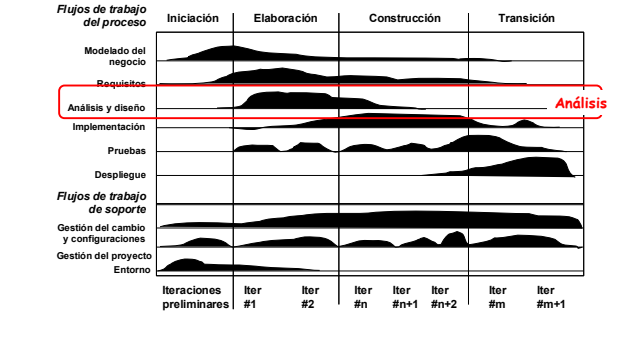
15/12/2021

4

4

Diseño y Arquitectura de Software

Modelado de Análisis

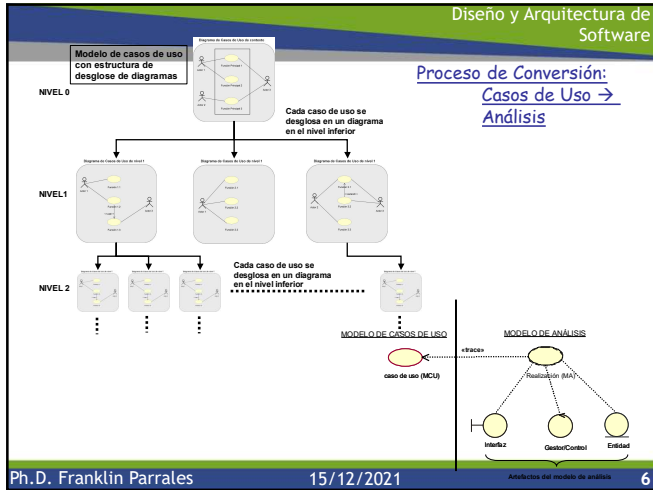


Ph.D. Franklin Parrales

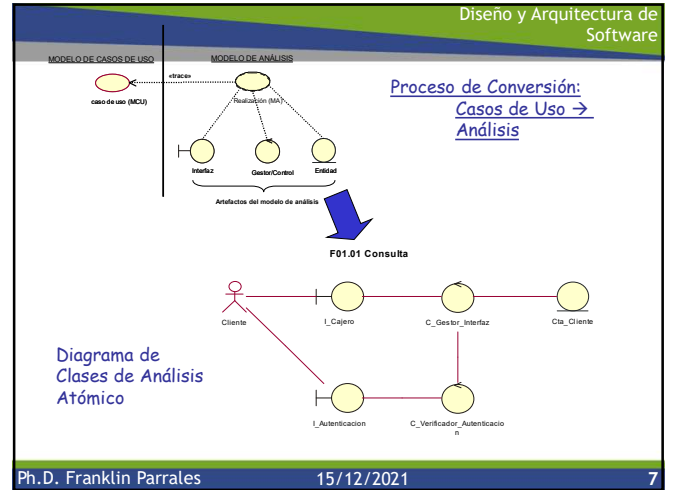
15/12/2021

5

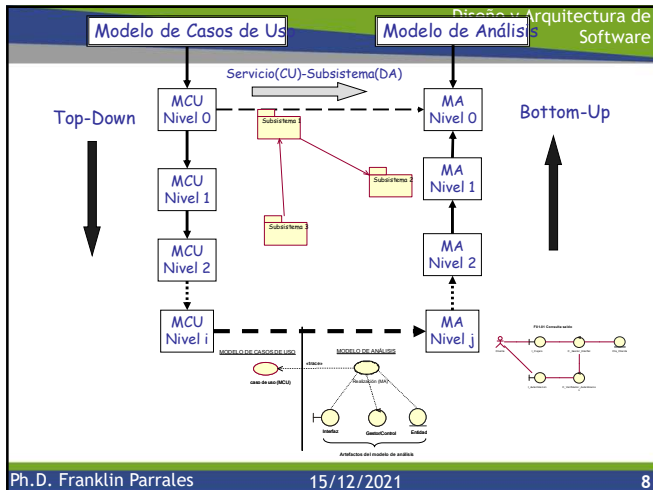
5



6



7



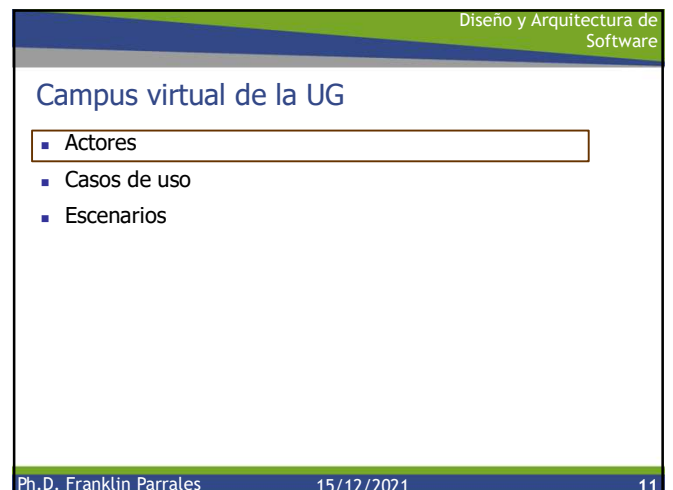
8



9



10



11

Diseño y Arquitectura de Software

Descripción de Actores: Profesor

Número	ACT-1
Actor	Profesor
Descripción	Persona responsable de dictar la materia y de administrar sus cursos. Es un actor primario.
Responsabilidades	El profesor puede realizar lo siguiente: • Publicar información del curso, tal como: anuncios, actividades, tareas, recursos, información general, políticas del curso, políticas de calificaciones, etc. • Agregar o eliminar alumnos al curso y definir sus ayudantes académicos. • Publicar las calificaciones de sus estudiantes a través de archivos existentes o mediante el ingreso directo al sistema. • Entre otros.
Fuentes	Gestoría de personal docente

Ph.D. Franklin Parrales 15/12/2021 12

12

Diseño y Arquitectura de Software

Descripción de Actores: Alumno

Número	ACT-2
Actor	Alumno
Descripción	Persona que utiliza la información disponible en el aula virtual. Es un actor primario.
Responsabilidades	El alumno puede realizar lo siguiente: • Acceder al material expositivo del curso. • Realizar lecciones en línea. • Subir tareas individuales y grupales. • Realizar un trabajo colaborativo (foros y chat) en donde se de un ambiente de intercambio, manejo, uso, de la información con todos los otros miembros que forman el Aula Virtual. • Consultar calificaciones. • Evaluar el curso.
Fuentes	Coordinador de la carrera

Ph.D. Franklin Parrales 15/12/2021 13

13

Diseño y Arquitectura de Software

Campus virtual de la UG

- Actores
- Casos de uso
- Escenarios

Ph.D. Franklin Parrales 15/12/2021 14

14

Diseño y Arquitectura de Software

Lista de casos de uso

- Publicar anuncio
- Publicar evento en un día de calendario
- Publicar tarea
- Responder tarea
- Revisar tarea
- Publicar recurso

Ph.D. Franklin Parrales 15/12/2021 15

15

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO:	NOMBRE:
CU-1	Publicar anuncio
COMPLEJIDAD: Baja	PRIORIDAD: Alta
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO	
ACTORES: Profesor	
CASOS DE USO ASOCIADOS: NINGUNO	
DESCRIPCIÓN: El profesor ingresa un anuncio que podrá ser consultado por los miembros del curso.	
NOTAS: • Se podrá especificar en forma de texto o en forma de archivo • Si se desea se puede enviar un mail a cada uno de los integrantes del curso, informándoles de que existe un nuevo anuncio publicado. • El publicar una nueva tarea o un nuevo recurso académico, también permitirá publicar un nuevo anuncio de manera automática. • Los anuncios deberán aparecer ordenados por fecha de publicación descendientemente. • El último anuncio publicado debe ser más notorio que los demás y se debe mostrar su contenido.	
CRITERIOS DE ACEPTACIÓN: El anuncio es publicado o no	

Ph.D. Franklin Parrales 15/12/2021 16

16

Diseño y Arquitectura de Software

Campus virtual de la UG

- Actores
- Casos de uso
- Escenarios

Ph.D. Franklin Parrales 15/12/2021 17

17

Diseño y Arquitectura de Software

Lista de escenarios del Casos de Uso **CU-1**

■ **Escenarios:**

- 1.1 El anuncio se publica en forma de texto exitosamente.
- 1.2 El anuncio se publica en forma de archivo exitosamente.
- 1.3 No se puede publicar el anuncio en forma de archivo porque éste ya existe en el servidor.
- 1.4 No se puede publicar el anuncio en forma de archivo porque el tamaño de éste sobrepasa al tamaño máximo especificado para el usuario.

Ph.D. Franklin Parrales 15/12/2021 18

18

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO: CU-1	NOMBRE: Publicar anuncio
... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-1.1	NOMBRE: El anuncio se publica en forma de texto exitosamente. SUPOSICIONES/ASUNCIONES: • El directorio de trabajo del paralelo existe • Se ha ingresado una descripción y un texto. • El usuario especifica enviar mails. RESULTADOS: • Un nuevo anuncio es registrado. • Es generado un archivo texto con el contenido del anuncio. • Un mail es enviado a cada uno de los integrantes del curso informándoles de la publicación de una nueva tarea. • Un mensaje de éxito es presentado con los datos registrados.

Ph.D. Franklin Parrales 15/12/2021 19

19

Diseño y Arquitectura de Software





Carrera de **SOFTWARE**

Ciencia e innovación para la excelencia





Caso 2:

Internet Banking

Ph.D. Franklin Parrales 15/12/2021 20

20

Diseño y Arquitectura de Software

Sistema de Internet Banking para el móvil

- Los clientes podrán ingresar al sistema, consultar sus cuentas corrientes y pagar su planilla telefónica.
- Para ingresar al sistema, los usuarios deben ingresar su número de cédula y su clave personal.
 - Si el cliente ingresa 3 veces mal la clave, su ingreso al sistema se bloquea.
- Los clientes podrán consultar los saldos disponible, por confirmar y total de sus cuentas corrientes.
- El cliente podrá pagar su planilla telefónica usando su cuenta corriente.
- Se podrán realizar pagos de teléfonos propios y de terceros.
- El sistema de Internet Banking accederá al Mainframe del banco (sistema externo) donde accederá a los datos de las cuentas de clientes. Asumir que los datos de los teléfonos también son obtenidos de un sistema externo que pertenece a una compañía telefónica.

Ph.D. Franklin Parrales 15/12/2021 21

21

Diseño y Arquitectura de Software

Tareas

- Identificar los actores, especificarlos formalmente.
- Listar los casos de uso y especificarlos formalmente.
- Listar los escenarios y especificarlos formalmente.
- Realizar el Diagrama de Casos de Uso

Ph.D. Franklin Parrales 15/12/2021 22

22

Diseño y Arquitectura de Software

Descripción de Actores: Cliente

Número	ACT-1
Actor	Cliente
Descripción	Persona natural que realiza transacciones bancarias por internet. Es un actor primario.
Responsabilidades	El Cliente puede realizar lo siguiente: • Ingresar al sistema • Consultar su saldo disponible, por confirmar y total de sus cuentas corrientes. • Pagar su planilla telefónica y la de otros clientes.
Fuentes	Departamento de atención al cliente

Ph.D. Franklin Parrales 15/12/2021 23

23

Diseño y Arquitectura de Software

Descripción de Actores: Mainframe

Número	ACT-2
Actor	Mainframe
Descripción	Sistema externo del cual el sistema de Internet Banking obtiene los datos de las cuentas de los clientes.
Responsabilidades	El Mainframe realiza lo siguiente: Acceder a los datos de las cuentas de clientes.
Fuentes	Departamento de sistemas

Ph.D. Franklin Parrales 15/12/2021 24

24

Diseño y Arquitectura de Software

Descripción de Actores: Mainframe

Número	ACT-3
Actor	Sistema (externo) de la Compañía Telefónica
Descripción	Sistema externo del cual el sistema de Internet Banking que contiene información de pago de los usuarios.
Responsabilidades	El sistema externo realiza lo siguiente: - Enviar los datos pago de las planillas telefónicas. - Validar el pago de la planilla
Fuentes	Departamento de sistemas

Ph.D. Franklin Parrales 15/12/2021 25

25

Diseño y Arquitectura de Software

Tareas

- Identificar los actores, especificarlos formalmente.
- Listar los casos de uso y especificarlos formalmente.
- Listar los escenarios y especificarlos formalmente.
- Realizar el Diagrama de Casos de Uso

Ph.D. Franklin Parrales 15/12/2021 26

26

Diseño y Arquitectura de Software

Lista de Casos de Uso

- 1) Ingresar al Sistema
- 2) Consultar Saldos de Cuentas Corrientes
- 3) Pagar Planilla Telefónica

Ph.D. Franklin Parrales 15/12/2021 27

27

Diseño y Arquitectura de Software

Especificación de Casos de Uso

IDENTIFICADOR CASO DE USO: CU-1	NOMBRE: Ingresar al Sistema
COMPLEJIDAD: Baja	PRIORIDAD: Alta
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO	
ACTORES: Cliente	
CASOS DE USO ASOCIADOS: NINGUNO	
DESCRIPCIÓN: Los clientes ingresarán al sistema con su identificador y clave personal.	
NOTAS: - El identificador será la cédula. - Si el usuario ingresa una clave errónea tres veces, el acceso al sistema es bloqueado.	
CRITERIOS DE ACEPTACIÓN: El cliente entra al sistema o no	

Ph.D. Franklin Parrales 15/12/2021 28

28

Diseño y Arquitectura de Software

Especificación de Casos de Uso

IDENTIFICADOR CASO DE USO: CU-2	NOMBRE: Consultar Saldos de Cuentas Corrientes
COMPLEJIDAD: Baja	PRIORIDAD: Alta
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO	
ACTORES: Cliente	
CASOS DE USO ASOCIADOS: NINGUNO	
DESCRIPCIÓN: El cliente visualiza los saldos de sus cuentas corrientes.	
NOTAS: - El cliente solo podrá consultar sus cuentas corrientes. - Se mostrarán los saldos: disponible, por confirmar y total.	
CRITERIOS DE ACEPTACIÓN: El cliente visualiza los saldos o no	

Ph.D. Franklin Parrales 15/12/2021 29

29

Diseño y Arquitectura de Software

Especificación de Casos de Uso

IDENTIFICADOR CASO DE USO: CU-3	NOMBRE: Pagar Planilla Telefónica
COMPLEJIDAD: Baja	PRIORIDAD: Alta
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO	
ACTORES: Cliente	
CASOS DE USO ASOCIADOS: NINGUNO	
DESCRIPCIÓN: El cliente paga la planilla telefónica usando su cuenta corriente.	
NOTAS:	
- El cliente deberá tener una cuenta corriente en el banco para poder efectuar el pago de la planilla telefónica. - Se podrán realizar pagos de teléfonos propios y de terceros.	
CRITERIOS DE ACEPTACIÓN: El cliente realiza los pagos o no	

Ph.D. Franklin Parrales 15/12/2021 30

30

Diseño y Arquitectura de Software

Lista de Escenarios

- Caso de Uso 1: Ingresar al Sistema
 - Escenario 1.1: Ingreso exitoso al sistema
 - Escenario 1.2: Ingreso no exitoso porque cliente ingreso identificación errónea.
 - Escenario 1.3: Ingreso no exitoso porque cliente ingreso clave errónea.
 - Escenario 1.4: Ingreso no exitoso por bloqueo de cliente en el sistema.
 - Escenario 1.5: Ingreso no exitoso por fallas técnicas.

Ph.D. Franklin Parrales 15/12/2021 31

31

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO: CU-1	NOMBRE: Ingreso al sistema
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-1.1	DESCRIPCIÓN: <i>Ingreso al sistema de forma exitosa.</i> SUPOSICIONES/ASUNCIONES: - La identificación ingresada es correcta. - La clave ingresada es correcta. RESULTADOS: - El cliente ingresa al sistema y recibe el mensaje de Bienvenida. - Se guarda un registro del ingreso del cliente al sistema.
ES-1.2	DESCRIPCIÓN: <i>Ingreso no exitoso porque cliente ingresó identificación errónea.</i> SUPOSICIONES/ASUNCIONES: - La identificación ingresada es incorrecta. RESULTADOS: - El cliente no puede ingresar al sistema y recibe un mensaje que le indica que ha ingresado una identificación incorrecta. - Se guarda el registro de intento fallido de ingreso al sistema.

Ph.D. Franklin Parrales 15/12/2021 32

32

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO: CU-1	NOMBRE: Ingreso al sistema
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-1.3	DESCRIPCIÓN: <i>Ingreso no exitoso porque cliente ingreso clave errónea.</i> SUPOSICIONES/ASUNCIONES: - La identificación ingresada es correcta. - La clave ingresada es incorrecta. RESULTADOS: - El cliente no puede ingresar al sistema y recibe un mensaje que le indica que ha ingresado una clave incorrecta. - Se guarda un registro del intento fallido de ingreso al sistema.
ES-1.4	DESCRIPCIÓN: <i>Ingreso no exitoso por bloqueo de cliente en el sistema.</i> SUPOSICIONES/ASUNCIONES: - El cliente tiene bloqueado el ingreso al sistema porque ingresó erróneamente la clave tres veces. RESULTADOS: - El cliente no puede ingresar al sistema y recibe un mensaje que le indica que ha su ingreso al sistema está bloqueado. - Se guarda el registro de intento fallido de ingreso al sistema.

Ph.D. Franklin Parrales 15/12/2021 33

33

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO: CU-1	NOMBRE: Ingreso al sistema
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-1.5	DESCRIPCIÓN: <i>Ingreso no exitoso por fallas técnicas.</i> SUPOSICIONES/ASUNCIONES: - Se tiene alguna falla técnica en el sistema. RESULTADOS: - El cliente no puede ingresar al sistema por fallas técnicas. - Se guarda el registro de intento fallido de ingreso al sistema.

Ph.D. Franklin Parrales 15/12/2021 34

34

Diseño y Arquitectura de Software

Lista de Escenarios

- Caso de Uso 2: Consultar Saldo de Cuentas Corrientes
 - Escenario 2.1: Consulta exitosa de Saldo de Cuentas Corrientes
 - Escenario 2.2: Consulta no exitosa puesto que el usuario no tiene cuentas corrientes titulares.
 - Escenario 2.3: Consulta no exitosa por fallas técnicas.

Ph.D. Franklin Parrales 15/12/2021 35

35

Diseño y Arquitectura de Software	
IDENTIFICADOR CASO DE USO: CU-2	NOMBRE: Consultar Saldos de Cuentas Corrientes
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-2.1	DESCRIPCIÓN: Consulta exitosa de Saldos de Cuentas Corrientes. SUPOSICIONES/ASUNCIONES: - El cliente posee cuentas corrientes titulares. RESULTADOS: - El cliente visualiza los saldos de sus cuentas corrientes. - Se guarda un registro de la consulta exitosa de saldos de cuentas corrientes.
ES-2.2	DESCRIPCIÓN: Consulta no exitosa puesto que el usuario no tiene cuentas corrientes titulares. SUPOSICIONES/ASUNCIONES: - El cliente no posee cuentas corrientes titulares. RESULTADOS: - El cliente recibe el mensaje de que no tiene cuentas corrientes titulares en el banco. - Se guarda un registro de la consulta no exitosa.

Ph.D. Franklin Parrales

15/12/2021

36

36

Diseño y Arquitectura de Software	
IDENTIFICADOR CASO DE USO: CU-2	NOMBRE: Consultar Saldos de Cuentas Corrientes
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-2.3	DESCRIPCIÓN: Consulta no exitosa por fallas técnicas. SUPOSICIONES/ASUNCIONES: - Se tiene alguna falla técnica en el sistema. RESULTADOS: - El cliente no puede visualizar el saldo de sus cuentas corrientes por fallas técnicas. - Se guarda el registro de consulta no exitosa.

Ph.D. Franklin Parrales

15/12/2021

37

37

Diseño y Arquitectura de Software	
Lista de Escenarios	
- Caso de Uso 3: Pagar Planilla Telefónica - Escenario 3.1: El pago de la planilla telefónica se realizó exitosamente. - Escenario 3.2: El pago de la planilla telefónica no fue exitoso, debido a que el número de teléfono ingresado no existe. - Escenario 3.3: El pago de la planilla telefónica no fue exitoso, debido a que la cuenta no tenía saldo suficiente. - Escenario 3.4: El pago de la planilla telefónica no fue exitoso por fallas técnicas.	

Ph.D. Franklin Parrales

15/12/2021

38

38

Diseño y Arquitectura de Software	
IDENTIFICADOR CASO DE USO: CU-3	NOMBRE: Pagar Planilla Telefónica
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-3.1	DESCRIPCIÓN: El pago de la planilla telefónica se realizó exitosamente. SUPOSICIONES/ASUNCIONES: - El cliente ingresó un número telefónico válido. - El saldo de la cuenta corriente seleccionada para el pago es mayor o igual al valor de la planilla telefónica.. RESULTADOS: - El pago de la planilla telefónica se realiza exitosamente. - Se guarda un registro de la transacción exitosa de pago.
ES-3.2	DESCRIPCIÓN: El pago de la planilla telefónica no fue exitoso, debido a que el número de teléfono ingresado no existe. SUPOSICIONES/ASUNCIONES: - El número de teléfono ingresado no existe. RESULTADOS: - El pago de la planilla telefónica no se pudo realizar y el cliente recibe el mensaje que le indica, que el número telefónico ingresado no existe. - Se registra el intento fallido del pago de la planilla telefónica..

Ph.D. Franklin Parrales

15/12/2021

39

39

Diseño y Arquitectura de Software	
IDENTIFICADOR CASO DE USO: CU-3	NOMBRE: Pagar Planilla Telefónica
.... En la plantilla, los escenarios se especifican en el mismo caso de uso.	
ESCENARIOS:	
ES-3.3	DESCRIPCIÓN: El pago de la planilla telefónica no fue exitoso, debido a que la cuenta no tenía saldo suficiente. SUPOSICIONES/ASUNCIONES: - El número de teléfono ingresado es válido. - El saldo de la cuenta era inferior al valor de la planilla telefónica. RESULTADOS: - El pago de la planilla telefónica no se pudo realizar y el cliente recibe el mensaje que le indica, que el saldo de la cuenta es insuficiente para realizar esa transacción. - Se registra el intento fallido de pago de la planilla telefónica.
ES-3.4	DESCRIPCIÓN: El pago de la planilla telefónica no fue exitoso por fallas técnicas. SUPOSICIONES/ASUNCIONES: - Hubo una falla técnica en el sistema. RESULTADOS: - No se puede realizar el pago por razones técnicas. - Se guarda un registro del pago fallido de planilla telefónica.

Ph.D. Franklin Parrales

15/12/2021

40

40

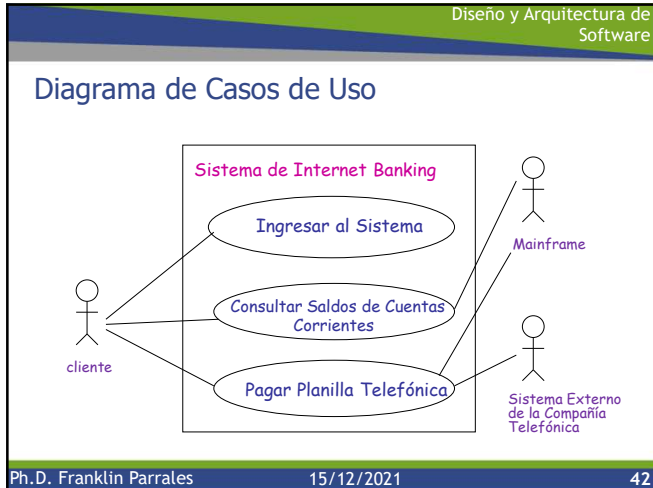
Diseño y Arquitectura de Software	
Tareas	
- Identificar los actores, especificarlos formalmente. - Listar los casos de uso y especificarlos formalmente. - Listar los escenarios y especificarlos formalmente. - Realizar el Diagrama de Casos de Uso	

Ph.D. Franklin Parrales

15/12/2021

41

41



42

Diseño y Arquitectura de Software

Tarea 1

Se desea realizar un sistema web para un hotel.

- De las habitaciones se desea conocer el número, el piso en el que se encuentra y el costo.
- De los clientes se desea conocer su identificación, nombre, apellido, dirección, teléfono (los clientes se encargan de ingresar estos datos en el sistema).
- Se requieren guardar los siguientes datos cada vez que un cliente realiza una reserva: fecha inicial y final de estadía, para reservar no es necesario hacer un pago.
- El pago del hospedaje lo realiza el cliente a través del sistema con su tarjeta de crédito.
- Solamente la recepcionista del hotel puede visualizar un listado de las habitaciones disponibles, y consultar el precio de una habitación de acuerdo al tipo (simple, doble y triple).
- Tanto la recepcionista como los clientes pueden hacer una consulta del costo total del hospedaje, ingresando la identificación del cliente, el tipo de habitación y el número de noches, además ambos pueden realizar reservas de habitaciones.
- Solamente el administrador podrá ingresar los datos de las habitaciones, eliminar una reserva, y obtener un reporte de las ganancias del hotel durante un mes específico.

Para este sistema, realizar el diagrama de casos de uso.

Ph.D. Franklin Parrales 15/12/2021 43

43

Diseño y Arquitectura de Software

Tarea 2

Se requiere desarrollar un sistema web, para la búsqueda de trabajo. A este sistema podrán ingresar las personas que busquen empleo y las empresas que soliciten empleados. Para poder acceder al sistema, los usuarios tienen que tener un identificador (cédula o ruc) y una contraseña. La persona que busca trabajo puede ingresar sus datos personales: apellido, dirección, email, profesión, área de interés y curriculum vitae, además podrá visualizar un listado de las oportunidades de trabajo. El sistema permitirá ingresar los siguientes datos de las empresas: ruc, razón social, dirección, email, y sitio web, además se podrán ingresar: las ofertas de trabajo, también se podrán visualizar un listado de los posibles candidatos para las vacantes en la empresa. Los usuarios pueden realizar el pago del servicio de búsqueda de trabajo, a través de este sistema con tarjeta de crédito. Para este sistema se pide:

- 1) Identifique a los actores y especifíquelos formalmente.
- 2) Listar los casos de uso y especifique formalmente 3 de ellos.
- 3) Listar los escenarios para los 3 casos de uso seleccionados y especificar formalmente 2 escenarios por cada caso de uso seleccionado.
- 4) Diagrama de Casos de Uso del Sistema.

Ph.D. Franklin Parrales 15/12/2021 44

44

Diseño y Arquitectura de Software

Tarea 3

Es necesario implementar un sistema para una biblioteca. El sistema debe guardar información de los libros y de los autores. En el sistema los registros de los libros y autores son ingresados por un digitador. Además, en el sistema se registran los préstamos y devoluciones de libros; donde tales operaciones son efectuadas por la bibliotecaria. Para tal sistema efectúe lo siguiente:

- Identifique los actores
- Realice la descripción de los actores.
- Liste los casos de uso.
- Realice la descripción de los casos de uso
- Realice la especificación de los escenarios
- Diagrama de casos de uso

Ph.D. Franklin Parrales 15/12/2021 45

45

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO: CU-1		NOMBRE: Ingresar al Sistema	
COMPLEJIDAD: Baja		PRIORIDAD: Alta	
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO			
ACTORES: Cliente			
CASOS DE USO ASOCIADOS: NINGUNO			
PROPOSITO: Los clientes ingresarán al sistema con su identificador y clave personal.			
DATOS DE ENTRADA: Identificador y clave personal.			
Id.	Nombre	Id.	Nombre
1	Identificador	✓	Único
		✓	Tipo de Dato ALFANUMÉRICO
		✓	Obligatorio
		✓	Ingreso manual
DATOS DE SALIDA:			
Id.	Nombre	Id.	Nombre
	Resultado	✓	Formato: "Datos ingresados".
		✓	Obligatorio/Opcional
PRECONDICIÓN: «Hechos particulares para el caso de uso que se han de cumplir para que el flujo normal de trabajo se pueda llevar a cabo, incluyendo la ubicación del usuario en el sistema antes de iniciar la ejecución de este caso de uso.»			
SECUENCIA NORMAL DE TRABAJO: SN-1 «Nombre Secuencia (flujo) Normal de Trabajo» «Corresponde a la ejecución normal y exitosa del caso de uso.»			
• Cuando se haga referencia a atributos de entrada o salida en los flujos (normal o alternativo) se debe hacer referencia a su identificador (Id). • Las opciones que muestra el sistema deben ir en negrilla y no se deben colocar términos técnicos. • Las mensajes de confirmación o alerta deben ir entre comillas dobles y con letra cursiva. • Solo debe existir un Flujo Normal en cada caso de uso. • Los verbos utilizados en los pasos deben ser en infinitivo Ejem: Seleccionar, Mostrar, Ejecutar, Listar, etc. • Si el caso de uso tiene varios flujos de trabajo alternativos, es necesario colocar Nombre a cada uno de ellos.			

Ph.D. Franklin Parrales 15/12/2021 46

46

Diseño y Arquitectura de Software

IDENTIFICADOR CASO DE USO: CU-1		NOMBRE: Ingresar al Sistema	
COMPLEJIDAD: Baja		PRIORIDAD: Alta	
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO			
ACTORES: Cliente			
CASOS DE USO ASOCIADOS: NINGUNO			
DESCRIPCIÓN: Los clientes ingresarán al sistema con su identificador (cédula) y clave personal.			
DATOS DE ENTRADA: Identificador (cédula) y clave personal.			
VALOR MEDIBLE: Usuario ingresa o no al sistema.			
SECUENCIA NORMAL DE TRABAJO: SN-1 Ingreso exitoso al sistema.			
• Ingresar el correcto identificador y contraseña. • Visualizar el mensaje "Bienvenido".			

Ph.D. Franklin Parrales 15/12/2021 47

47

Diseño y Arquitectura de Software											
IDENTIFICADOR CASO DE USO:											
CU-1		NOMBRE:									
COMPLEJIDAD: Baja		Ingresar al Sistema									
REQUERIMIENTO FUNCIONAL ASOCIADO: NINGUNO		PRIORIDAD: Alta									
ACTORES: Cliente											
CASOS DE USO ASOCIADOS: NINGUNO											
PROPÓSITO: Los clientes ingresarán al sistema con su identificador y clave personal.											
DATOS DE ENTRADA: Identificador y clave personal.											
<table><tr><th>Id.</th><th>Nombre</th><th>Id.</th></tr><tr><td>DE1</td><td>Identificador</td><td>✓ Único ✓ Tipo de Dato ALFANUMÉRICO ✓ Obligatorio ✓ Ingreso manual</td></tr><tr><td>DE2</td><td>Contraseña</td><td>✓ Tipo de Dato ALFANUMÉRICO ✓ Obligatorio ✓ Ingreso manual</td></tr></table>			Id.	Nombre	Id.	DE1	Identificador	✓ Único ✓ Tipo de Dato ALFANUMÉRICO ✓ Obligatorio ✓ Ingreso manual	DE2	Contraseña	✓ Tipo de Dato ALFANUMÉRICO ✓ Obligatorio ✓ Ingreso manual
Id.	Nombre	Id.									
DE1	Identificador	✓ Único ✓ Tipo de Dato ALFANUMÉRICO ✓ Obligatorio ✓ Ingreso manual									
DE2	Contraseña	✓ Tipo de Dato ALFANUMÉRICO ✓ Obligatorio ✓ Ingreso manual									
DATOS DE SALIDA:											
<table><tr><th>Id.</th><th>Nombre</th><th>Id.</th></tr><tr><td>DS1</td><td>Resultado</td><td>✓ Formato: "Bienvenido", "Datos incorrectos", "Cuenta bloqueada". ✓ Obligatorio</td></tr></table>			Id.	Nombre	Id.	DS1	Resultado	✓ Formato: "Bienvenido", "Datos incorrectos", "Cuenta bloqueada". ✓ Obligatorio			
Id.	Nombre	Id.									
DS1	Resultado	✓ Formato: "Bienvenido", "Datos incorrectos", "Cuenta bloqueada". ✓ Obligatorio									
PRECONDICIÓN:											
SECUENCIA NORMAL DE TRABAJO: SN-1 Ingreso exitoso al sistema.											
• Ingresar el correcto identificador (DE1) y contraseña (DE2).											
• Visualizar el mensaje "Bienvenido".											
<table><tr><th>Paso</th><th>Actor</th><th>Sistema</th></tr><tr><td>1</td><td>Ingresar datos de identificador (DE1) y contraseña (DE2).</td><td>Revisión de los datos ingresados. Estos son correctos.</td></tr><tr><td>2</td><td>Recibe mensaje "Bienvenido"</td><td></td></tr></table>			Paso	Actor	Sistema	1	Ingresar datos de identificador (DE1) y contraseña (DE2).	Revisión de los datos ingresados. Estos son correctos.	2	Recibe mensaje "Bienvenido"	
Paso	Actor	Sistema									
1	Ingresar datos de identificador (DE1) y contraseña (DE2).	Revisión de los datos ingresados. Estos son correctos.									
2	Recibe mensaje "Bienvenido"										
POSCONDICIÓN: Ninguna											
Ph.D. Franklin Parrales 15/12/2021 48											

48

Diseño y Arquitectura de Software																							
SECUENCIA ALTERNATIVA: SA-1 Bloqueo del sistema																							
• Ingresar el identificador (DE1) y contraseña (DE2) incorrectamente 3 veces																							
• Visualizar el mensaje "Cuenta bloqueada".																							
<table><tr><th>Paso</th><th>Actor</th><th>Sistema</th></tr><tr><td>1</td><td>Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.</td><td>Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 1 intento fallido.</td></tr><tr><td>2</td><td>Recibe mensaje "Datos incorrectos"</td><td></td></tr><tr><td>3</td><td>Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.</td><td>Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 2 intentos fallidos.</td></tr><tr><td>4</td><td>Recibe mensaje "Datos incorrectos"</td><td></td></tr><tr><td>5</td><td>Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.</td><td>Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 3 intentos fallidos. Bloquea la cuenta del cliente.</td></tr><tr><td>6</td><td>Recibe mensaje "Cuenta bloqueada"</td><td></td></tr></table>			Paso	Actor	Sistema	1	Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.	Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 1 intento fallido.	2	Recibe mensaje "Datos incorrectos"		3	Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.	Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 2 intentos fallidos.	4	Recibe mensaje "Datos incorrectos"		5	Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.	Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 3 intentos fallidos. Bloquea la cuenta del cliente.	6	Recibe mensaje "Cuenta bloqueada"	
Paso	Actor	Sistema																					
1	Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.	Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 1 intento fallido.																					
2	Recibe mensaje "Datos incorrectos"																						
3	Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.	Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 2 intentos fallidos.																					
4	Recibe mensaje "Datos incorrectos"																						
5	Ingresar identificador (DE1) incorrecto y/o contraseña (DE2) incorrecta.	Revisión de los datos ingresados. Estos son incorrectos. Contabiliza 3 intentos fallidos. Bloquea la cuenta del cliente.																					
6	Recibe mensaje "Cuenta bloqueada"																						
Ph.D. Franklin Parrales 15/12/2021 49																							

49