



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Clase 11

# Diagramas del Diseño

IIC2143 - Ingeniería de Software  
Sección 1

Rodrigo Saffie

[rasaffie@uc.cl](mailto:rasaffie@uc.cl)

16 de abril de 2018

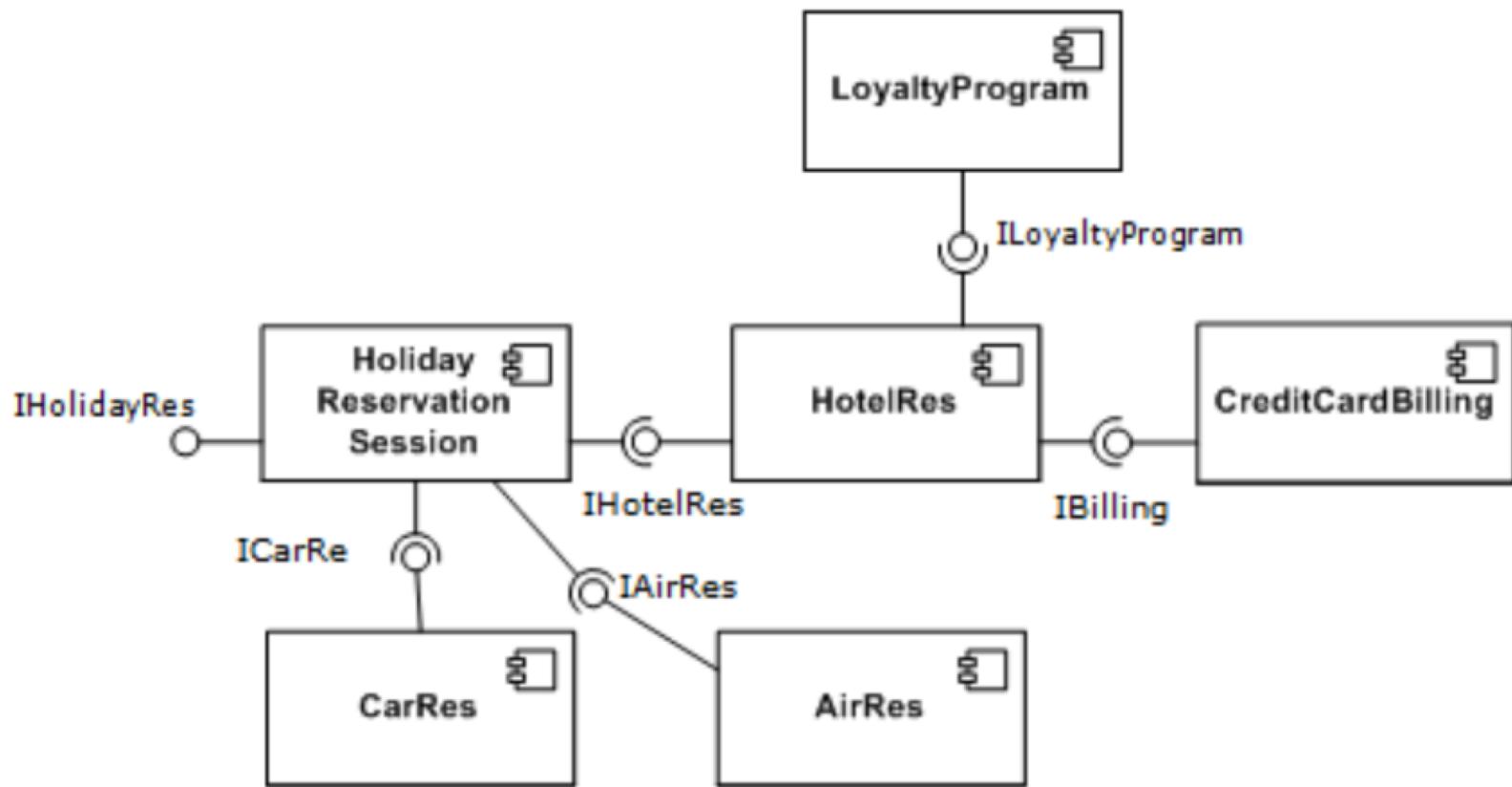
# Diseño de software

## ¿Qué es el diseño de software?

- Un concepto reciente (70 años)
- Traduce los requisitos a especificaciones técnicas
- Se divide en 4 áreas:
  - Diseño de componentes
  - Diseño de arquitectura
  - Diseño de clases/datos
  - Diseño de interfaces

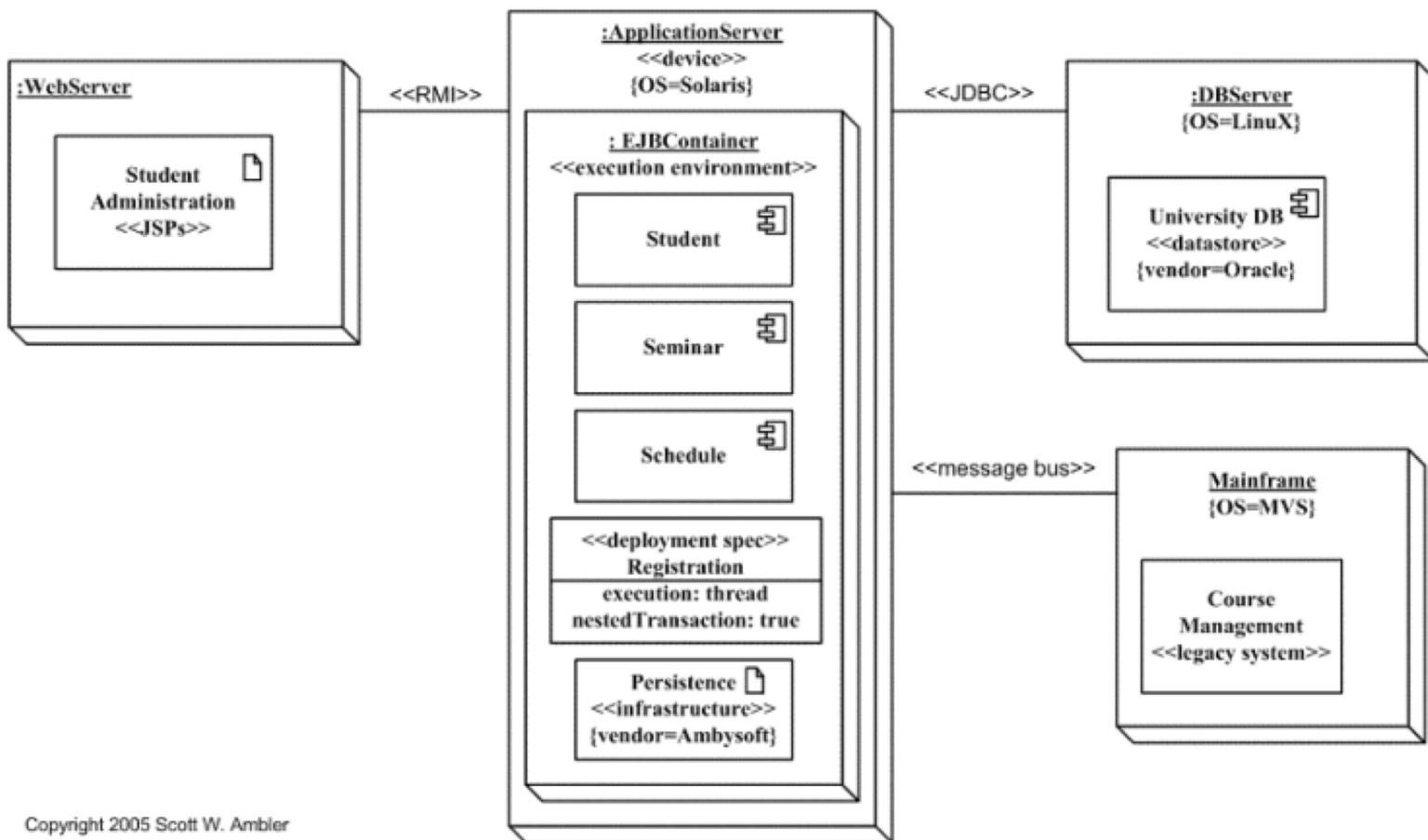
# Diseño de componentes

Descripción de los componentes del sistema



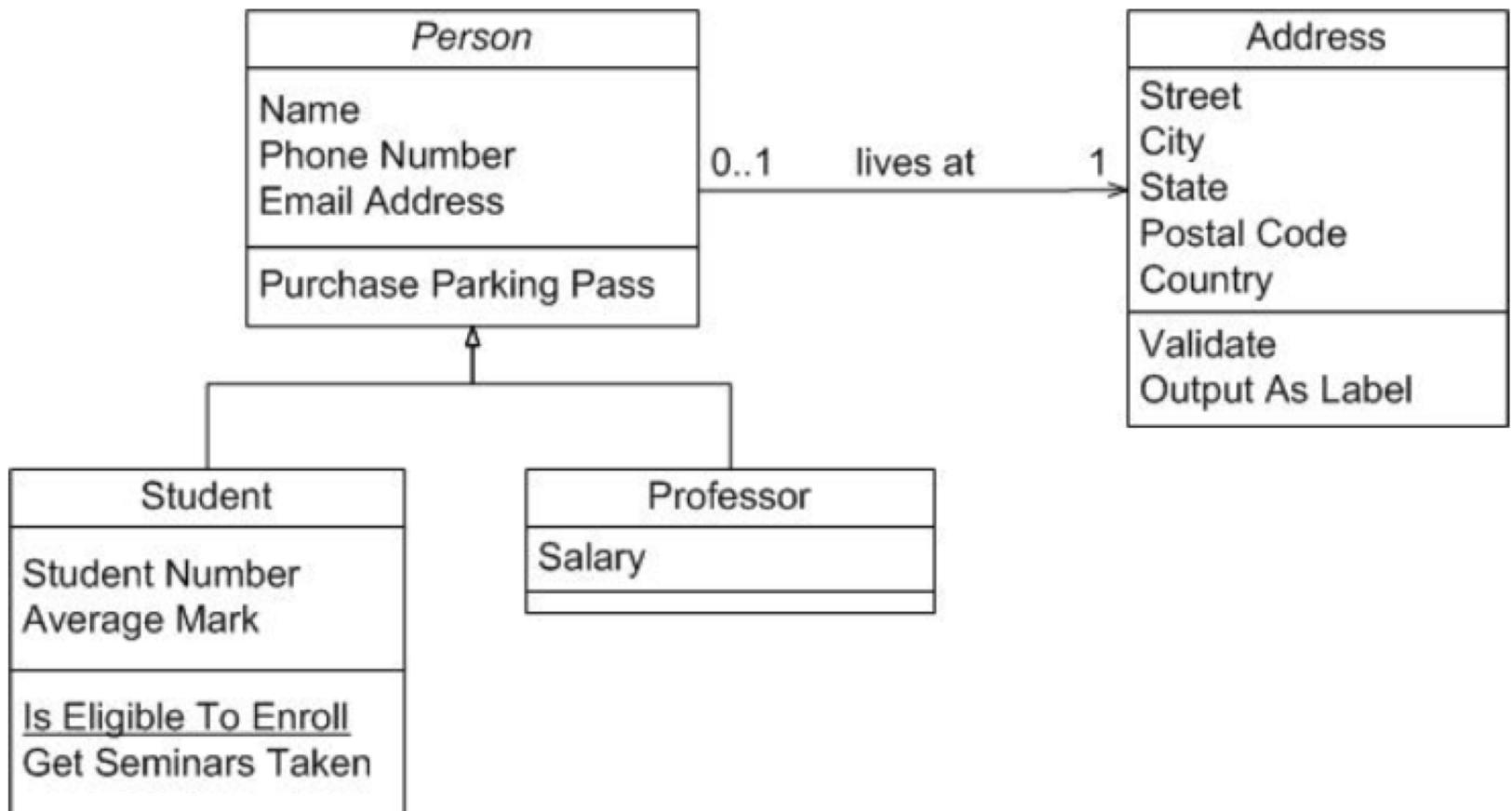
# Diseño de arquitectura

## Relaciones entre los componentes del sistema



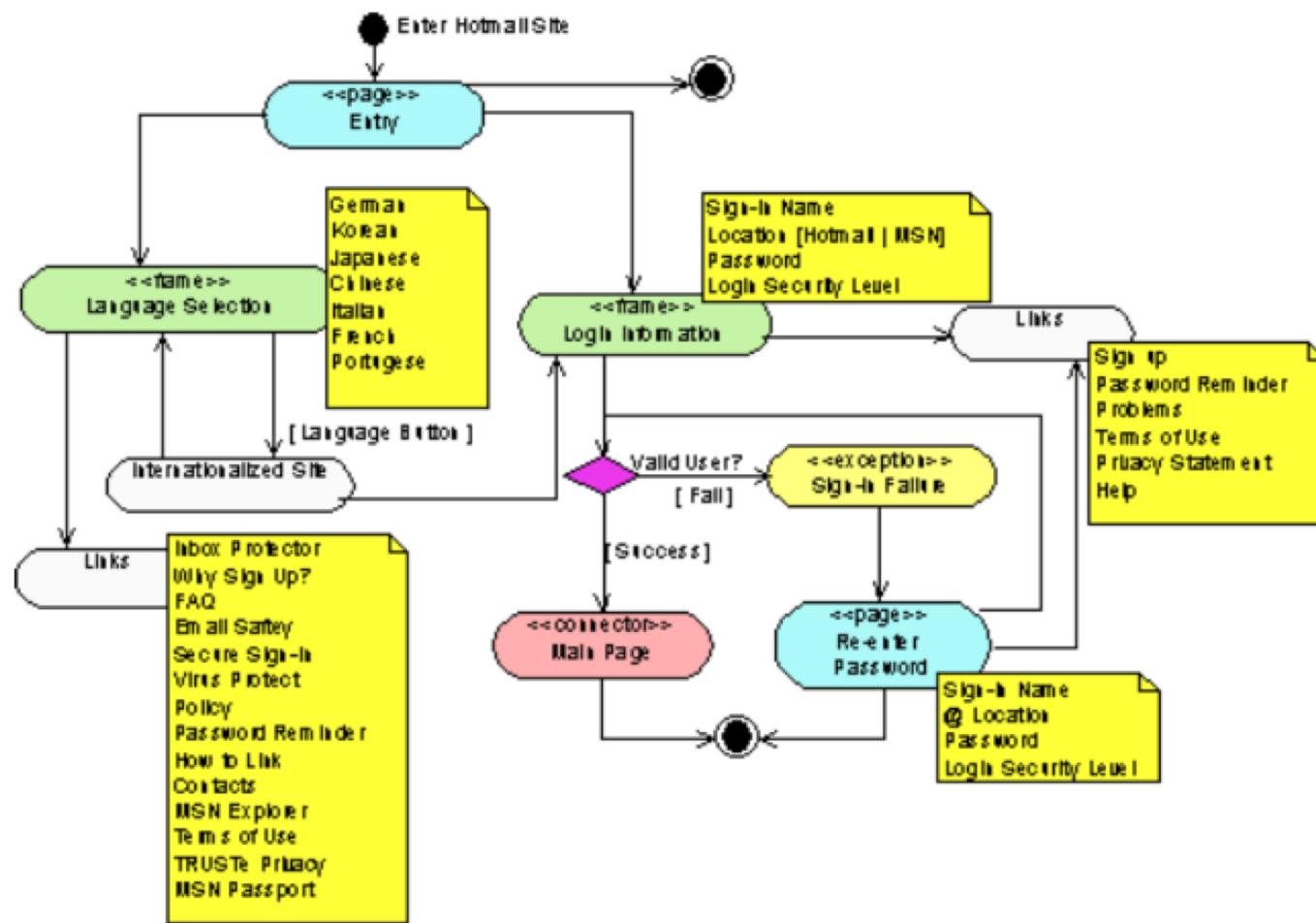
# Diseño de clases/datos

## Esquema de clases y sus relaciones



# Diseño de interfaces

## Comunicación con sistemas externos (y humanos)



# Diseño de software

¿Qué cualidades debe tener la descripción del diseño de un software?

- Sin ambigüedades, preciso y objetivo
- Convenciones claras para los involucrados
- Modificable, con menor costo que la construcción

# Diseño de software

¿Quiénes están interesados en la descripción del diseño?

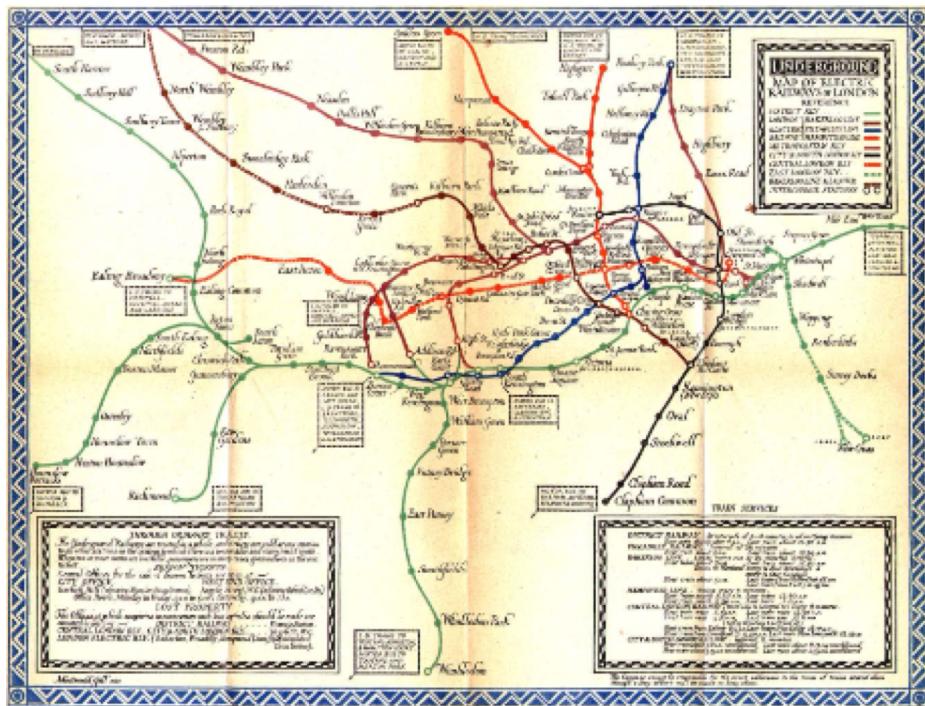
- Desarrolladores
- Líderes de proyectos
- Arquitectos de sistemas
- Analistas
- Clientes
- Usuarios

# Principios del diseño de software

- Abstracción
- Ocultamiento
- Cohesión
- Acoplamiento

# Abstracción

Rescatar información relevante dado un contexto



# Ocultamiento

No exponer información o lógica innecesaria

## Ejemplos

- Servicios Web
- Librerías
- Módulos con modificadores de acceso (*public, private*)

# Cohesión

Medida de cuán relacionados están los datos, responsabilidades y métodos de una clase.

“qué tanto pertenecen los elementos de un modulo a este”

Ejemplo

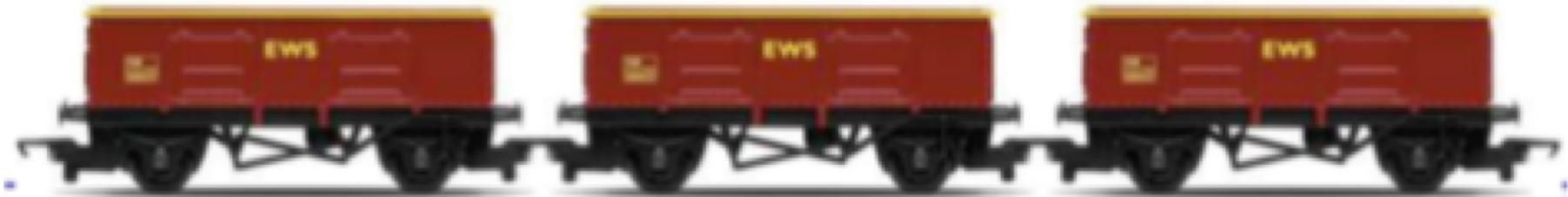
# Acoplamiento

Medida de cuán conectados están dos subsistemas o clases

```
class Warehouse
  def sale_price(item)
    (item.price - item.rebate) * @vat
  end
end
```

# Bajo Acoplamiento, Alta Cohesión

# Cohesión y Acoplamiento en un tren



- Carros acoplados mediante interfaz simple y pequeña
- Separación en carros permite separar contenidos cohesionados

# ¿Por qué alta cohesión y bajo acoplamiento?

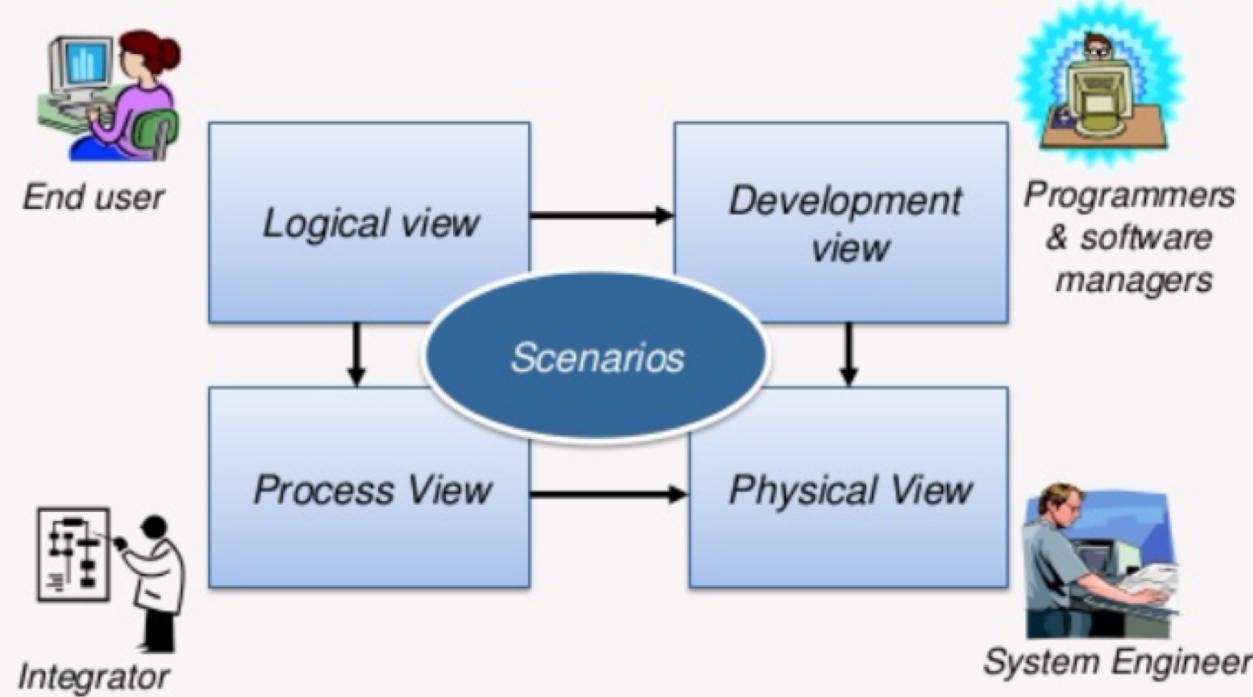
- Interfaces simples
- Comunicación simple
- Cambios afectan a sectores limitados del código
- Aumenta reusabilidad
- Aumenta extensibilidad

# Modelo 4 + 1

- Propuesto por Philippe Kruchten en 1995
- *Framework* para describir la arquitectura de un *software*, basado en el uso de múltiples vistas concurrentes.

# Modelo 4 + 1

## The 4+1 View model



- Describes software architecture using five concurrent views.

# Modelo 4 + 1

- ¿Qué es una vista?

Es una representación del sistema completo, enfocada desde una perspectiva con consideraciones específicas.

# Unified Modeling Language 2.0

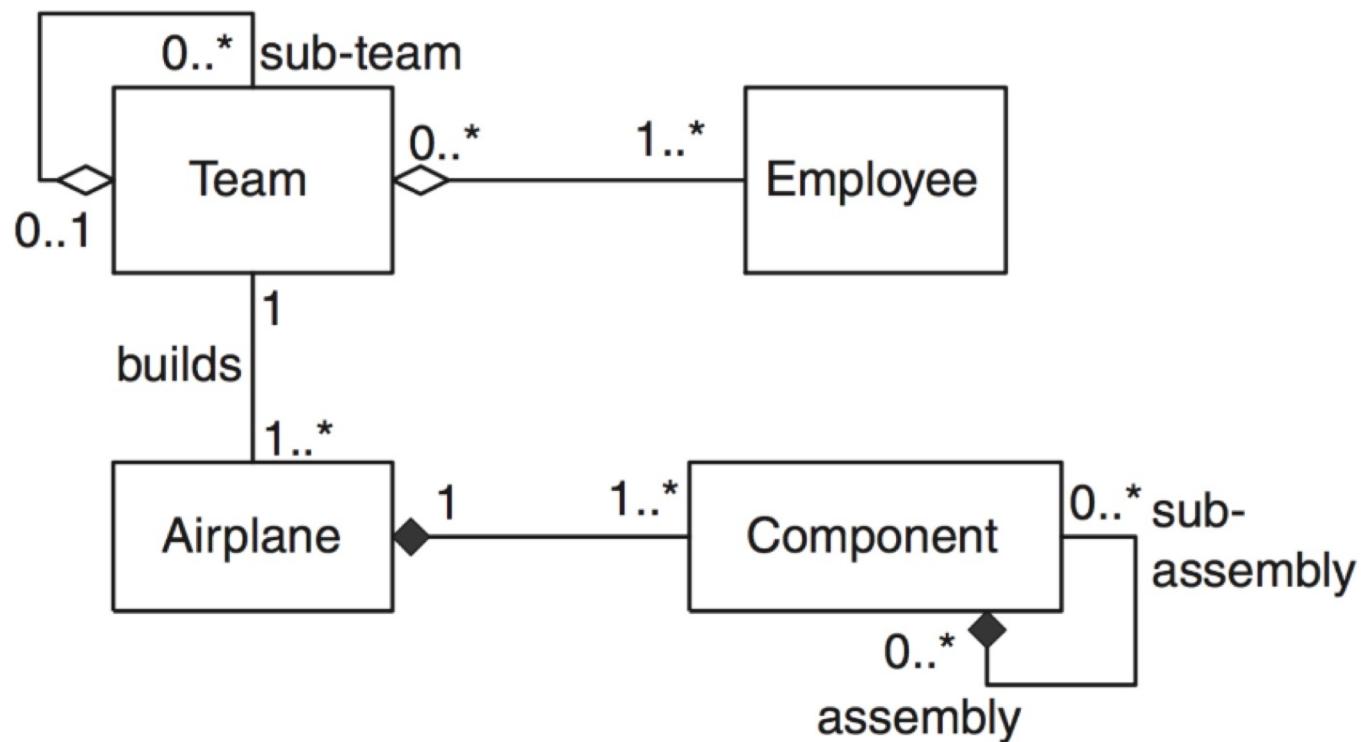
- UML: estándar para especificar un sistema
- 1º versión propuesta en 1997
- Ayuda a entender y explicar un sistema de manera consistente
- *The Elements of UML 2.0 Style*

# Vista lógica

- Interesados: Usuarios finales
- Consideraciones: Requisitos funcionales
  - ¿Qué es lo que el sistema debería proveer a sus usuarios?
- Diagramas:
  - Clases
  - Comunicación
  - Secuencia
  - Estado

# Diagrama de clases

Modela las clases, sus relaciones, operaciones y atributos.



# Diagrama de comunicación

Modela la comunicación y dirección de los mensajes entre clases.

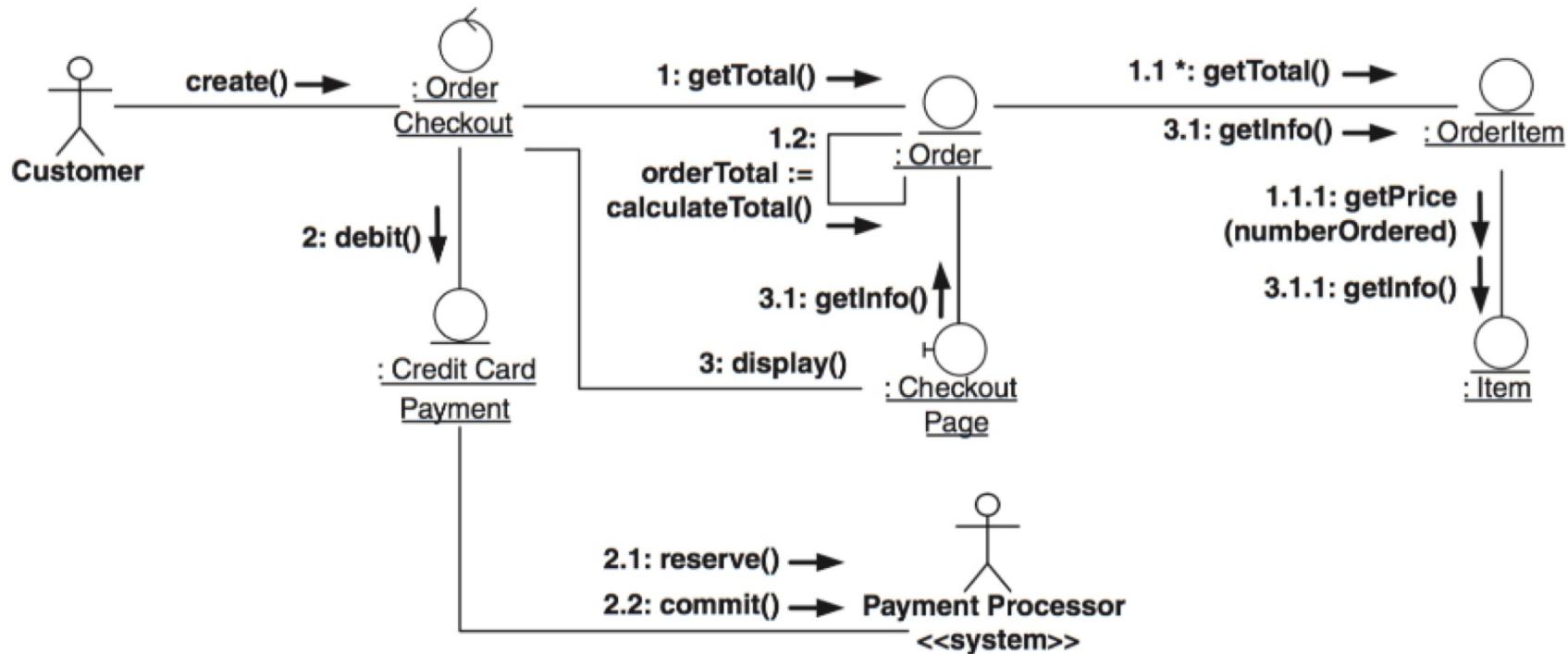
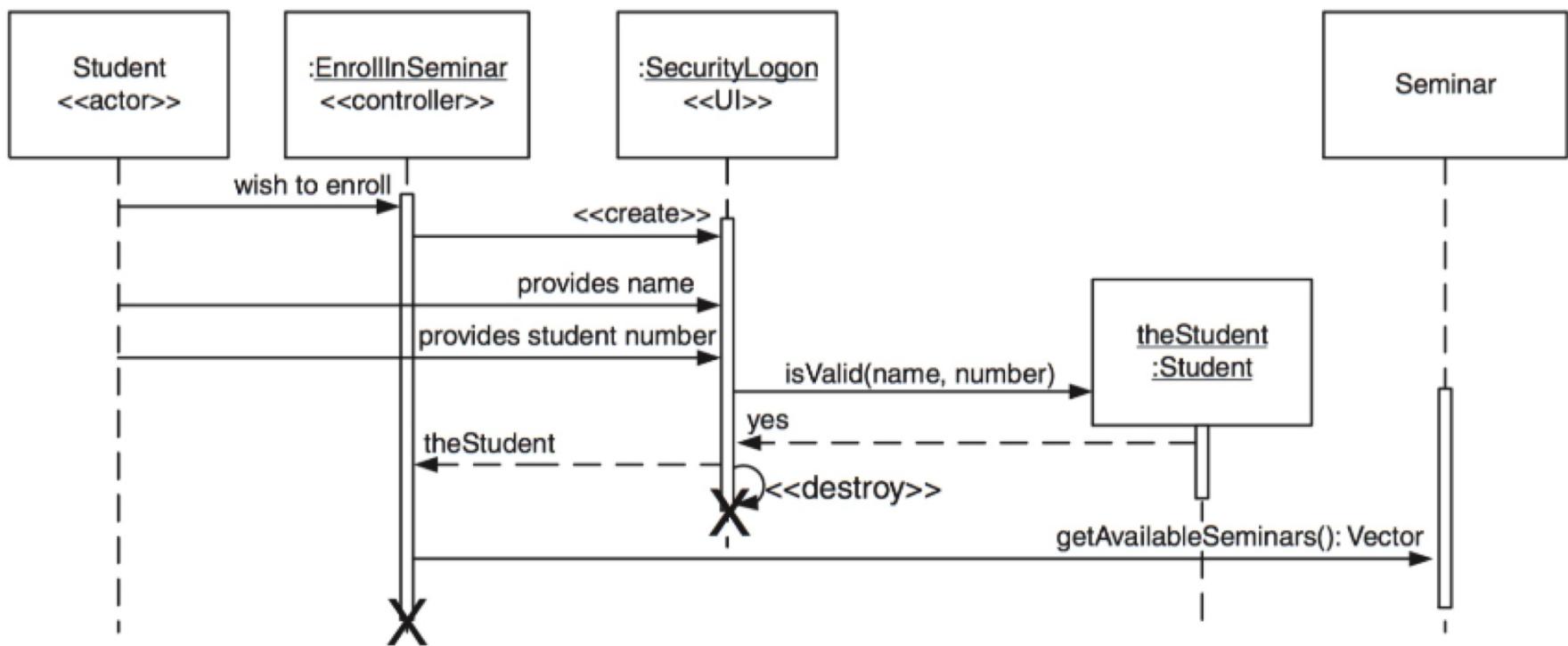


Figure 33. An instance-level UML communication diagram.

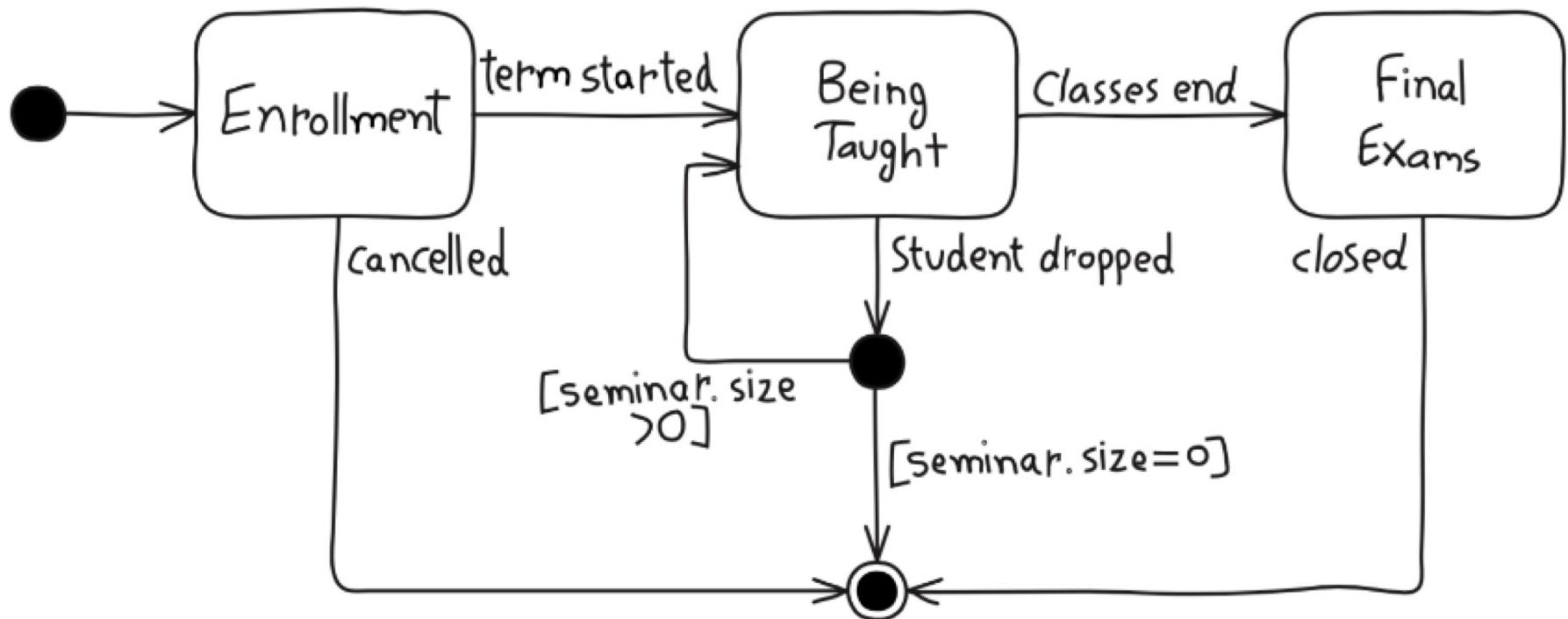
# Diagrama de secuencia

Modela la interacción y orden en que las entidades se relacionan.



# Diagrama de estado

Modela el comportamiento de las entidades dado distintos estados, basado en sus respuestas a eventos.

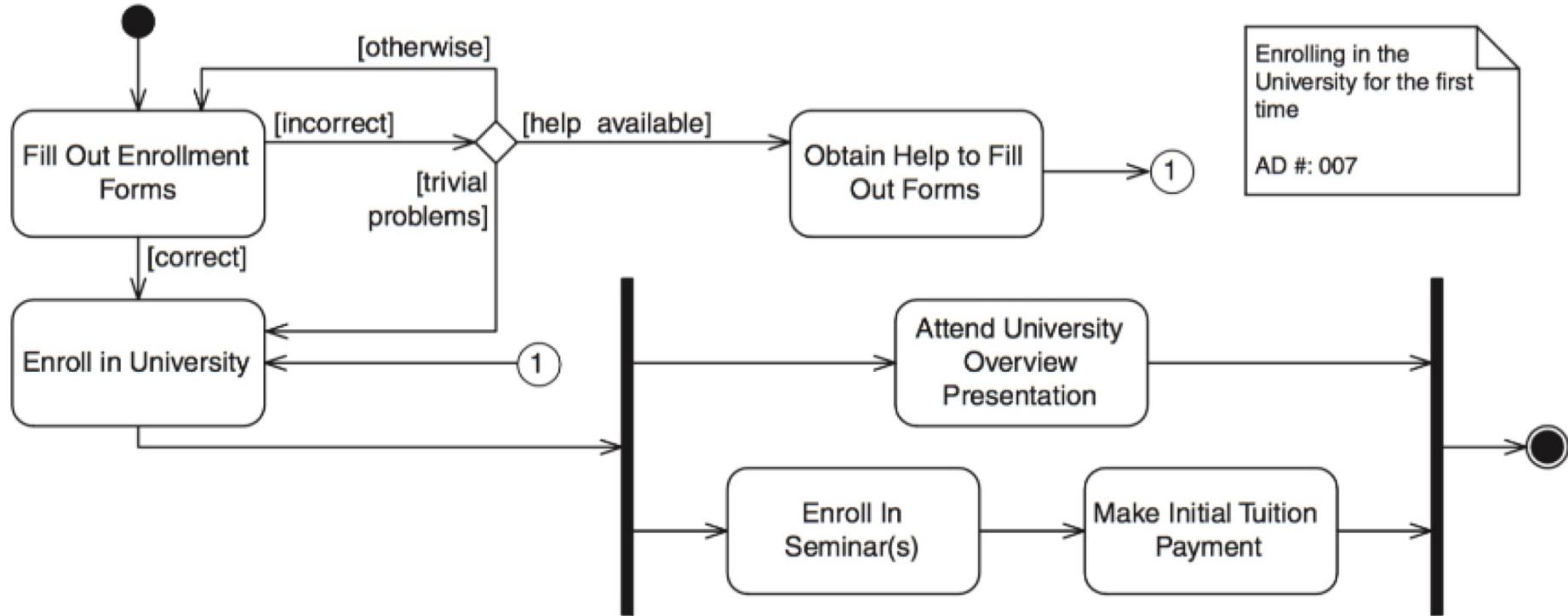


# Vista de procesos

- Interesados: Analistas
- Consideraciones: Requisitos no funcionales (conurrencia, rendimiento, escalabilidad)
  - ¿Qué procesos y reglas tiene el sistema, y cómo interactúan los componentes?
- Diagramas:
  - Actividad

# Diagrama de actividad (o flujo)

Modela procesos en base a reglas.

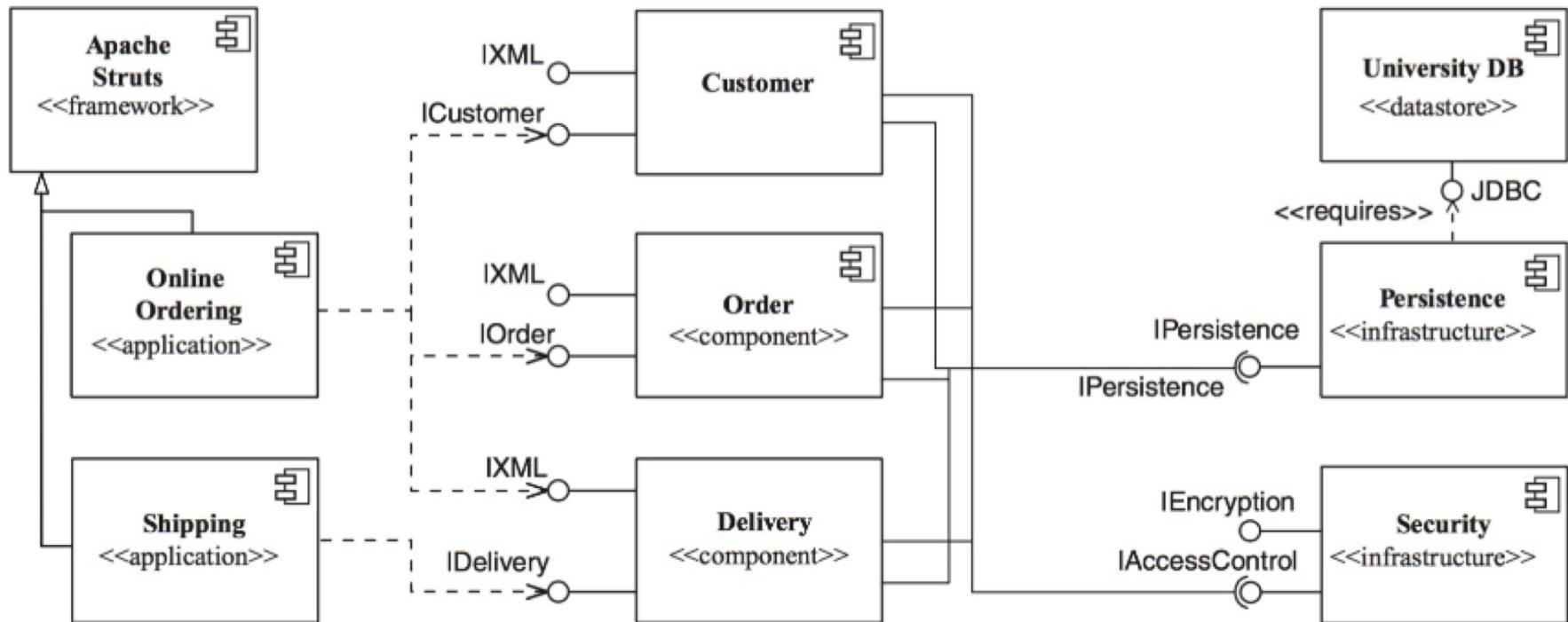


# Vista de implementación

- Interesados: Desarrolladores y líderes de proyectos
- Consideraciones: Organización del *software*
  - ¿Cómo se organiza el *software* del sistema?
- Diagramas:
  - Componentes
  - Paquetes

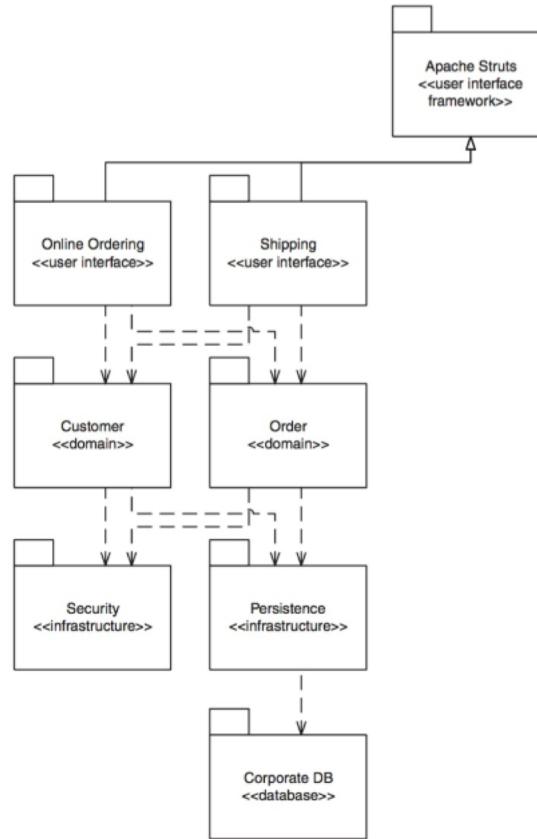
# Diagrama de componentes

Modela la dependencia entre los componentes del *software*.



# Diagrama de paquetes

Modela la dependencia entre los paquetes del *software*.  
Un paquete se puede ver como una carpeta del proyecto.

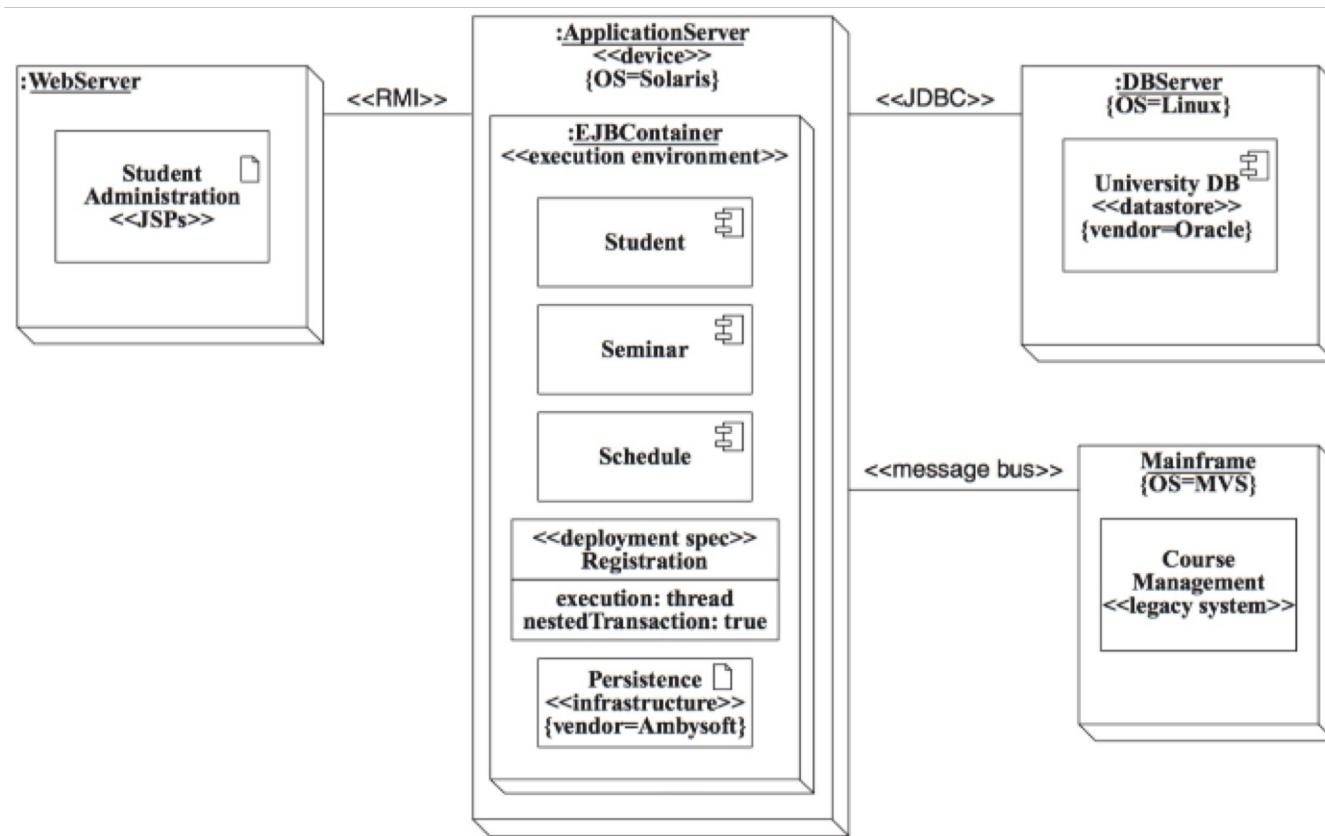


# Vista física

- Interesados: Arquitectos de sistemas
- Consideraciones: Requisitos no funcionales
  - ¿Cómo es el ambiente de ejecución del sistema?
- Diagramas:
  - Despliegue

# Diagrama de despliegue

Modela la configuración en tiempo de ejecución del *hardware*, y el *software* que se ejecuta.

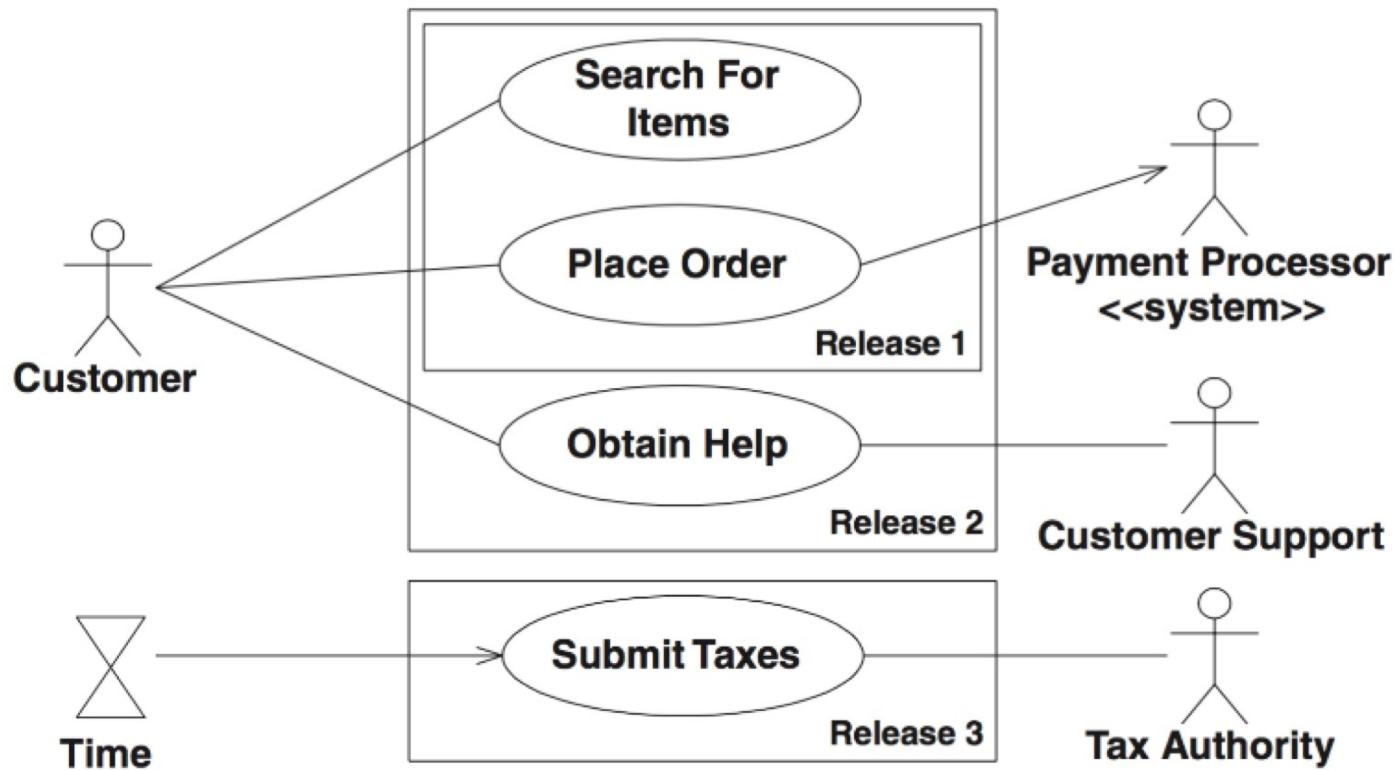


# Vista de escenarios

- Interesados: Todos
- Consideraciones: Consistencia, validez
  - ¿Qué es lo que el sistema debería hacer?
- Diagramas:
  - Casos de uso

# Diagrama de casos de uso

Modela la interacción entre los actores y las funcionalidades de un sistema.





Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Clase 11

# Diagramas del Diseño

IIC2143 - Ingeniería de Software  
Sección 1

Rodrigo Saffie

[rasaffie@uc.cl](mailto:rasaffie@uc.cl)

16 de abril de 2018