IIC2113 - Diseño Detallado de Software

CLASE 1 - MOTIVACIÓN

Pontificia Universidad Católica de Chile
2020-2

Índice

01. Motivación

Repaso de presentación clase 0

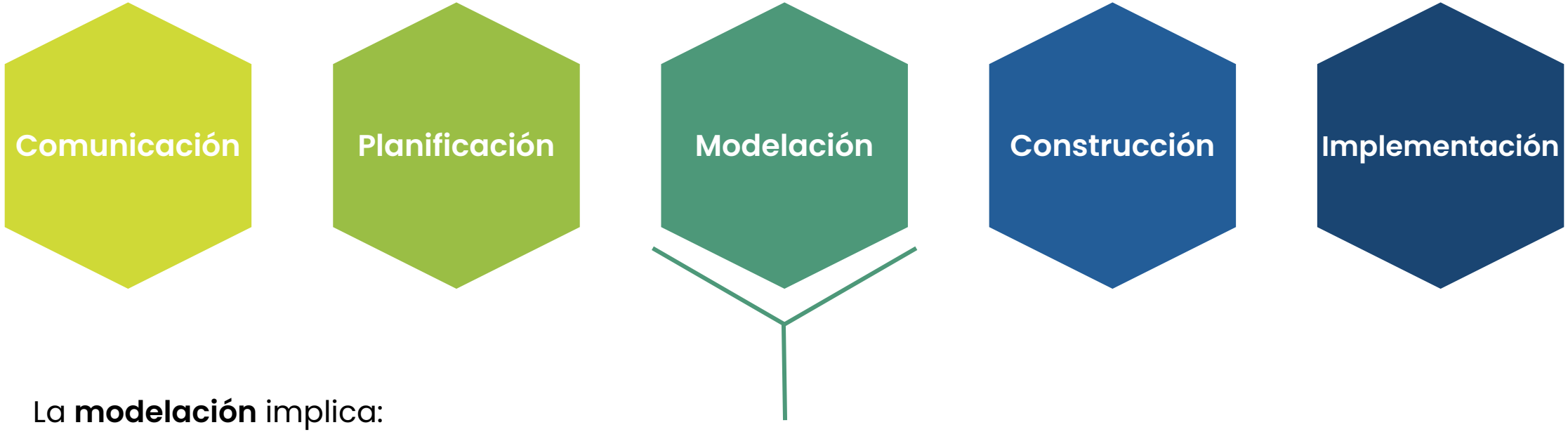
Desarrollo de software

Los procesos de Desarrollo de Software contemplan una serie de actividades comunes. Dependiendo de la metodología el orden de cada etapa podría variar y repetirse.

El **ciclo de vida de un proyecto de Software** varía dependiendo de la metodología (como Ágil, Cascada, Espiral, entre otras). Pero todas mantienen en común algunas actividades como: comunicación, planificación, modelación, construcción e implementación.

Desarrollo de software

Los procesos de Desarrollo de Software contemplan una serie de actividades comunes. Dependiendo de la metodología el orden de cada etapa podría variar y repetirse.



La **modelación** implica:

- Análisis del problema → entender el problema no necesariamente desde una vereda "programática". Es importante comprender el **contexto** y el **negocio**.
- **Diseño de la solución** → determinar y guiar a cómo resolver el problema. Se deciden los **componentes**, **la forma de los datos** y la **arquitectura del sistema**.

¿Cómo es posible **tomar decisiones** sobre un proyecto sin involucrarse de forma integral en el proceso?

¿Cómo es posible **tomar decisiones** sobre un proyecto sin involucrarse de forma integral en el proceso?

↑

Esta es la motivación tras este curso

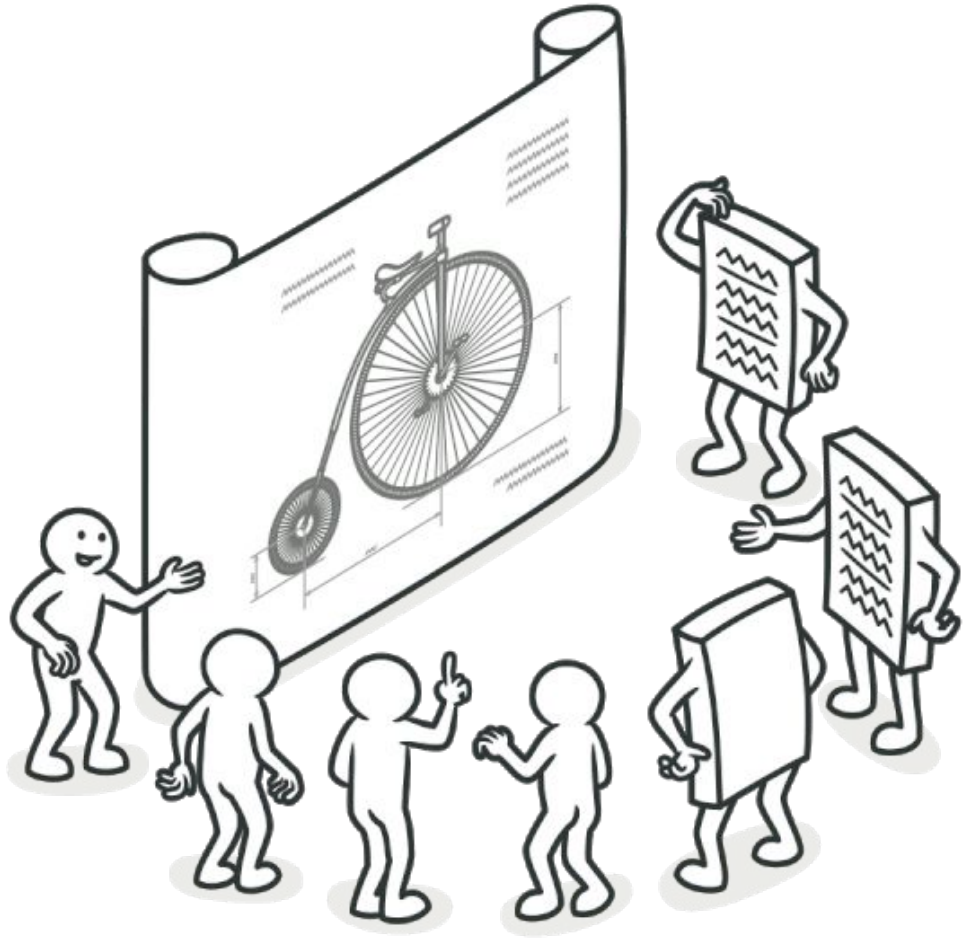
¿De qué trata el curso?

Entregarles las herramientas de forma integral para poder diseñar de forma detallada software que resuelva un problema y que perdure en el tiempo.

1

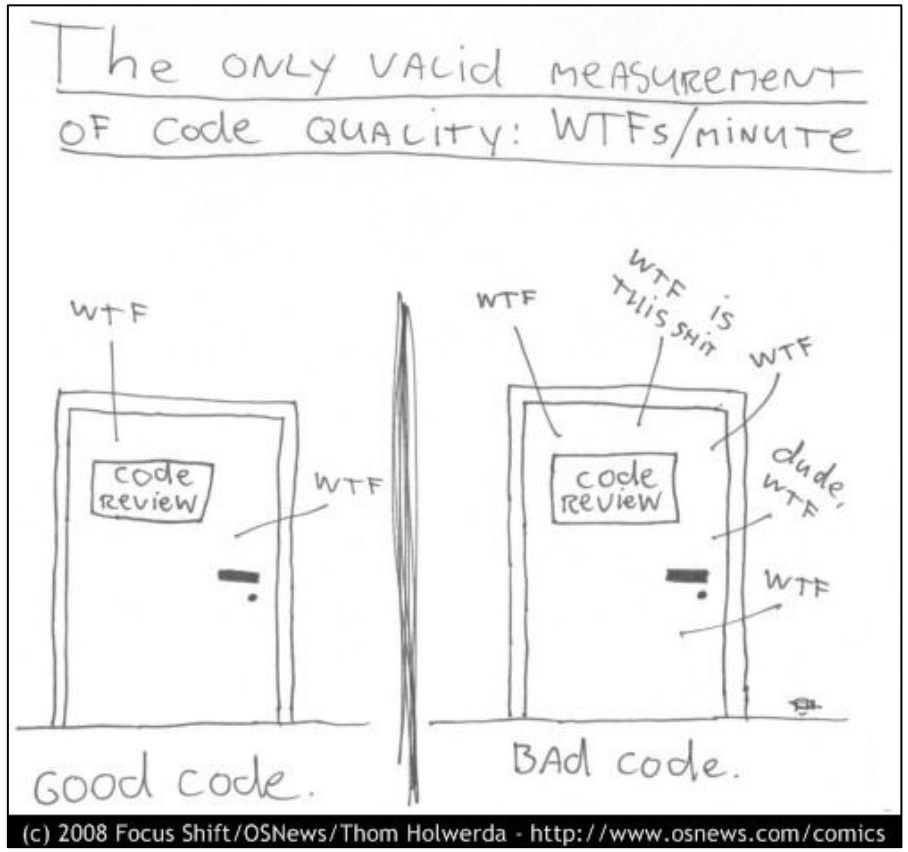
Existen problemas no tan complejos que pueden prescindir del diseño. Sin embargo, los problemas en la vida real suelen ser complejos.

Cuando tenemos que trabajar en un área compleja tenemos que comunicar decisiones y orientar el diseño a las cualidades de la solución (extensibilidad, *performance*, requisitos no funcionales, etc.-)



1: Ilustración de Dmitry Zhart, sitio web de https://refactoring.guru/.

2



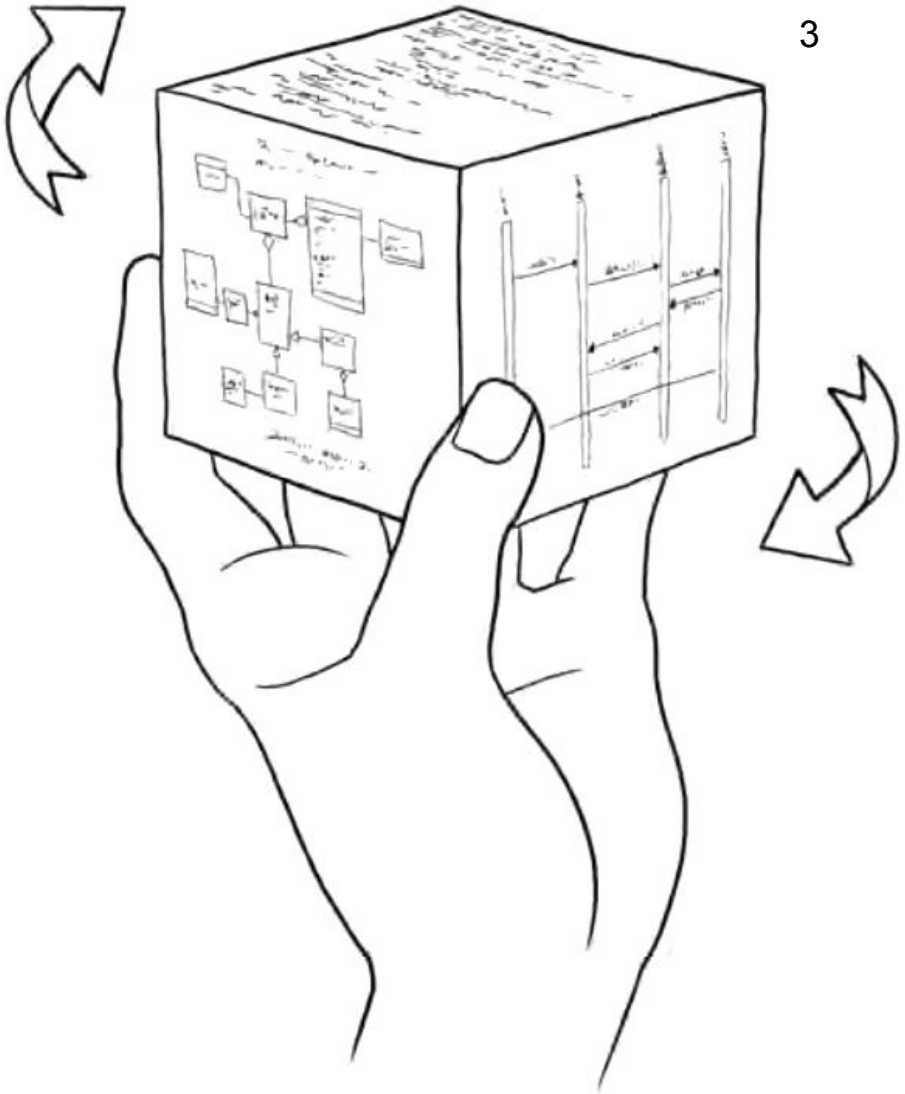
Estudiaremos cómo lograr código entendible y cuyo diseño permita extensibilidad de la solución.

Entenderemos las diferencias entre un "**buen código**" y un "**mal código**".

2: Ilustración de Thom Holwerda, sitio web de https://www.osnews.com/comics/.

Diseñar en distintas notaciones permite un mejor entendimiento para las distintas áreas involucradas.

Los distintos diagramas son **herramientas** que facilitan el trabajo de **comunicar** el problema y la solución.



3

3: Ilustración de Yen Quach. Libro *Software Design Decoded: 66 Ways Experts Think.*

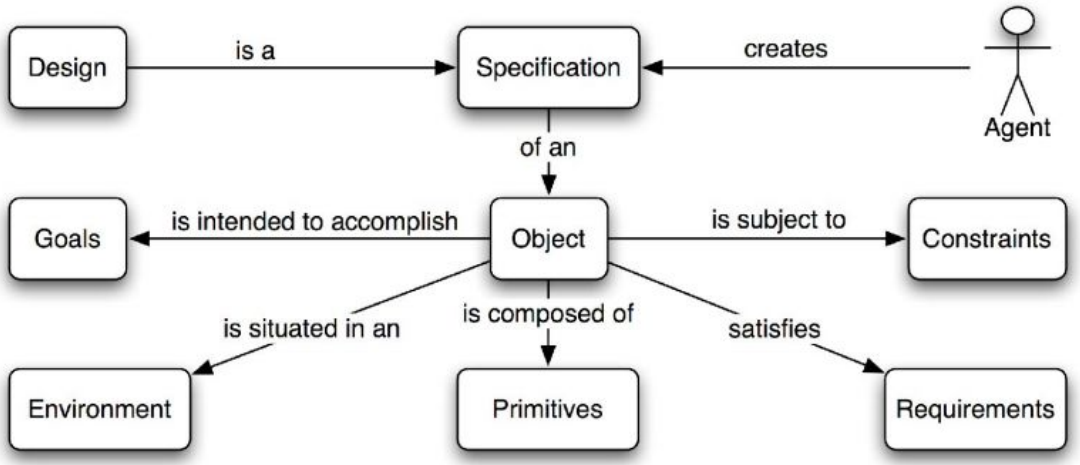
02. ¿Qué es el Diseño Detallado de Software?

Definición de Diseño*

"(noun) A specification of an object, manifested by an agent, intended to accomplish goals, in a particular environment, using a set of primitive components, satisfying a set of requirements, subject to constraints;

(verb, transitive) to create a design, in an environment (where the designer operates)".

- *Paul Ralph, Yair Wand (2013)*

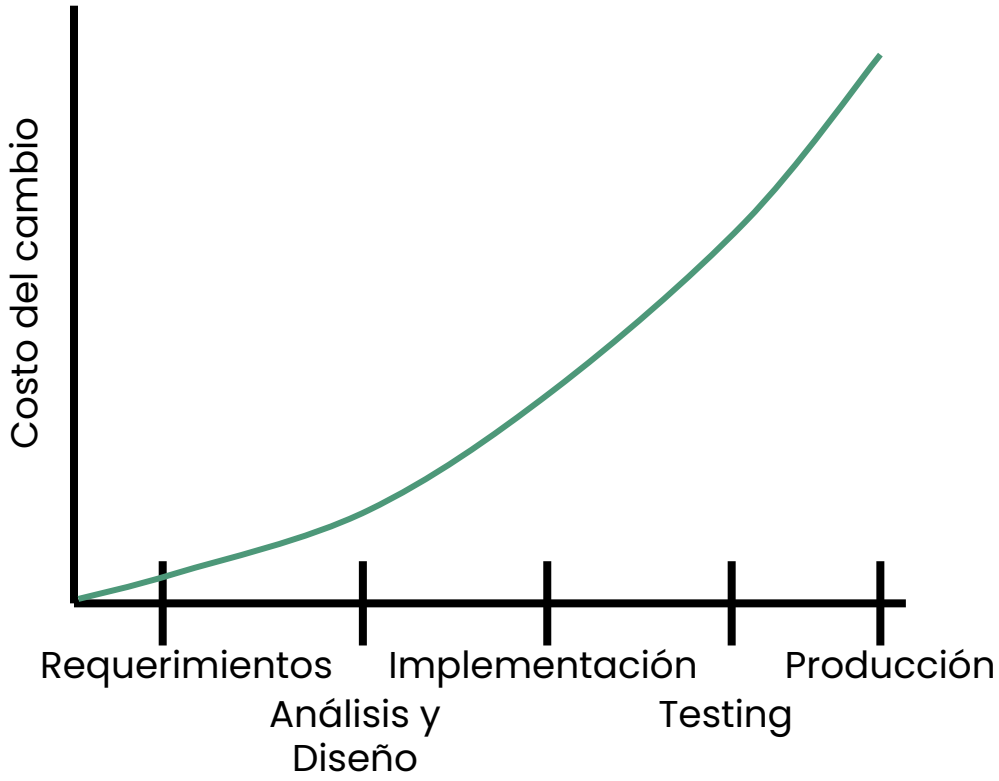


** Definición para efectos de este curso en el contexto del Desarrollo de Software.*

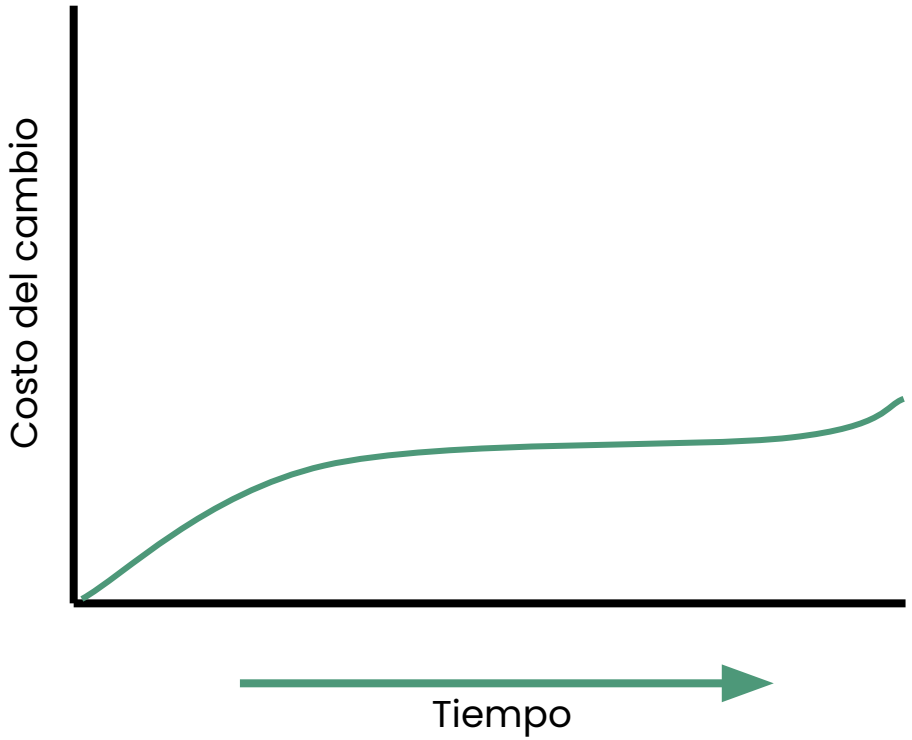
Importancia del diseño

Metodologías ágiles nos enseñan la importancia de un desarrollo iterativo-incremental para ir refinando nuestra solución a medida que nuestro entendimiento de la solución requerida vaya aumentando. ***Es inevitable que al trabajar un producto de software cometamos errores.***

Costo Tradicional



Costo Ágil

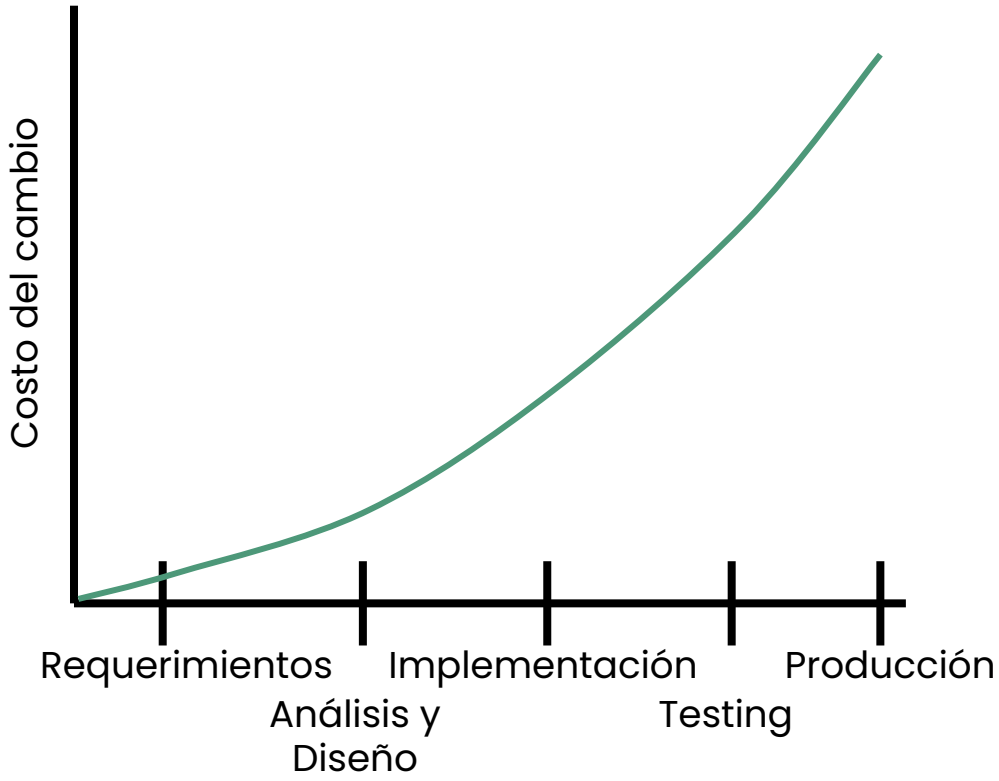


Scott Ambler, "Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process" (2002)

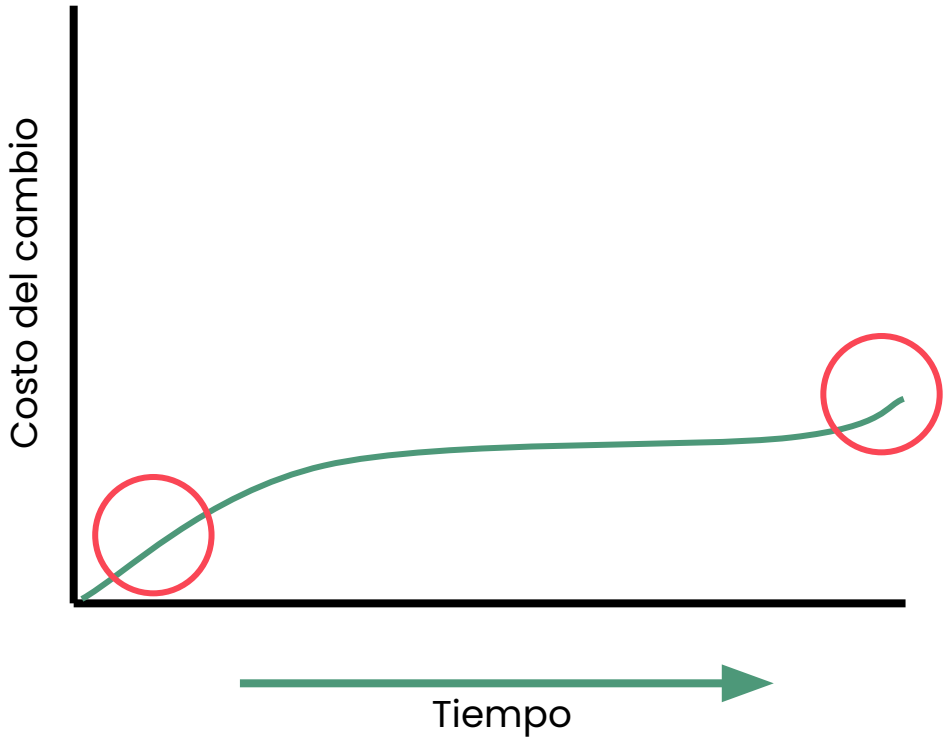
Importancia del diseño

No obstante, es mucho más barato corregir un error al momento de idear su concepción que al momento de programar. La fase de diseño es el momento en que la calidad del código se establece.

Costo Tradicional



Costo Ágil



Scott Ambler, "Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process" (2002)

Importancia del diseño

- ¿Qué pasa si se omite el diseño en todo el proceso? → **Corregir los errores toma tiempo**. Los problemas típicos que comienzan a aparecer son:
  - Hacks → deuda técnica.
  - Omisión de casos bordes → bugs.
  - Omisión de requisitos esenciales.
  - Retrasos.
  - Dificultad en integrar, mantener y extender.
  - Dependencia de miembros específicos del equipo.
  - Imposibilidad de testing.
  - Alejamiento de la necesidad del cliente.

03. Diseño en el proceso de desarrollo

Diseño en el proceso de desarrollo

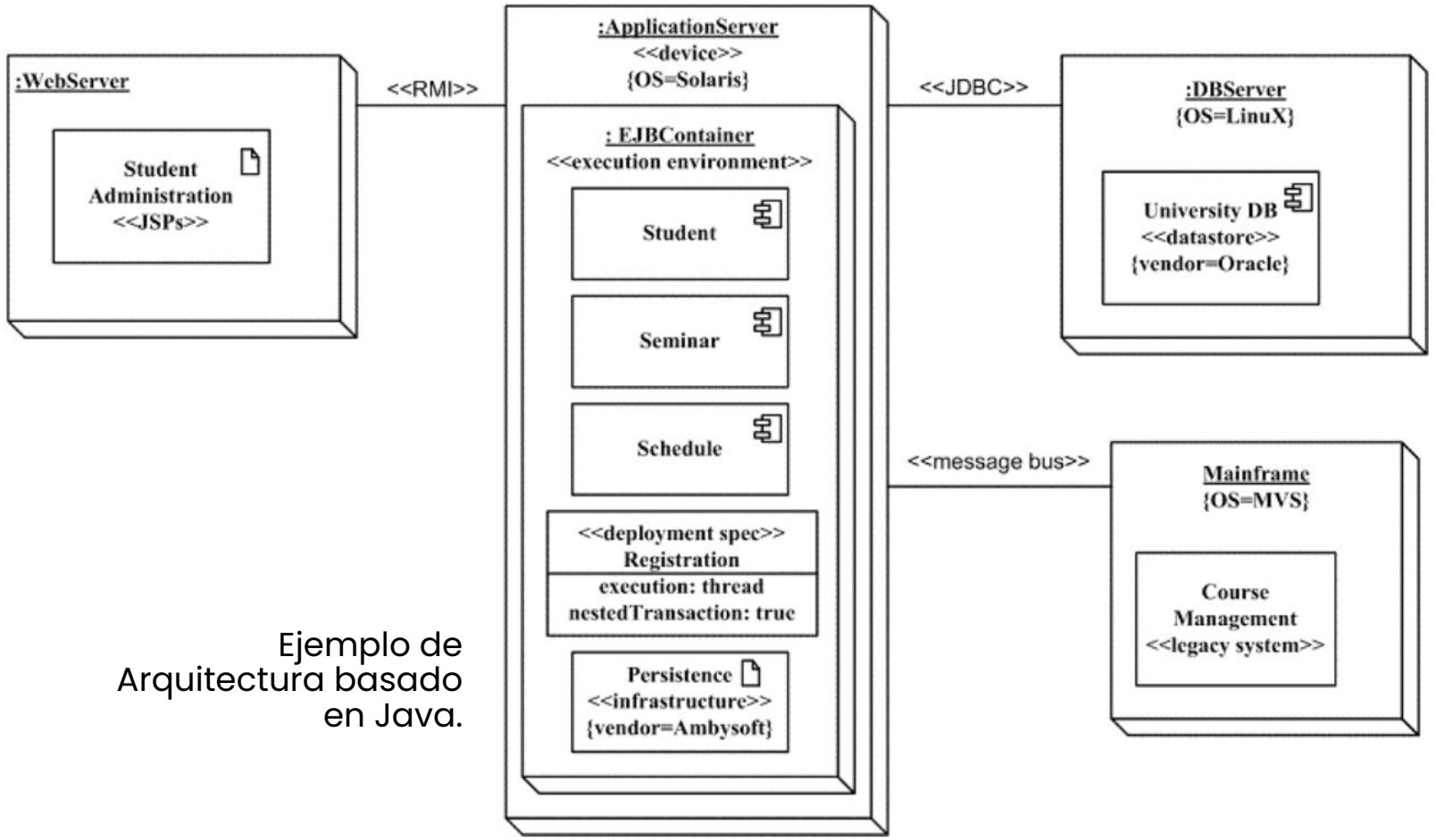
- En **metodologías ágiles**, se espera que la fase de diseño se mezcle con el proceso de desarrollo en distintas etapas.
  - Se vuelve al iterar sobre el diseño en la medida que se aterriza la conceptualización de la solución que se construye y el código que se escribe.
  - Idealmente, diseño es una tarea que queremos hacer en equipo.
- Desarrollos que tengan sus etapas muy definidas, diferencian la etapa de análisis y de diseño, donde el análisis es una versión simplificada o menos estructurada del diseño.
- Una herramienta usada para el diseño son **diagramas.**

Diseño en el proceso de desarrollo

- Podemos dividir el diseño en 4 áreas:
  - Diseño de Arquitectura
  - Diseño de Componentes
  - Diseño de Interfaces
  - Diseño de Clases/Datos

Diseño de Arquitectura

Abarca la representación de las estructuras de datos y los componentes de software necesarios para la construcción. Implica también la interacción entre las distintas tecnologías del proyectos.

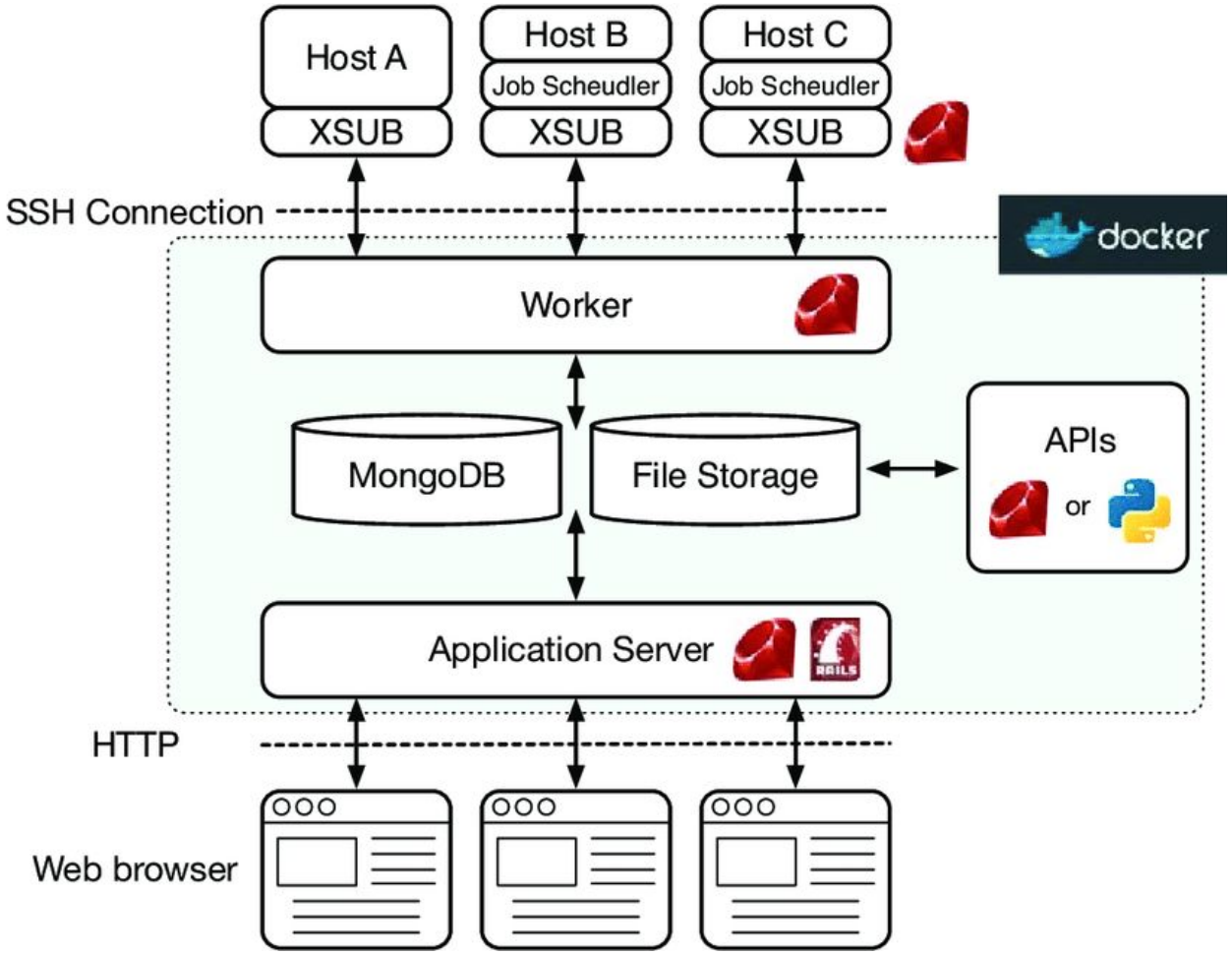


Ejemplo de Arquitectura basado en Java.

Scott Ambler, The Elements of UML 2.0 Style (2005)

Diseño de Arquitectura

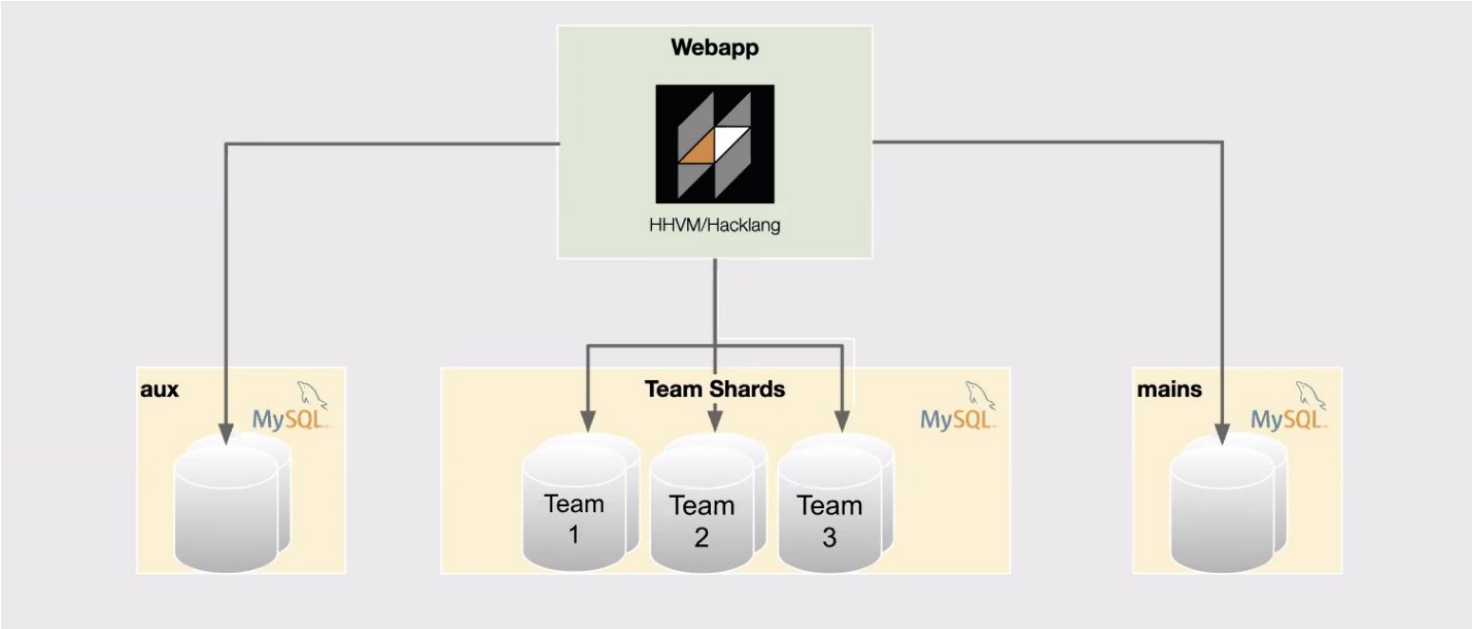
Ejemplo de Arquitectura Basado en Ruby.



Yohsuke Murase, An open-source job management framework for parameter-space exploration: OACIS (2001).

Diseño de Arquitectura

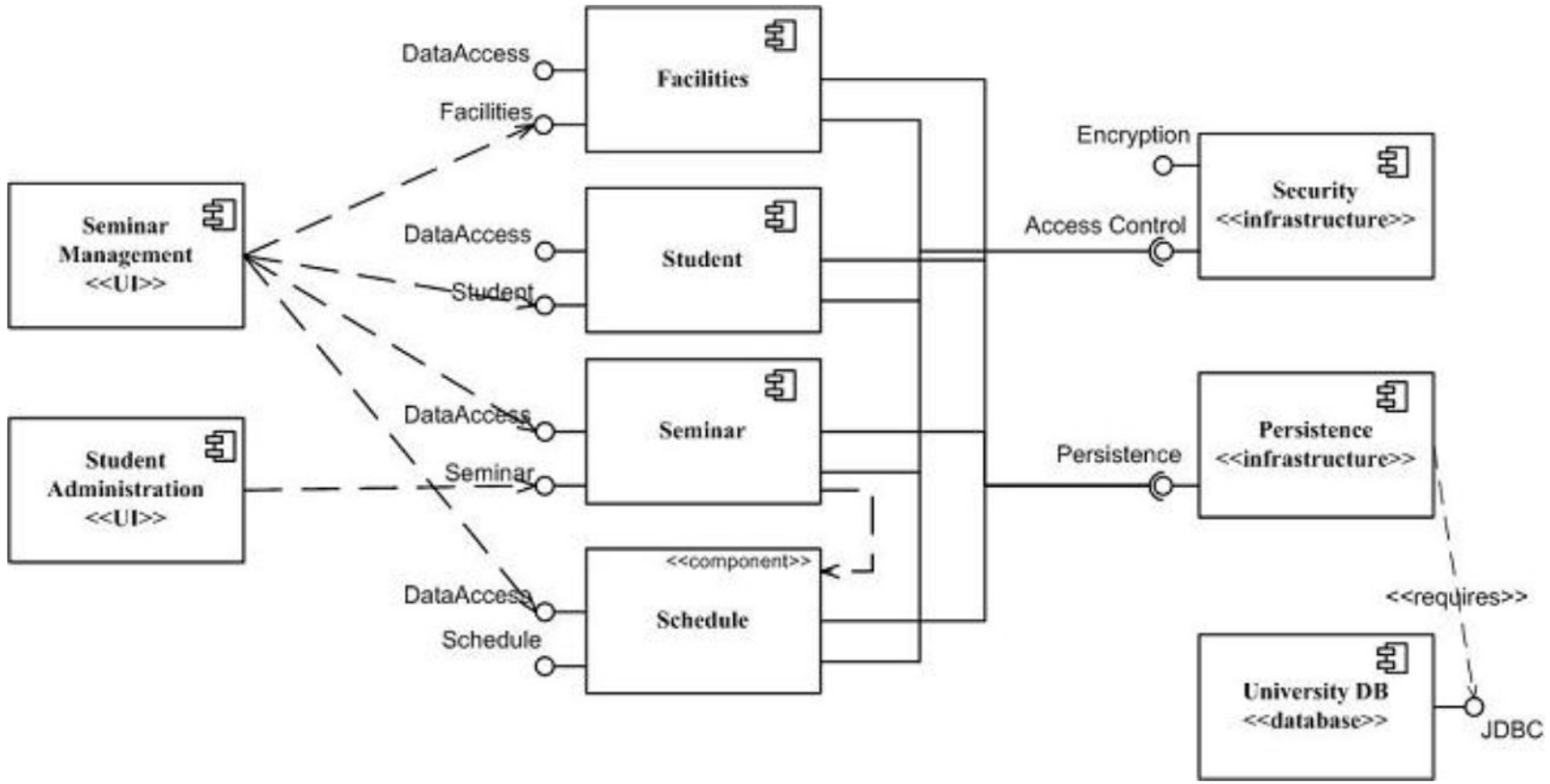
Otro de Arquitectura.



Diseño de Componentes

La arquitectura de un sistema define cuáles tipos de componentes mínimos debe tener.

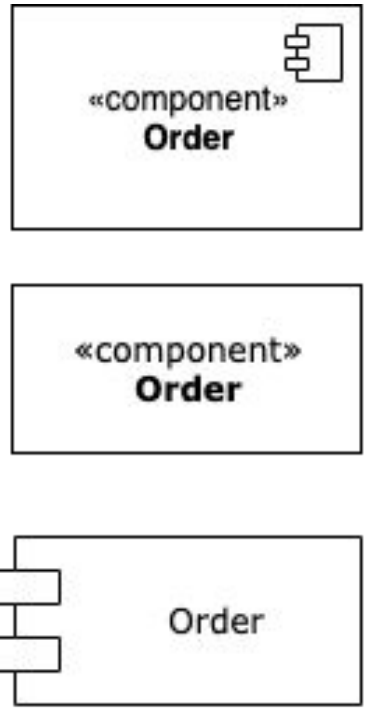
Durante el diseño de componentes se busca concretizar los componentes seleccionados, aportando mayor precisión sobre sus estructuras de datos, algoritmos, interfaces y mecanismos de comunicación; y dividiéndolos en sub-componentes con responsabilidades más acotadas.



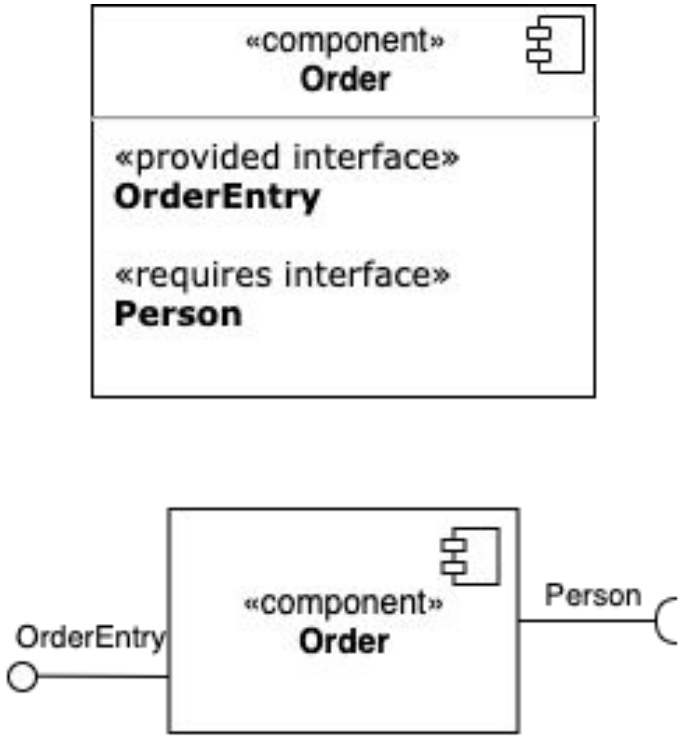
Scott Ambler, The Elements of UML 2.0 Style (2005)

Diseño de Componentes

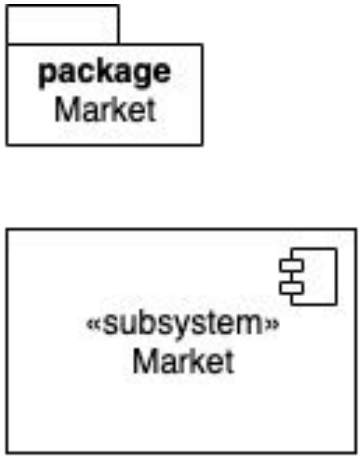
Ejemplos de componentes



Ejemplos de interfaces



Otros elementos usados



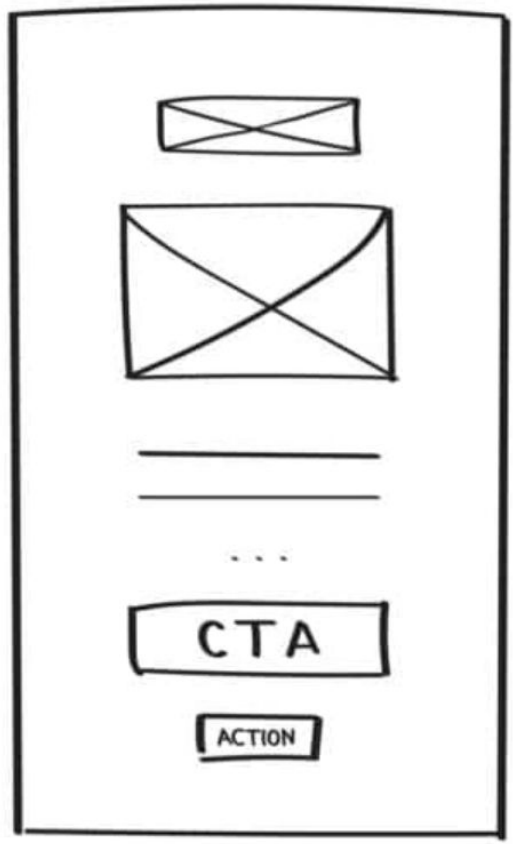
Diseño de Interfaces

- Comunicación con sistemas externos y humanos.
  - Diseño de Interfaces Gráficas (UI)
  - Diseño de API’s externas
  - Diseño de API’s internas
- Algunos diagramas utilizados: Wireframes, Flowchart, Communication diagram, etc.-

Scott Ambler, The Elements of UML 2.0 Style (2005)

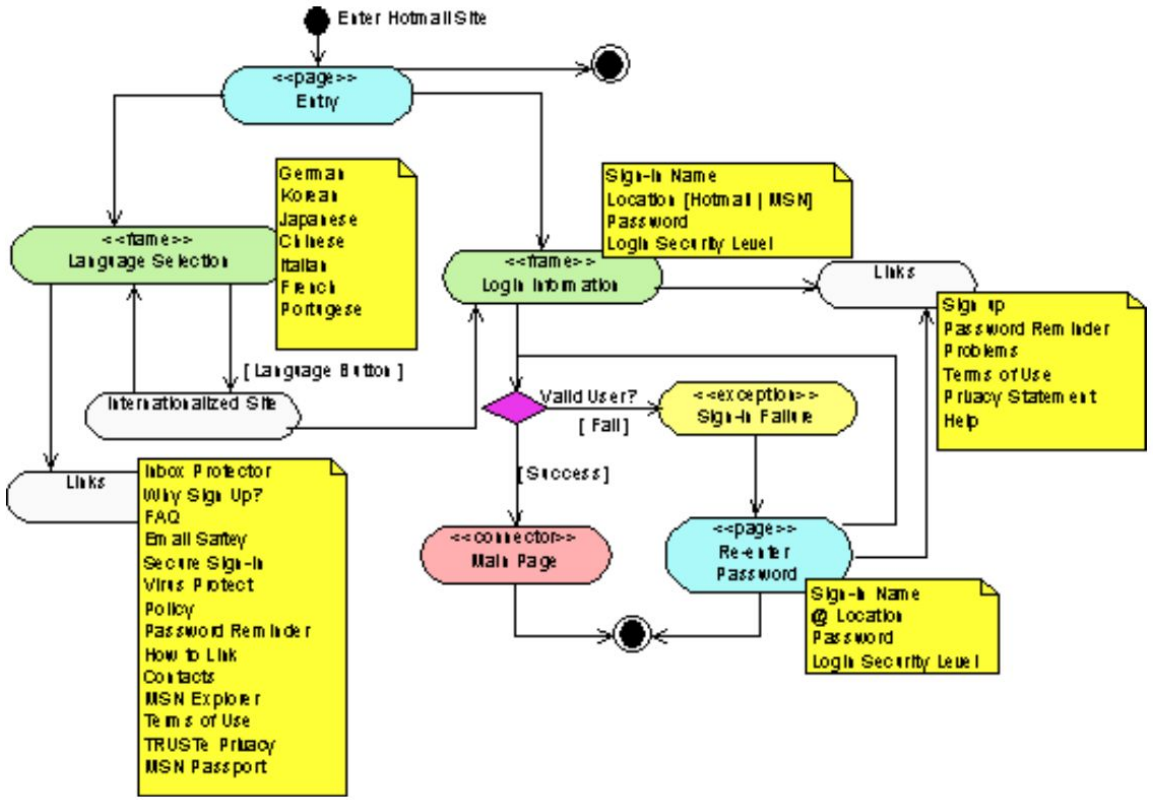
Diseño de Interfaces

Diseño UI: Wireframe (low fidelity)



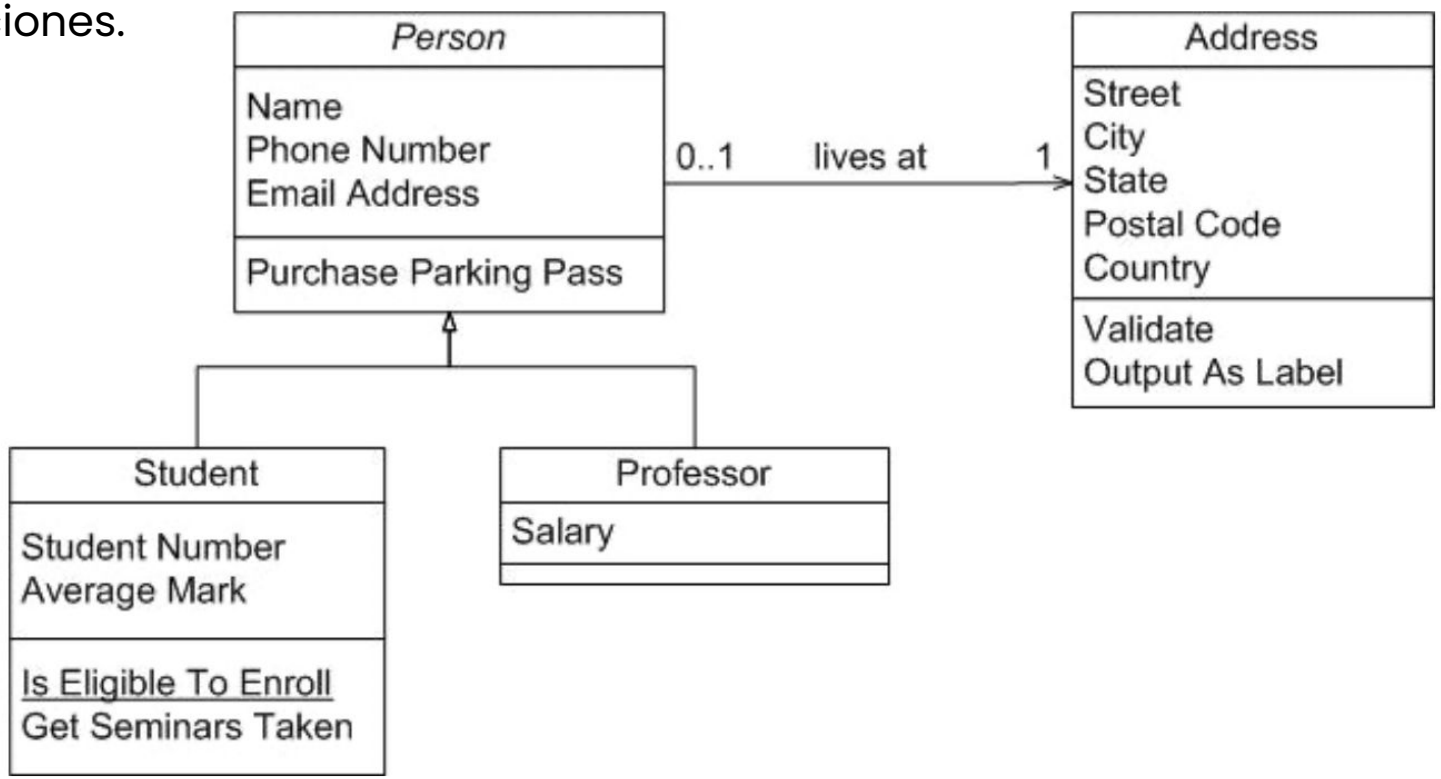
Wireframe hecho por Tsvetelina Lazarova.

Activity Diagram (o diagrama de flujo)



Diseño de Clases/Datos

- Esquema de clases y sus relaciones.



03. Principios Fundamentales

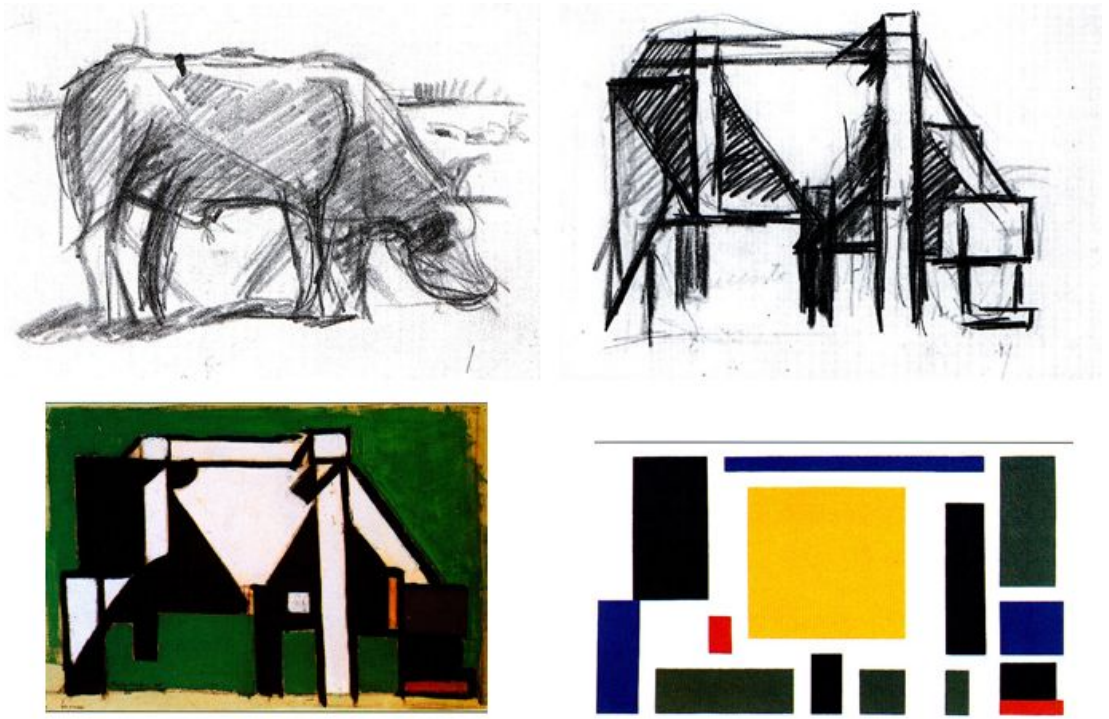
Principios fundamentales

- El objetivo de la fase de diseño es garantizar la calidad de la solución.

- Abstracción
- Cohesión
- Ocultamiento
- Acoplamiento

Abstracción

- Rescatar solo la información relevante dado un contexto.
- Debe cumplir con criterios de **Suficiencia** y **Completitud**



Theo van Doesburg, 1917.

Encapsulamiento

- O ocultamiento. No exponer información o lógica innecesaria.
- Ejemplos:
  - Servicios Web
  - Librerias
  - Módulos con modificadores de acceso
- Beneficios:
  - Mantenibilidad
  - Reusabilidad
  - Extensibilidad

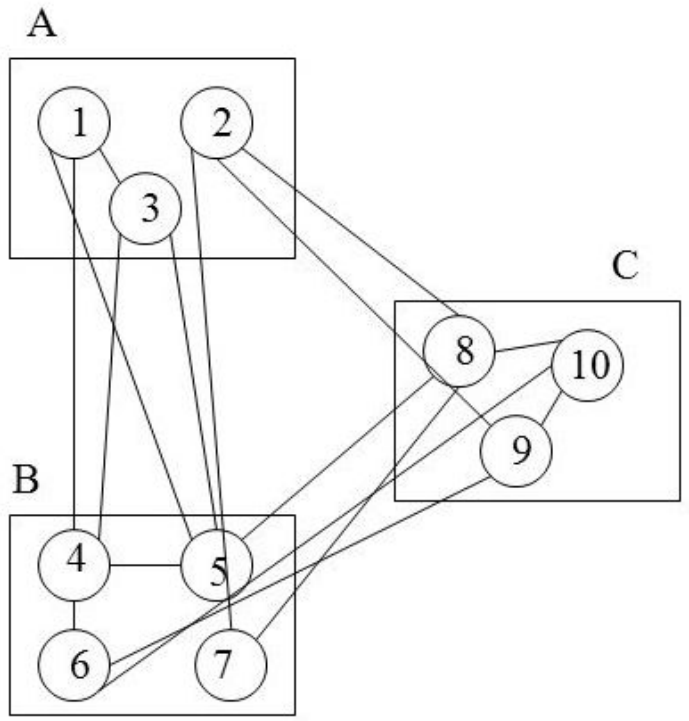
Cohesión

- Una clase tiene un conjunto preciso y acotado de responsabilidades y todos sus atributos y métodos están únicamente relacionados con cumplir esas responsabilidades
- Beneficios:
  - Reduce complejidad
  - Aumenta mantenibilidad

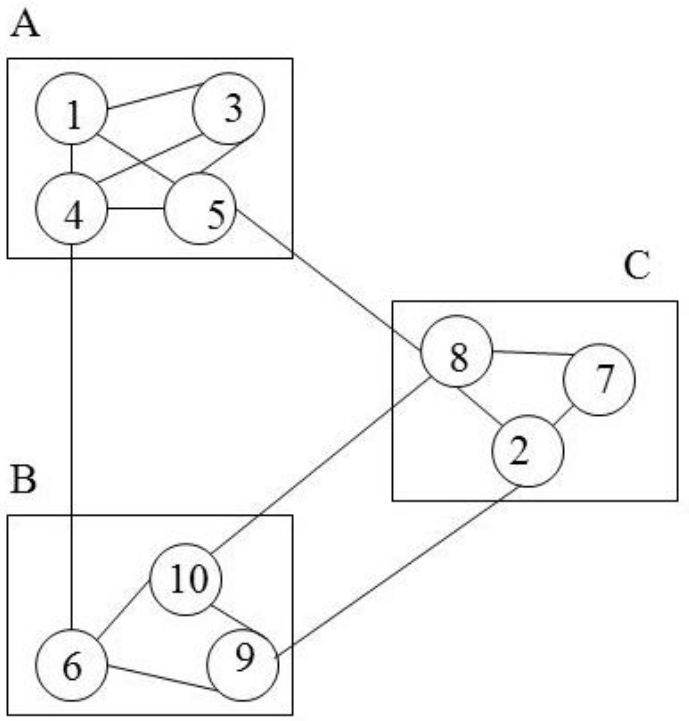
Acoplamiento

- El acoplamiento es una medida de cuán conectados están dos subsistemas o clases.
  - Alto acoplamiento: un módulo modifica o se apoya en el funcionamiento interno de otro módulo → En general es mala práctica.
    - Un cambio en un módulo puede generar un efecto dominó.
    - Una nueva conexión entre módulos puede ser más costosa debido a las conexiones ya existentes.
    - Es más difícil de testear.

Alta Cohesión, Bajo Acoplamiento



Bad modularization:
low cohesion, high coupling



Good modularization:
high cohesion, low coupling

Viva Differences. 2019. "Difference Between Coupling And Cohesion In Software Engineering.

Design Quality Guidelines

1. Exhibe una arquitectura con patrones reconocidos.
2. Es modular, particionado lógicamente en subsistemas.
3. Contempla datos, arquitectura, interfaces y componentes.
4. Permite derivar estructuras de datos óptimas para el problema en cuestión.
5. Revela componentes con características funcionales independientes.
6. Revela interfaces sencillas que minimizan la complejidad de interactuar con el sistema.
7. Se deriva lógicamente del análisis de los requisitos del proyecto.
8. Utiliza una notación estándar para comunicar su significado.

05. Próxima Clase

Próxima clase

- Modelo 4+1
- Diagramas UML 2.0

Clase 1 - Motivación

Bibliografía

- S. Ambler, Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.
- S. Ambler, The Elements of UML(TM) 2.0 Style.
- J. Kramer, Is Abstraction The Key To Computing?

IIC2113 - Diseño Detallado de Software

Fernanda Sepúlveda - mfsepulveda@uc.cl

Pontificia Universidad Católica de Chile
2020-2