Módulo 4: Diseño de software

Diseño y Desarrollo de Software (*1er. Cuat. 2019*)

Profesora titular de la cátedra:
Marcela Capobianco

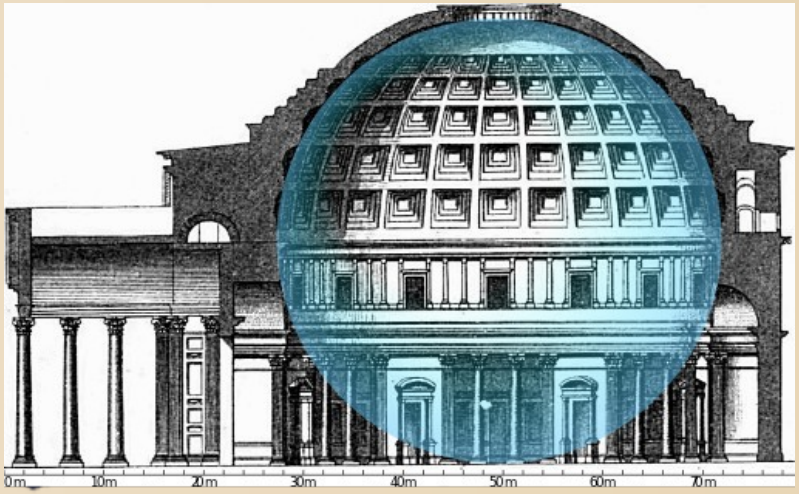
Profesores interinos:
Sebastián Gottifredi
Gerardo I. Simari

Licenciatura en Ciencias de la Computación – UNS

Licencia

Diseño

- Marcus Vitruvius Pollio (arquitecto romano conocido como *Vitruvius*): “Una estructura debe poseer *firmitas*, *utilitas* y *venustas*” (ser sólido, útil y bello).
- Este es el objetivo principal del diseño.



Diseño: ¿*Para que*?

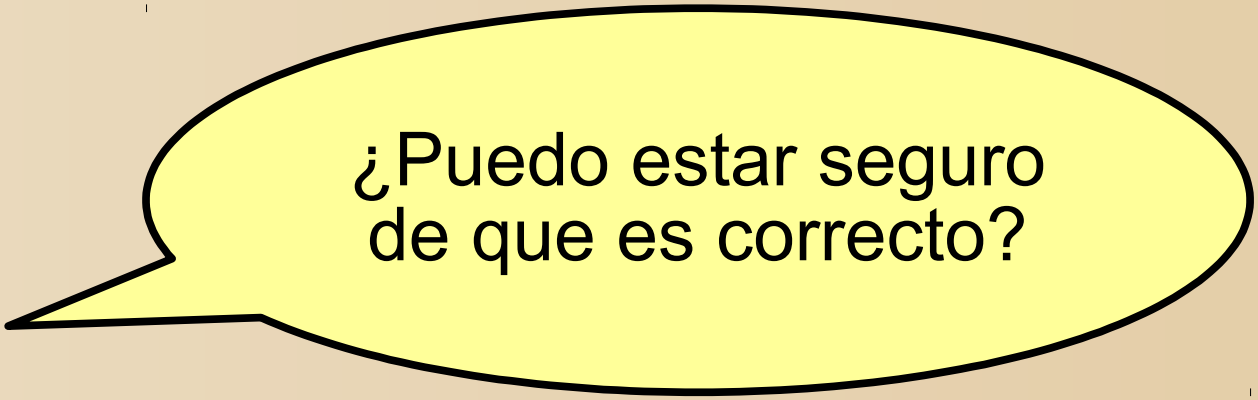
- *Objetivo*: Producir un modelo de una entidad que será construida *a posteriori*.
- Dos fases:
  - Diversificación
  - Convergencia.
- En el software, se centra en cuatro áreas importantes:
  - Datos
  - Arquitectura
  - Interfases
  - Componentes

Pasos para el diseño de SW

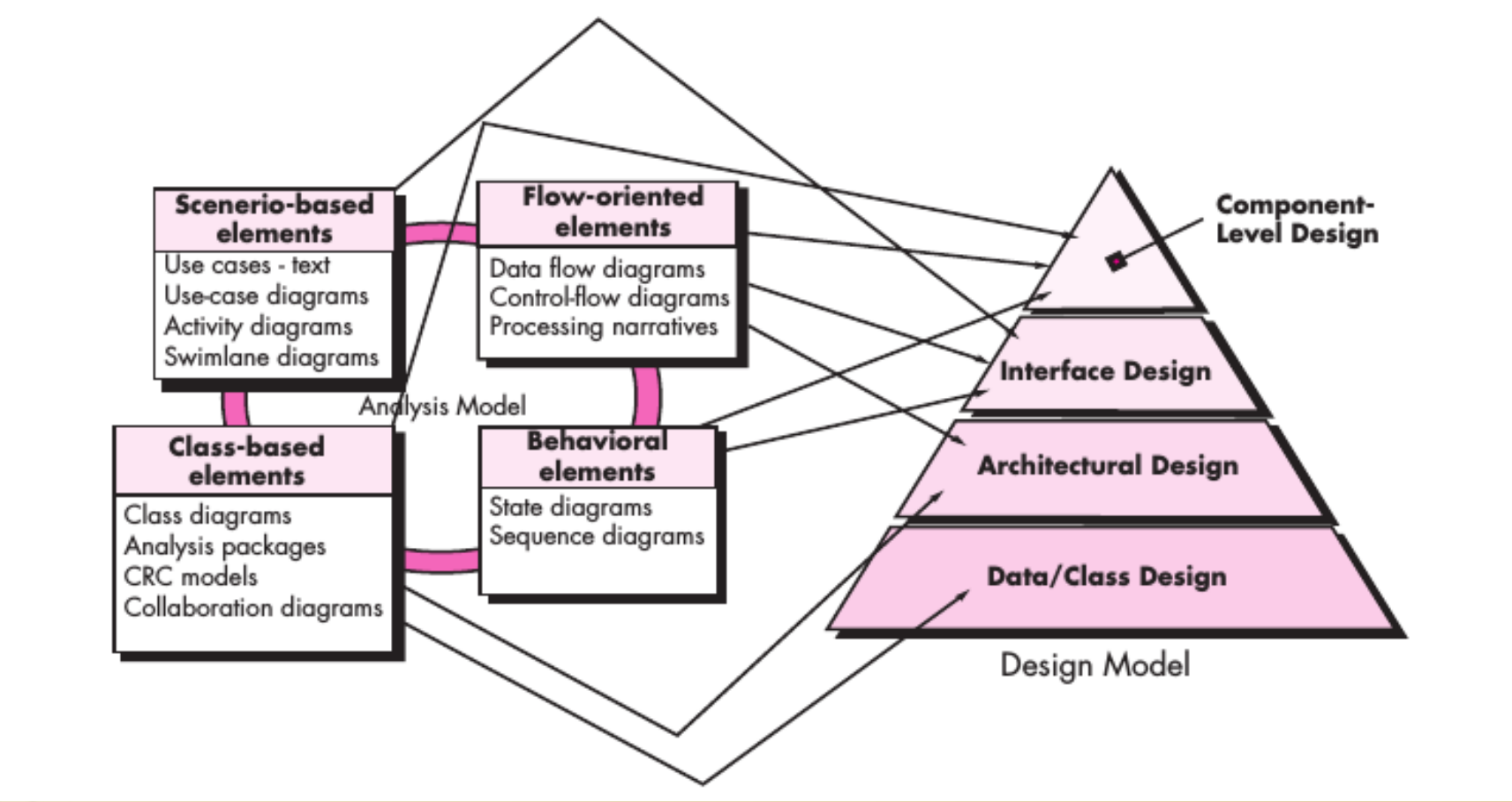
- Transformar el modelo de requerimientos.
- Se desean obtener cuatro niveles de detalle de diseño:
  - Estructura de datos
  - Arquitectura del sistema
  - Representación de la interfaz
  - Detalles a nivel de componentes

Producto obtenido

- Como resultado se obtiene una *especificación del diseño*.
- Se compone de los *modelos* de los datos, arquitectura, interfaz y componentes.



Del análisis al diseño



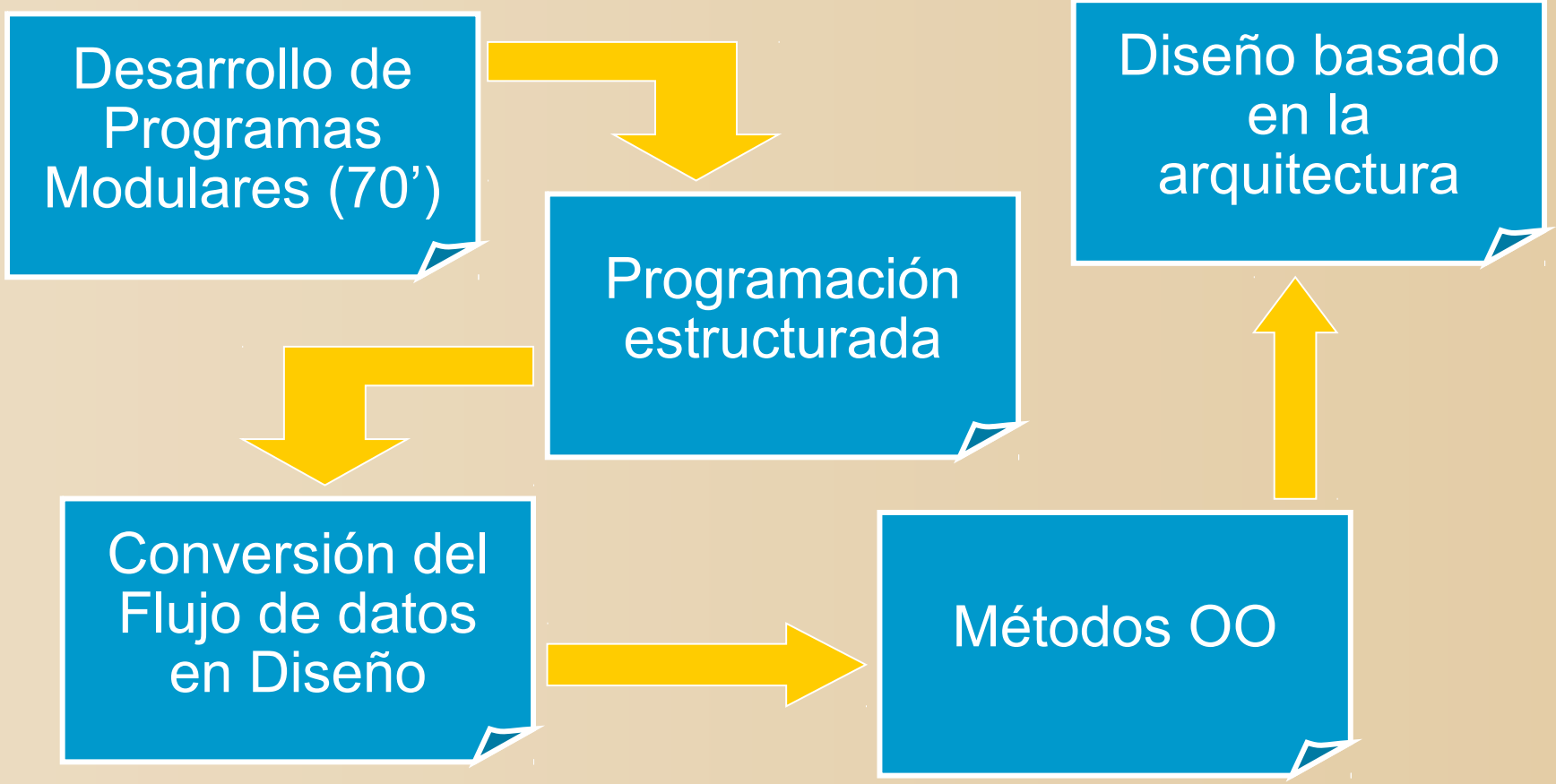
Cambio continuo

- Junto con los requerimientos, el diseño del SW sufre una continua *evolución*.
- Como vimos anteriormente, sus métodos carecen de la calidad de los asociados a otras disciplinas de ingeniería.
- Para ayudar, surge el concepto de *calidad del diseño*.

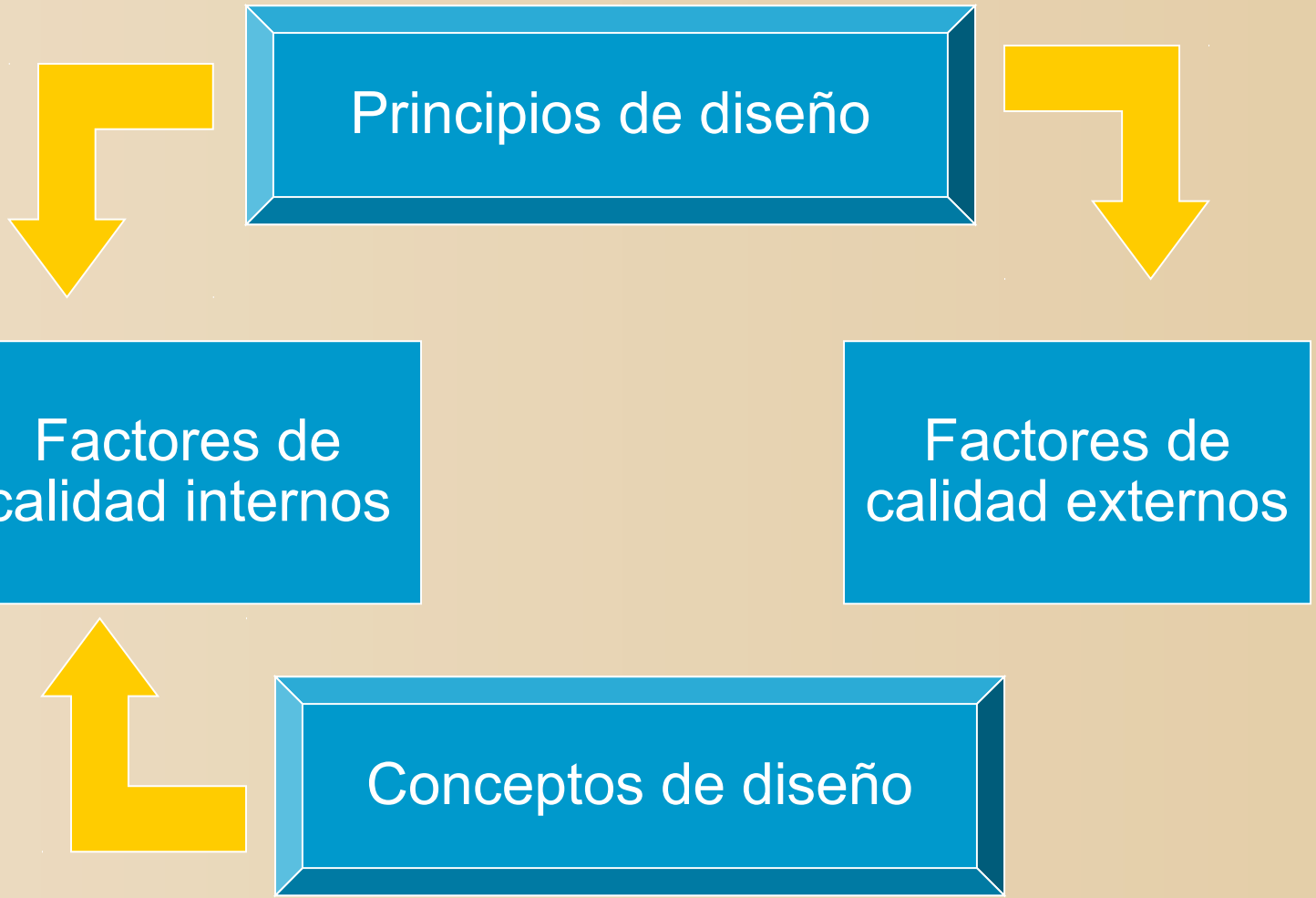
Criterios técnicos para el diseño

- Buena estructura arquitectónica.
- Modularidad
- Conducir a:
  - ED adecuadas
  - Componentes adecuados
  - Interfaces que reduzcan la complejidad del entorno

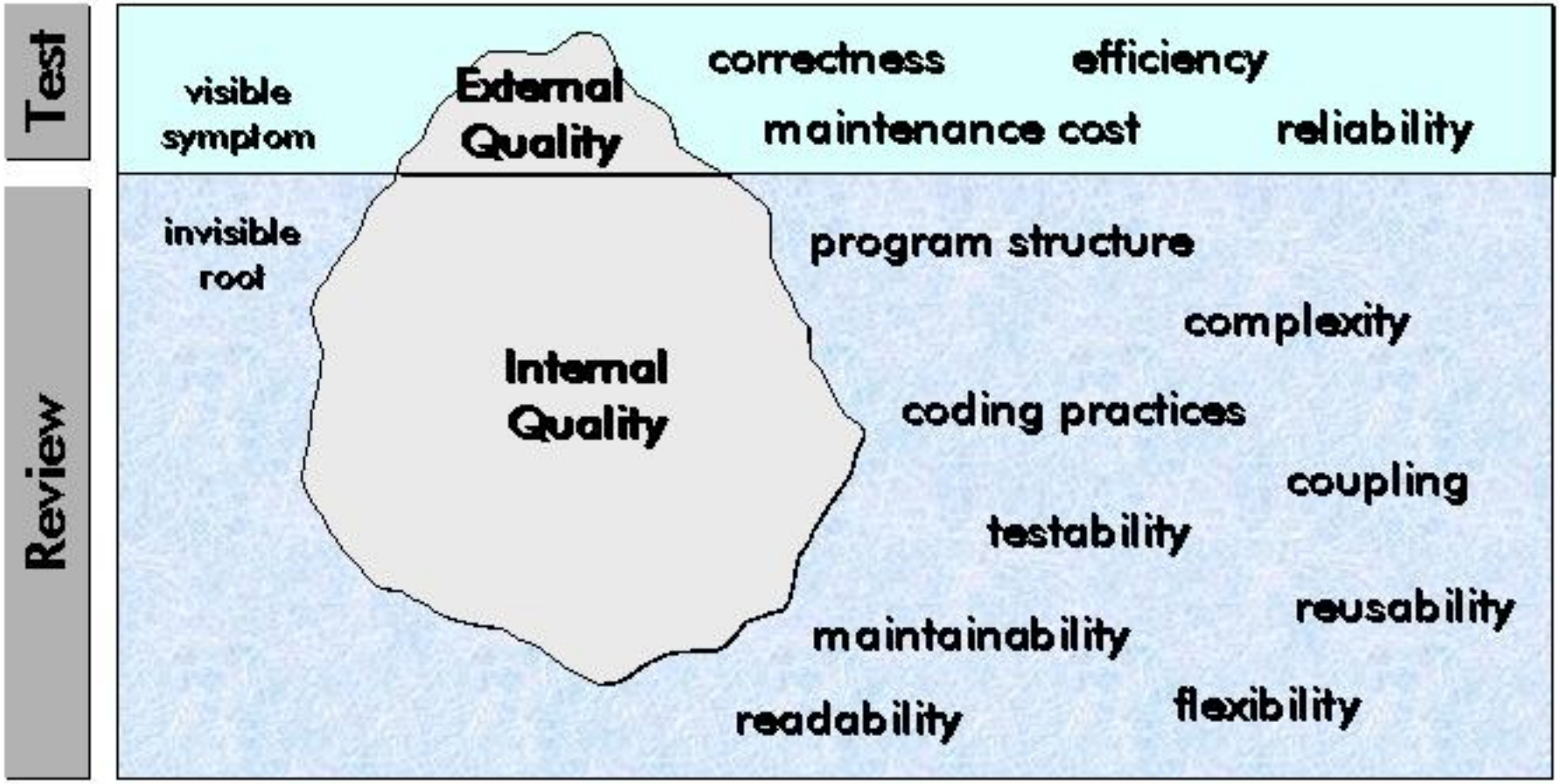
Evolución del diseño



Factores de calidad



The Software Quality Iceberg



Conceptos del diseño

- Los *conceptos* han experimentado el paso del tiempo y el grado de interés en cada uno ha ido variando.
- Ayudan a responder a las siguientes preguntas:
  - *¿Qué criterios usar para la partición del software*?
  - *¿Cómo separar la lógica y las estructuras de datos*?
  - *¿Existen criterios uniformes que definan la calidad técnica de un diseño de software*?

1. Abstracción

- La *abstracción* es una de las formas fundamentales de manejar la complejidad.
- Al considerar una solución *modular* a un problema se pueden exponer muchos niveles de abstracción.
- En los niveles inferiores la orientación es *procedimental*.
- En el nivel más alto se emplea el lenguaje del *entorno* del problema.

2. Refinamiento

- El *refinamiento paso a paso* es una estrategia propuesta por Niklaus Wirth.
- El refinamiento de programas es análogo al que se realiza en el análisis de requerimientos.
- *Consejo*: no entrar en los detalles inmediatamente, saltándose el refinamiento.
- El refinamiento es un proceso de *elaboración* de cada una de las sentencias de funciones definidas en un nivel alto de abstracción.

3. Modularidad

- *La arquitectura expresa la modularidad*: el SW se divide en componentes nombrados y abordados por separado (módulos).
- El software monolítico es *ilegible* y por ende también muy difícil de mantener.
- Se basa en la estrategia "*divide y vencerás*".
- *Consejo*: no modularizar de más.
- *¿Cómo se define un módulo con el tamaño adecuado*?

4. Arquitectura

La arquitectura del SW alude a la *estructura global* del software y a las formas en que la estructura proporciona la *integridad conceptual* de un sistema.

Arquitectura:

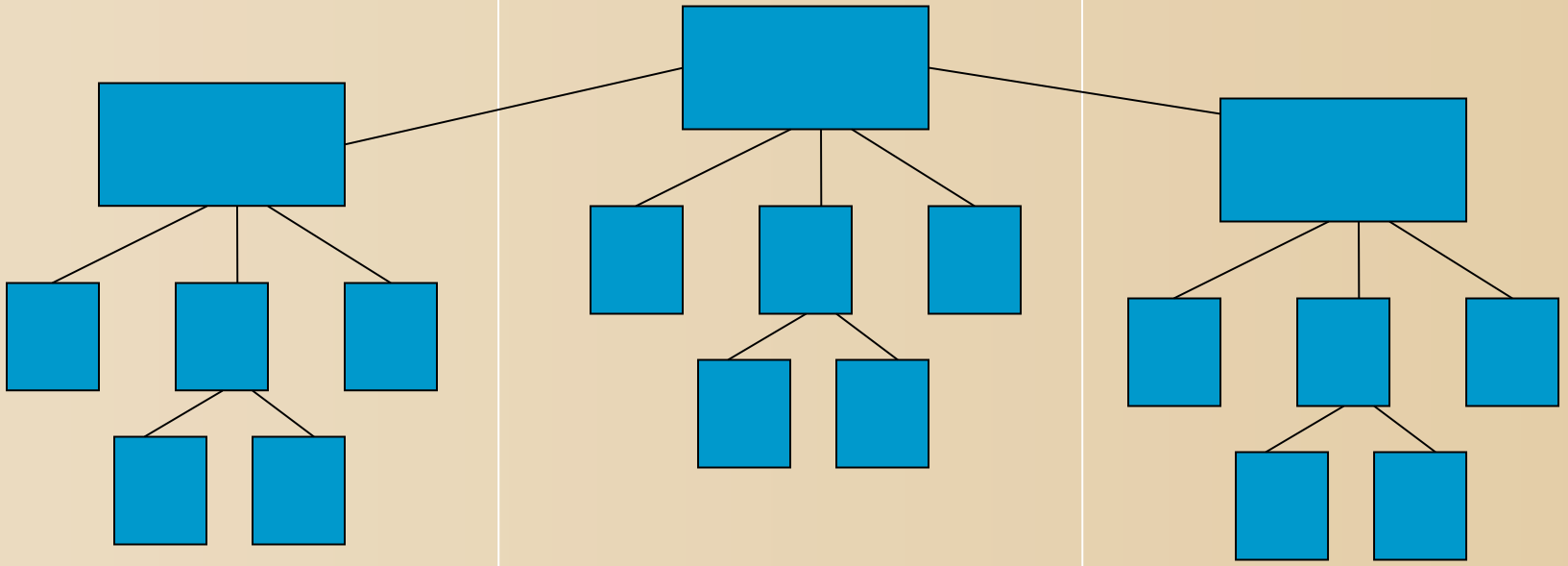
- Estructura jerárquica de los componentes del programa, forma en que interactúan y estructuras de datos que usan.
- Un conjunto de *patrones arquitectónicos* permiten reusabilidad a nivel de diseño.

5. Jerarquía de control

- Representa la jerarquía del *flujo de control* entre los módulos.
- No se puede usar en todos los estilos.
- Se usan distintos *esquemas*; el más común es el de *árbol* (para llamada y retorno).
- Se define una serie de *métricas* en este modelo (profundidad, grado de salida, grado de entrada)
- Métricas más sutiles: visibilidad y conectividad.

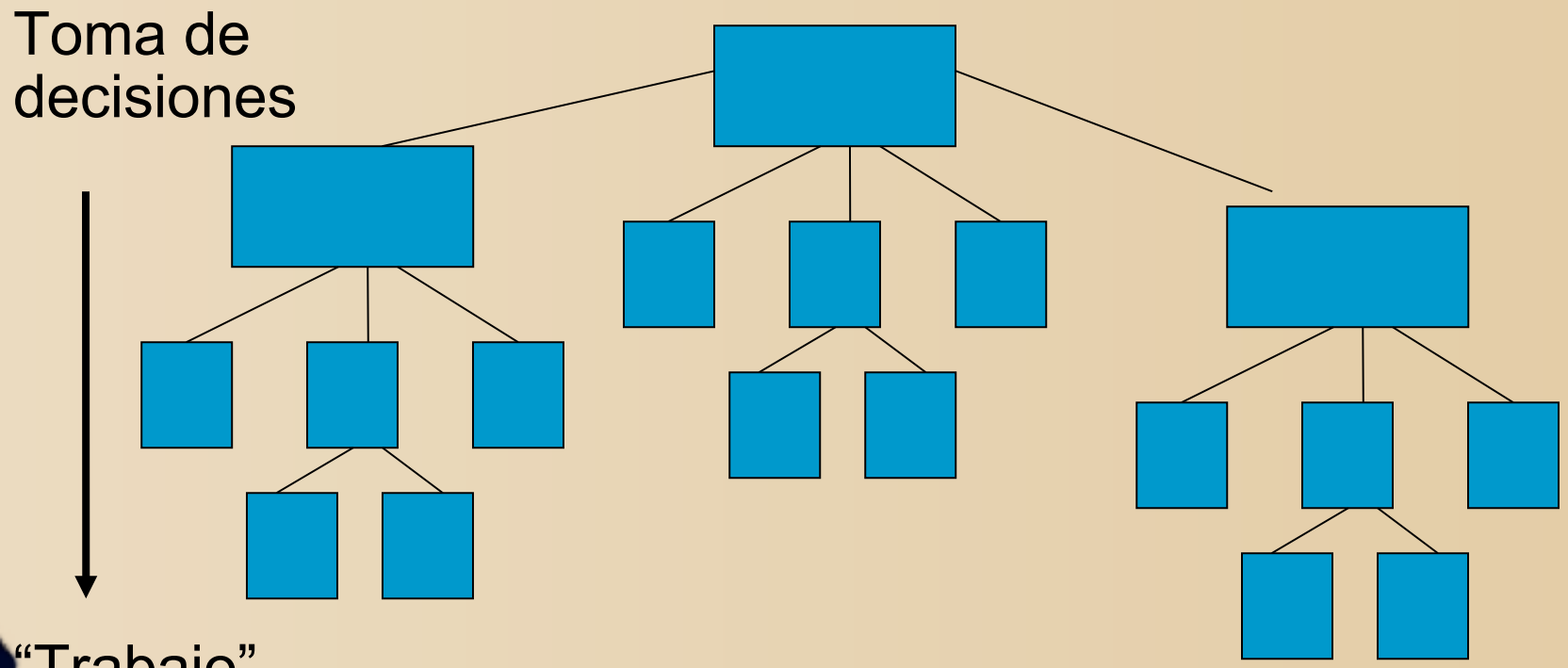
6. División estructural

En estilos arquitectónicos jerárquicos, la estructura del programa se puede dividir *horizontal* y *verticalmente:*



6. División estructural

Ventaja de la partición *vertical*: los cambios se acomodan más fácilmente:



7. Estructura de datos

- Representación de la relación lógica entre elementos individuales de datos.
- Afecta al diseño procedimental final.
- Dicta las alternativas de organización, métodos de acceso, etc.
- La elección de estructuras de datos puede tener un impacto importante en la *eficiencia en tiempo y espacio* del producto final.

8. Procesamiento

- Se centra en el procesamiento de cada módulo individual.
- Se debe proporcionar información precisa del procesamiento, incluyendo los puntos de decisión, secuencia de sucesos y operaciones repetitivas.

9. Ocultamiento de la información

- Los módulos se caracterizan por las decisiones de diseño que cada uno *oculta* al otro.
- La abstracción ayuda a definir las *entidades* en forma adecuada.

Diseño modular

- Los conceptos básicos incentivan un diseño modular.
- Esto reduce la *complejidad*, facilita los *cambios* y fomenta el *paralelismo* en el desarrollo.
- Independencia funcional:

  modularidad + abstracción + ocultamiento

- Los módulos deben poseer una función determinante y una aversión a la *interacción excesiva* con otros módulos.

Cohesión y acoplamiento

- La independencia se mide mediante dos criterios cualitativos: la *cohesión* y el *acoplamiento*.
- La *cohesión* es una medida de la fuerza relativa funcional de *cada módulo*.
- El *acoplamiento* es una medida de la independencia relativa *entre* los módulos.
- La cohesión suele clasificarse en:
  - procedimental (alta),
  - lógica,
  - temporal,
  - coincidental (baja)

Cohesión y acoplamiento

- En el diseño del SW intentamos conseguir un *acoplamiento lo más bajo posible*.
- Una conectividad baja es más fácil de entender y disminuye el “efecto en cadena” de errores.
- Se categoriza en:
  - de datos,
  - de control,
  - externo (de E/S),
  - común (área global de datos),
  - de contenido (debido a bifurcaciones de control, debe evitarse)

Especificación del diseño

- Diseño de datos
- Diseño arquitectónico (indica cómo se derivó la estructura del modelo de análisis)
- Diseño de interfases internas y externas; puede incluir un prototipo de la interfaz de usuario (GUI)
- Descripción de componentes
- Referencia cruzada de requisitos
- Restricciones de diseño
- Manual preliminar de operaciones

¿Qué es el diseño arquitectónico?

- *Arquitectura*: Organización del sistema como un conjunto de componentes, propiedades de estos componentes y su interrelación.
- Las arquitecturas han existido desde el primer programa que usó módulos.
- Históricamente se han dado en forma *implícita*.

Historia

- Originado en un trabajo de Dijkstra en 1968 y posteriormente ampliado por Parnas en los años 1970s.
- Se enfatizó que la estructura del sistema es crítica.
- Incrementó su popularidad en los años 1990s con trabajos de investigación sobre:
  - Patrones
  - *Architecture Description Languages* (ADLs)
  - Métodos formales
  - Documentación de la arquitectura

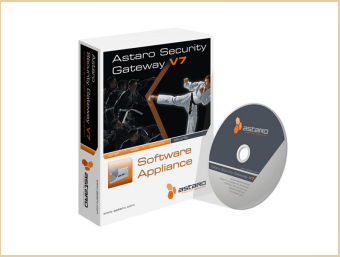
Nivel del diseño arquitectónico

- Organización del sistema como un conjunto de componentes
- Control global
- Protocolos de comunicación
- Acceso a los datos
- Distribución física
- Escalabilidad
- Performance
- ...

Analogía: Edificios vs. Software

- Requerimientos para el edificio
- Diseño preliminar
- Se hacen planos
- La construcción se basa en los planos

- Requerimientos para el software
- Diseño de alto nivel
- Diseño detallado (*algoritmos*)
- Implementación
- *Deployment*



Algunas consideraciones

Las propiedades de las estructuras son inducidas por el diseño de su arquitectura

Algunas consideraciones

- Un arquitecto requiere un entrenamiento amplio.
- No basta con ser competente en aspectos técnicos.
- La habilidad para programar, ¿es suficiente para crear aplicaciones complejas?

Algunas consideraciones

- El proceso de desarrollo no es tan importante como la arquitectura.
- La arquitectura (clásica) como disciplina ha madurado a través de los años.
- Una forma en que se resumieron las experiencias y lecciones de generaciones de arquitectos es a través de la noción de *estilos arquitectónicos*:
  - Catedral gótica
  - Villa romana
  - Chalet suizo
  - Casa colonial, etc

Cómo resumir un estilo

- El desarrollo de un estilo en el tiempo refleja el conocimiento y experiencia adquirido para cumplir con un *conjunto de requerimientos*.
- Respeta un conjunto de restricciones dictadas por el clima, topología, materiales disponibles, y muchos otros criterios.

Limitaciones de la analogía

- Sabemos mucho sobre edificios, pero no tanto sobre software.
- *El software es más abstracto*, y por lo tanto más difícil de comprender y analizar.
- El software es más maleable.
- El *deployment* del SW no tiene análogo en la construcción.
- El software tiene un carácter *más dinámico*.

Utilidad del diseño arquitectónico

- Proporciona un *panorama completo* del sistema a construir.
- Permite, entre otras cosas:
  - Realizar una descripción del sistema a través de *constructores* y *patrones*.
  - Comenzar con el diseño de datos para luego derivar una o más representaciones alternativas.

Utilidad del diseño arquitectónico

- Documentar la arquitectura del SW facilita la *comunicación entre los participantes*.
- Como vimos en la unidad anterior, este aspecto es fundamental para el desarrollo efectivo.
- Permite reutilizar diseños de componentes y patrones entre distintos proyectos.
- Documenta decisiones tempranas sobre diseño de alto nivel.

Ejemplo: La arquitectura de la Web

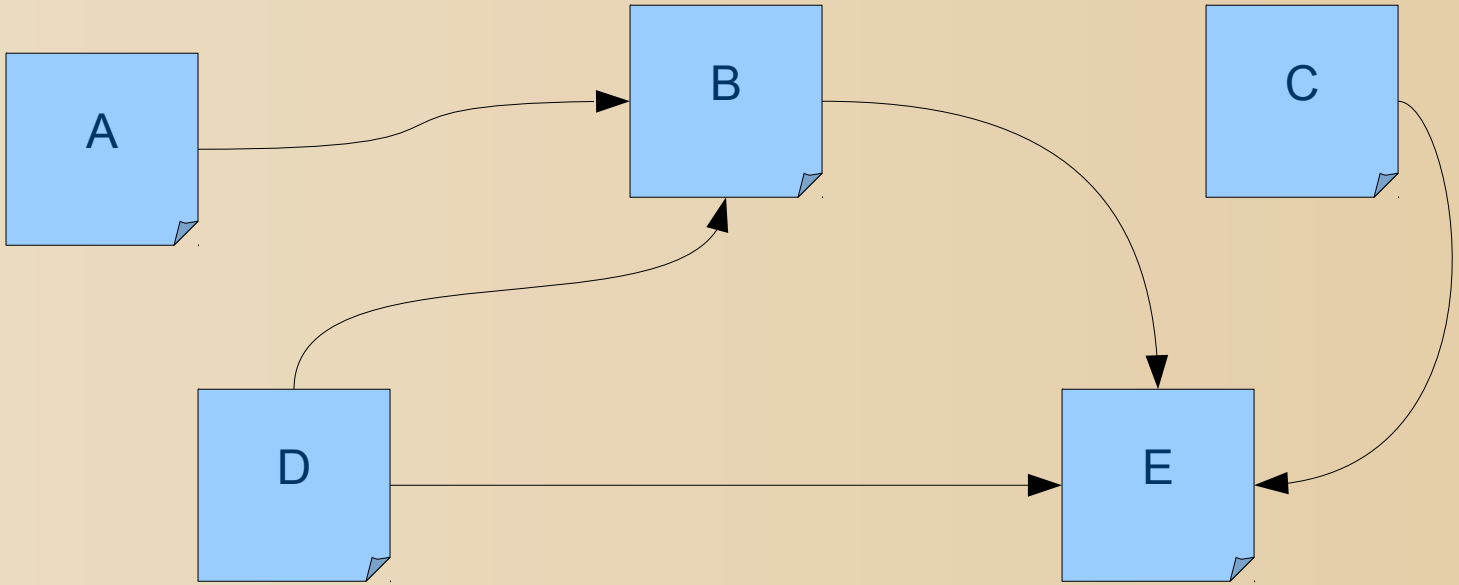
Ejercicio:

- Desde el punto de vista de un sistema de software, ¿*qué es la World Wide Web*?
- ¿Cómo la describiría?

Ejemplo: La arquitectura de la Web

A alto nivel, desde un punto de vista *funcional* (del usuario):

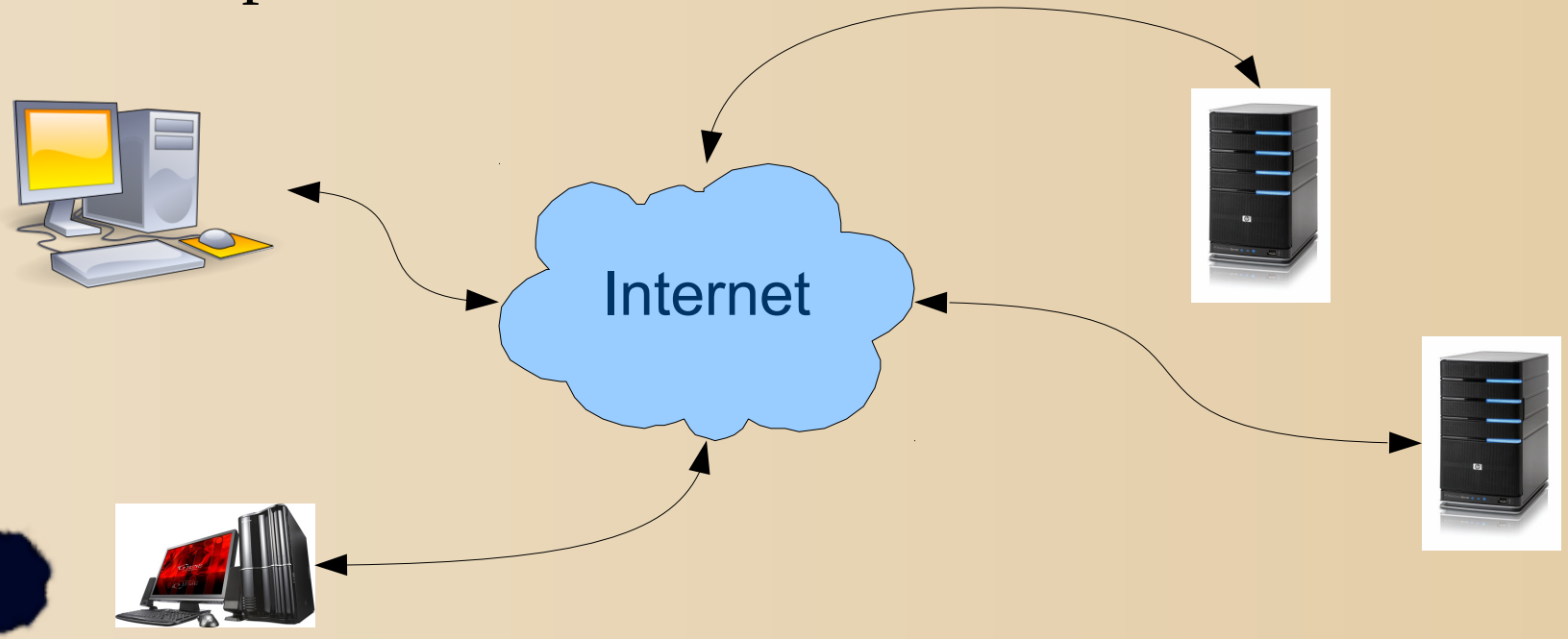
Conjunto dinámico de *relaciones* entre colecciones de información.



Ejemplo: La arquitectura de la Web

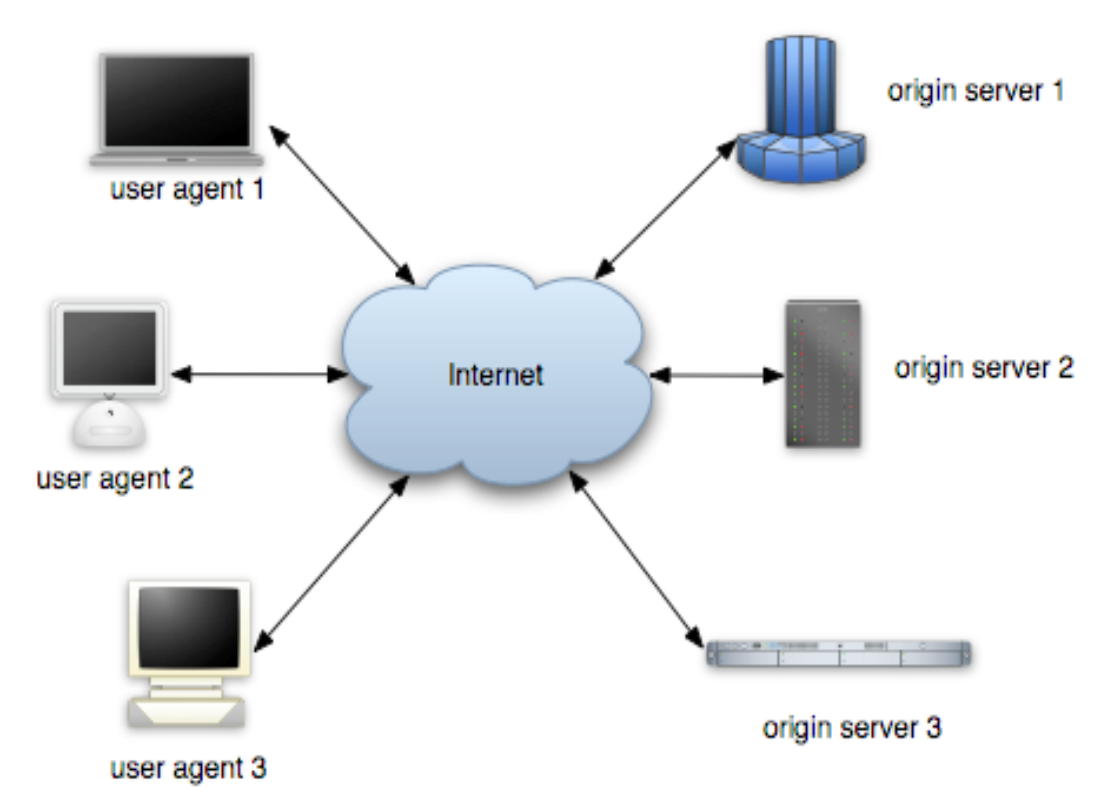
A alto nivel, desde un punto de vista *estructural*:

Colección dinámica de *máquinas independientes* dispersas en el mundo que *interactúan* mediante redes de computadoras.



Ejemplo: La arquitectura de la Web

A alto nivel, desde un punto de vista de *deployment*:

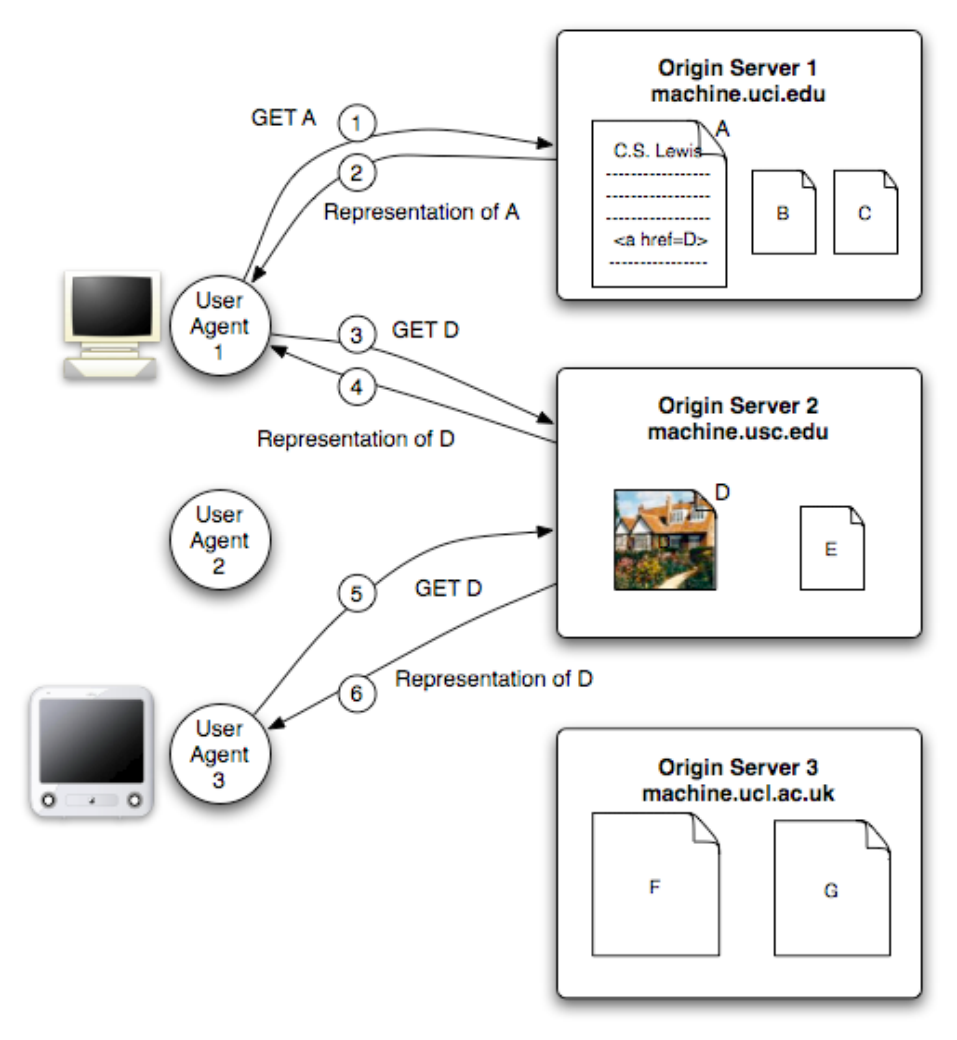


Ejemplo: La arquitectura de la Web

A alto nivel, desde el punto de vista de un *desarrollador de software*:

Colección de programas escritos en forma independiente que interactúan de acuerdo a reglas especificadas (protocolos), tales como TCP/IP, HTTP, HTML, URI, etc.

Ejemplo: La arquitectura de la Web



¿Son suficientes?

- Estos diagramas muestran *cómo funciona la Web*, pero no son suficientes para *explicar completamente* su funcionamiento.
- Puede ayudar tener una serie de *definiciones* y *restricciones*.

Restricciones

- La Web es una colección de recursos, cada uno de los cuales puede ser identificado por un URL.
- Un estándar especifica la sintáxis de los URL.
- Cada recurso denota *información*. La información puede ser un documento, imagen, etc.
- Los URL identifican a una máquina en Internet: un servidor donde se puede obtener un recurso.
- La comunicación es iniciada por los clientes (*user agents* en el esquema anterior).

Restricciones

- Los *Web browsers* son usualmente las instancias de user agents.
- Los recursos pueden ser manipulados a través de sus *representaciones*. HTML es el lenguaje de representación más común.
- Toda la comunicación se realiza de acuerdo al protocolo HTTP.
- Para esto, los participantes deben implementar algunas primitivas simples como *get* y *post*.

Observaciones

- Este acercamiento para “entender” la Web es claramente superior a intentar *analizar el código fuente*.
- La arquitectura de la Web está completamente *separada* del código que la implementa.
- La arquitectura es el conjunto de principales decisiones de diseño que determinan los elementos clave y sus relaciones.
- *Importante*: Puede haber muchas formas de implementar una arquitectura.

Observaciones

- El *estilo* restringe el código en algún sentido, pero existe mucha libertad.
- Las restricciones de estilo no son necesariamente aparentes en el código.
- Sin embargo, los *efectos* de las restricciones tal como hayan sido implementados en cada sistema son evidentes en la Web.

Algunos interrogantes

- ¿Por qué se tomaron estas decisiones particulares?
- ¿Por qué estas decisiones son importantes y otras no?
- ¿Por qué otros sistemas que tomaron decisiones parecidas fallaron, mientras que la Web fue un éxito?

Qué cambió

- El software actual es cada vez más complicado y difícil de desarrollar y mantener.
- Intervienen muchos aspectos, tales como:
  - Especificaciones
  - Equipo
  - Calidad
  - Proceso

Entonces...

- La arquitectura de software ayuda a enfrentar estos problemas.
- Al disponer de una documentación de diseño, es posible monitorear el estado del SW a *alto nivel*.
- Se puede *reutilizar*.
- Es difícil hacer un *seguimiento* de la arquitectura, ya que visualizarla requiere *experiencia* y *criterio*.

Bibliografía

- R. Pressman: "*Software Engineering: A Practitioner's Approach*", 8th ed. McGraw-Hill, 2014. (Capítulo 12)
- R. N. Taylor, N. Medividovic, E. Dashofy: "*Software Architecture: Foundations, Theory, and Practice*". Wiley, 2009. (Capítulos 1 y 2)

Otra bibliografía

- H. van Vliet: "*Software Engineering: Principles and Practices*", 3rd ed., 2008. (Capítulos 10, 11 y 12)