Diseño de Software

Semana 1

Agenda

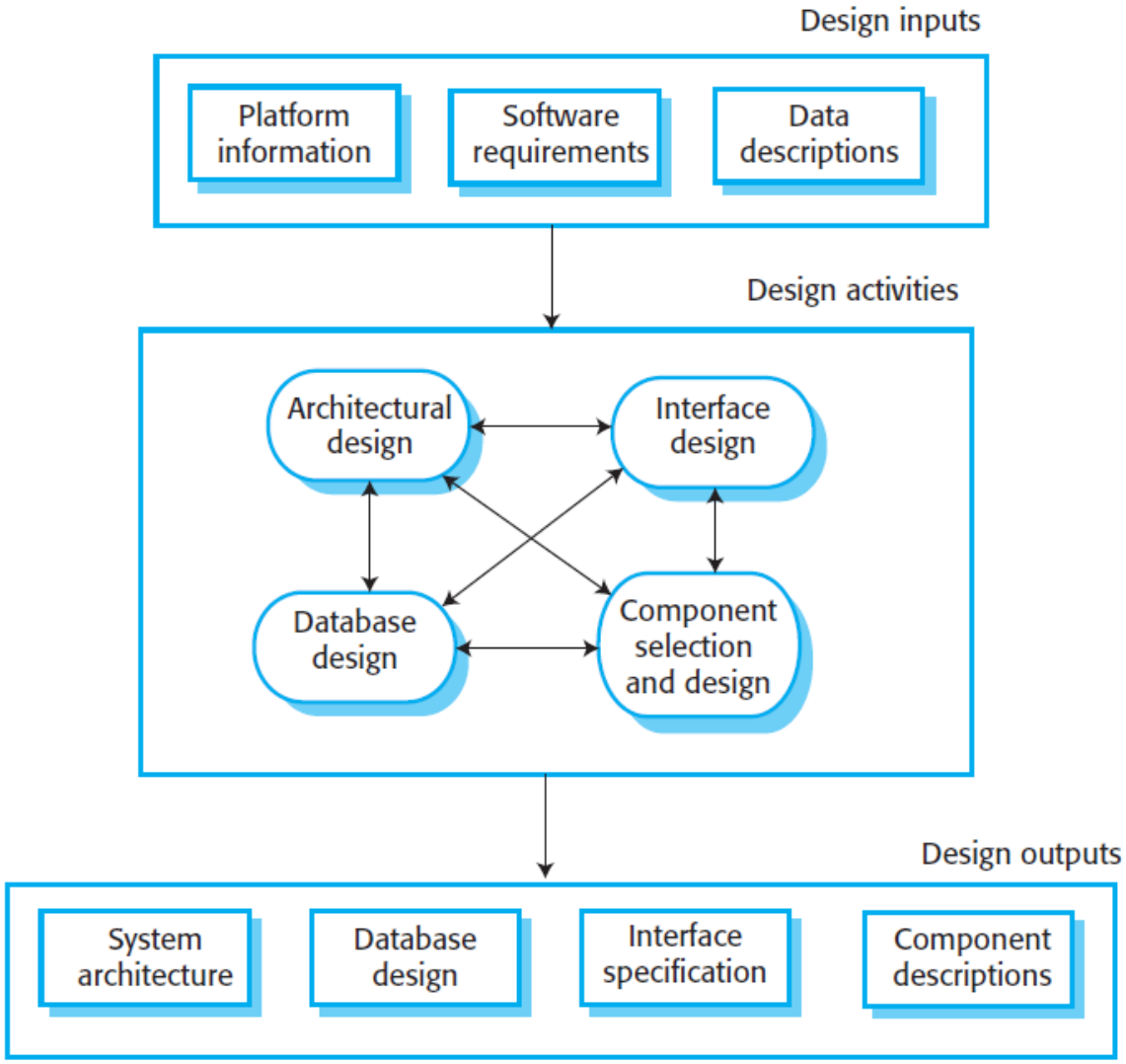
- Conceptos de diseño de software
- Diseño arquitectural vs. Diseño detallado
- Estilos arquitectónicos de software
- Control de versiones de software

Conceptos de diseño de software

¿Qué es diseño de software?

- Se puede referir a dos cosas: a un proceso o al resultado de este.
- “Diseño de software es un **proceso** que define la **arquitectura, componentes, interfaces** y otras características de un sistema o un componente”.
  - Software Engineering Body of Knowledge (SWEBOK)
- “Un diseño de software es una **descripción de la estructura del software** a ser implementado, **los modelos de datos** y estruturas usadas por el sistema, las **interfaces entre los componentes** del sistema y, algunas veces, los **algoritmos** utilizados”.
  - Ian Sommerville

Un modelo general de proceso de diseño

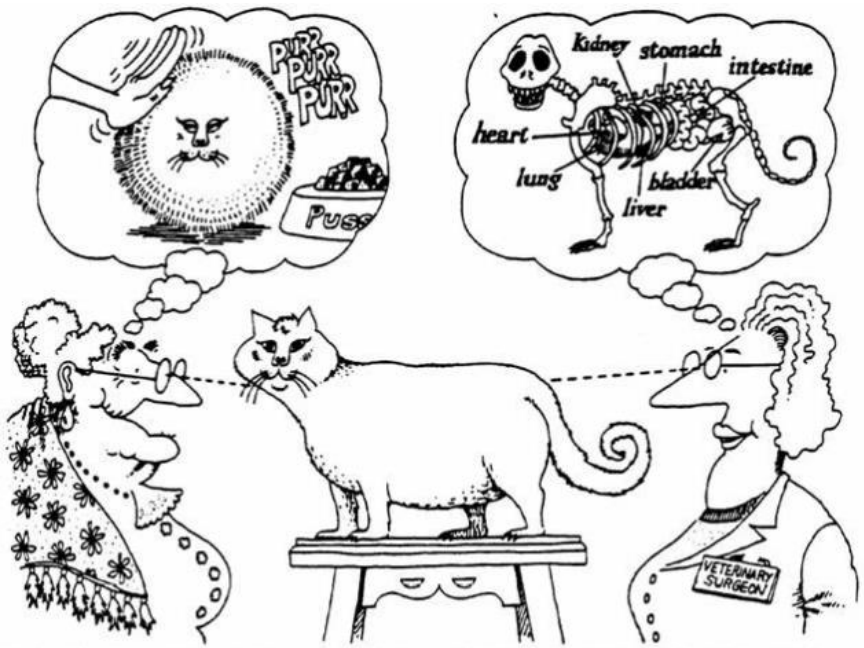


Principios de diseño de software

- Los principios de diseño de software son nociones **fundamentales** que proveen las bases para **diferentes enfoques** de diseño de software. Son **siete**:
  - abstracción;
  - cohesion y acoplamiento;
  - descomposición y modularización;
  - encapsulamiento/ocultamiento de información;
  - separación de interfaz e implementación;
  - suficiencia, completitud y primitivismo;
  - y separación de preocupaciones.

Principios de diseño de software

1. **Abstracción** se refiere a la vista de un objeto enfocándose en la información relevante a un propósito en particular e ignorando el resto de la información.



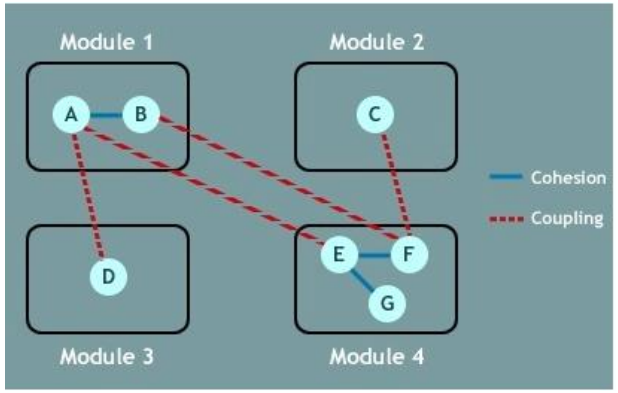
Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Cortesía: Hayim Makabee, The Cat as a Metaphor in Object-Oriented Software Development

Principios de diseño de software

2. **Acoplamiento** se define como el **grado de interdependencia o interconexión entre los módulos** de un software, mientras que **cohesión** se define como el **grado de relación entre las responsabilidades asignadas a un módulo**.
   - Si hay alta interdependencia, entonces los cambios en un módulo tendrán efectos significativos sobre el comportamiento de los demás.
   - Si un módulo es responsable de un número de funciones no relacionadas, entonces las funcionalidades no han sido bien distribuidas entre los módulos.

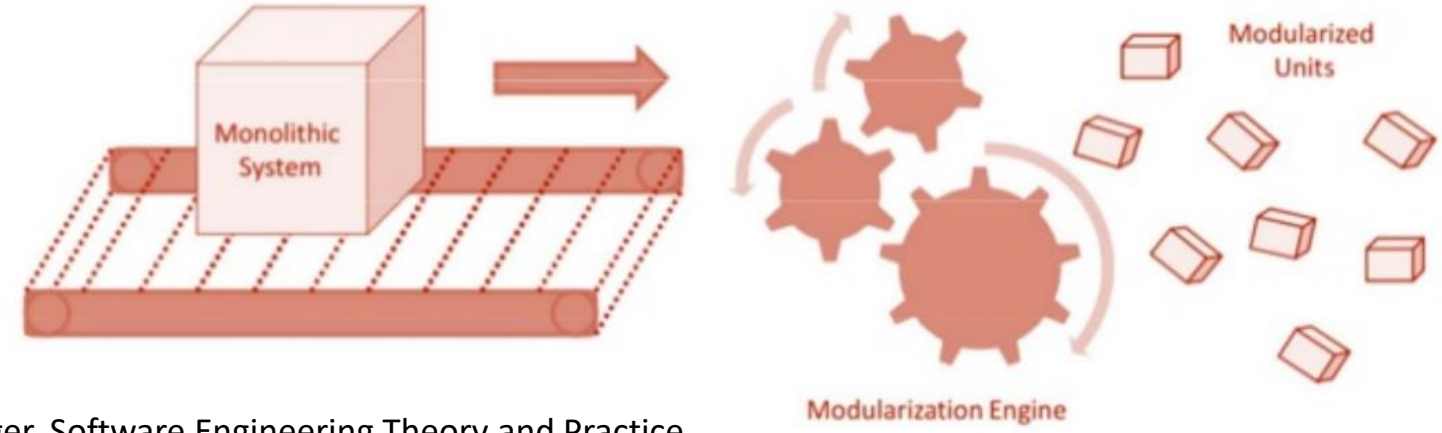
Cortesía: Masudur Rahman, Recommendation of Move Method Refactorings Using Coupling, Cohesion and Contextual Similarity



   - Ideal: **alta cohesión y bajo acoplamiento**.

Principios de diseño de software

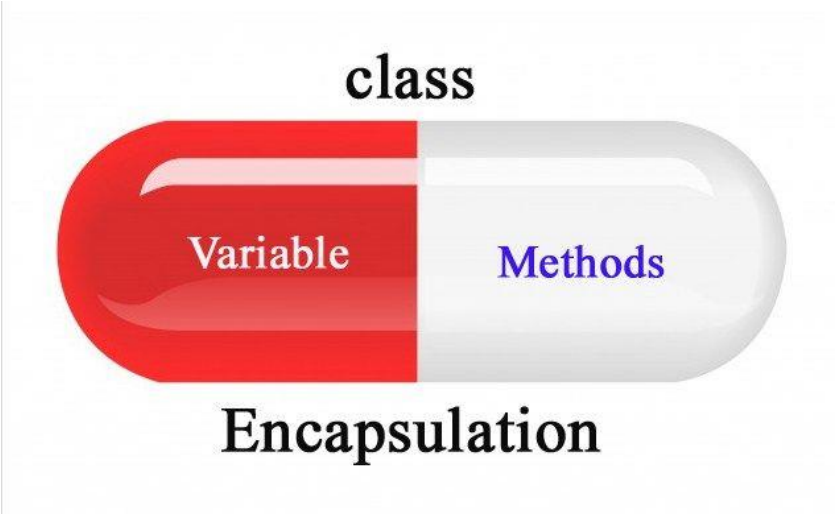
3. **Descomposición y modularización** significa que un software es dividido en un número de componentes más pequeños con interfaces bien definidas que describen las interacciones entre los componentes.
   - Usualmente, el objetivo es ubicar diferentes funcionalidades y responsabilidades en diferentes componentes.



Cortesía: Pfleeger, Software Engineering Theory and Practice

Principios de diseño de software

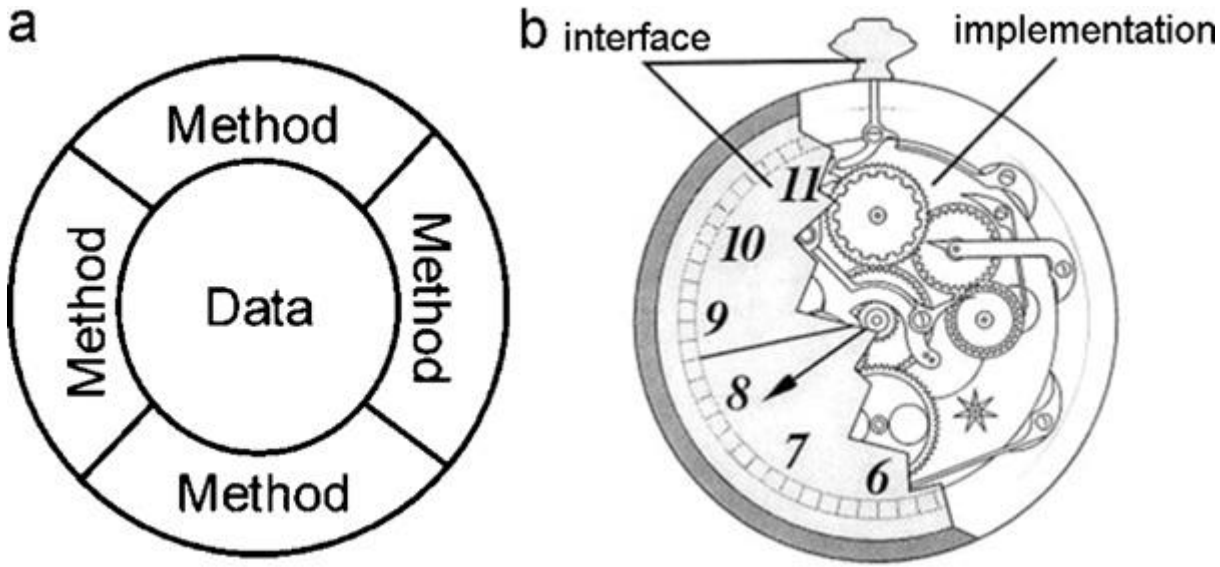
4. **Encapsulamiento y ocultamiento** de información significa agrupar y empaquetar los detalles internos de una abstracción y hacer esos detalles inaccesibles a entidades externas.



Cortesía: Sathasivam Karmehavannan, Encapsulation in Java programming language - Codeforcoding

Principios de diseño de software

5. **Separación de interfaz e implementación** implica definir un componente especificando una interfaz pública (conocida para los clientes) que está separada de los detalles de cómo el componente está implementado.



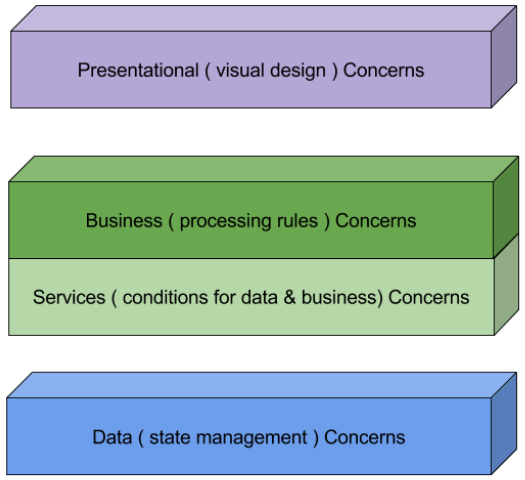
Cortesía: Li An, Modeling Human Decisions in Coupled Human and Natural Systems: Review of Agent-Based Models

Principios de diseño de software

6. **Suficiencia, completitud y primitivismo**: Alcanzar suficiencia y completitud significa asegurar que un componente de software **captura todas las características** importantes para alcanzar su objetivo, no más y no menos. Primitivismo significa que el diseño debería estar basado en patrones que sean de fácil implementación.

Principios de diseño de software

7. **Separación de preocupaciones**: Una preocupación (concern) es un área de interés con respecto al diseño de un software. La arquitectura de un software debe tener una o varias vistas de preocupaciones que permitan a los interesados concentrarse en pocas cosas a la vez a fin de reducir la complexidad.

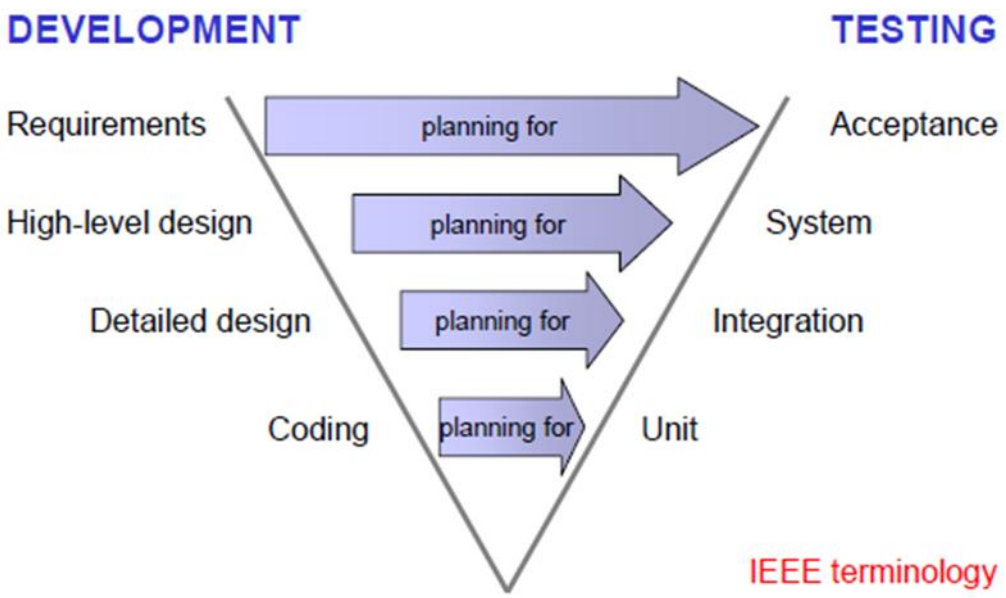


Cortesía: Willie Streeter, Practical Web Development and Architecture

Diseño arquitectural vs. Diseño detallado

Diseño de software

- Dos niveles de diseño de software:
  1. Diseño de alto nivel o arquitectural
  2. Diseño detallado

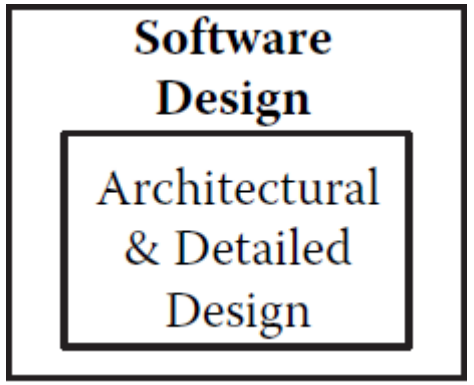


Diseño de alto nivel o arquitectural

- La palabra arquitectura es a menudo utilizada en el contexto de algo a alto nivel, es decir, está separado de detalles a bajo nivel.
- Es un enfoque de **macro diseño** para crear modelos que ilustren aspectos de calidad y funciones del sistema de software.
- ¿Qué se decide en el diseño arquitectural de un software?
  - **Plataforma** tecnológica
  - **Despliegue** físico del sistema de software, incluyendo subsistemas ubicados en diferentes lugares.
  - Selección de los principales **componentes estructurales**
  - La forma en la que el sistema de software, como un todo, se va a **comunicar** (ej.: protocolos de comunicación)
  - Problemas de **concurrencia**
  - Se evalúa **requerimientos** no funcionales (ej.: desempeño, seguridad, robustez, escalabilidad)
  - Entre otros

Diseño detallado

- Se **refina** el diseño arquitectural a un punto en el que el **diseño** está lo suficientemente **completo** como para **empezar** la **construcción** del software.
- ¿Qué se decide en el diseño detallado de un software?
  - Se refina componentes en clases
  - Se implementan interfaces
  - Se especifica las relaciones entre clases
  - Se identifica y aplica patrones de diseño
  - Se diseña componentes y sus interfaces



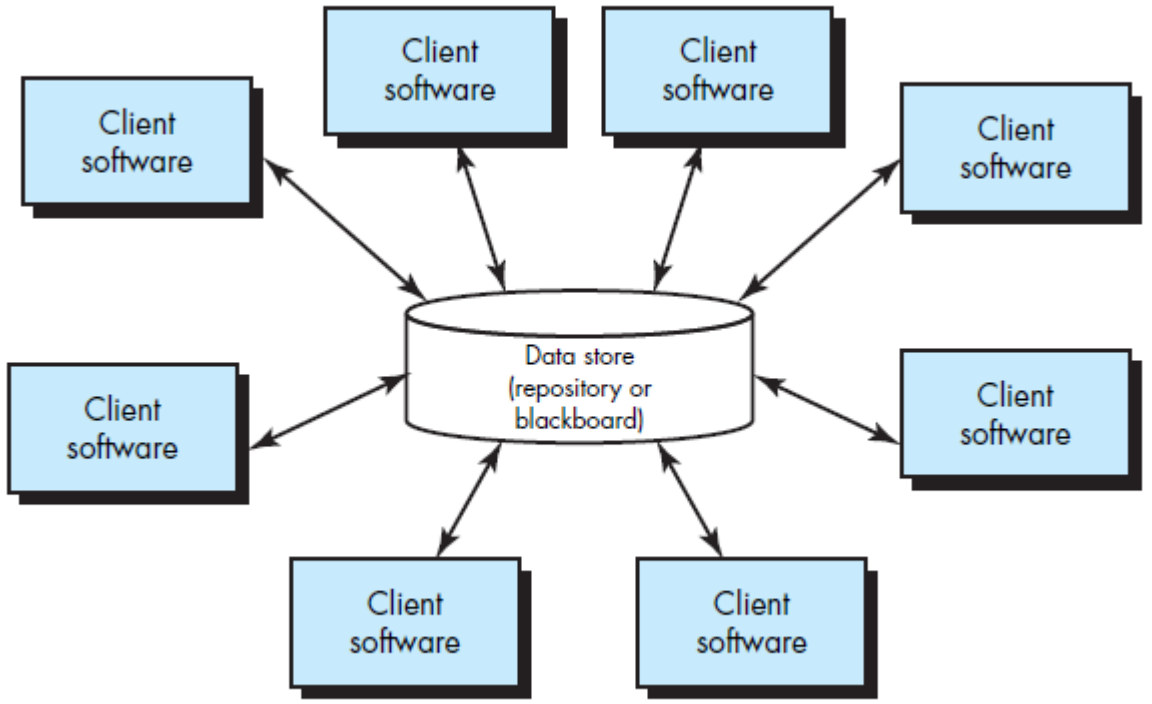
Estilos arquitectónicos

Estilos arquitectónicos

- En construcción, se utiliza estilo arquitectónico como un mecanismo descriptivo para diferenciar los diferentes **estilos de construcciones**.
- Cada estilo arquitectónico de software incluye:
  1. un conjunto de **componentes** que llevan a cabo la función requerida por el sistema;
  2. un conjunto de **conectores** que permiten la **comunicación, coordinación y cooperación** entre componentes;
  3. **restricciones** que definen cómo se **integran** los componentes para formar el sistema; y
  4. modelos **semánticos** que permiten al diseñador entender las propiedades del sistema.

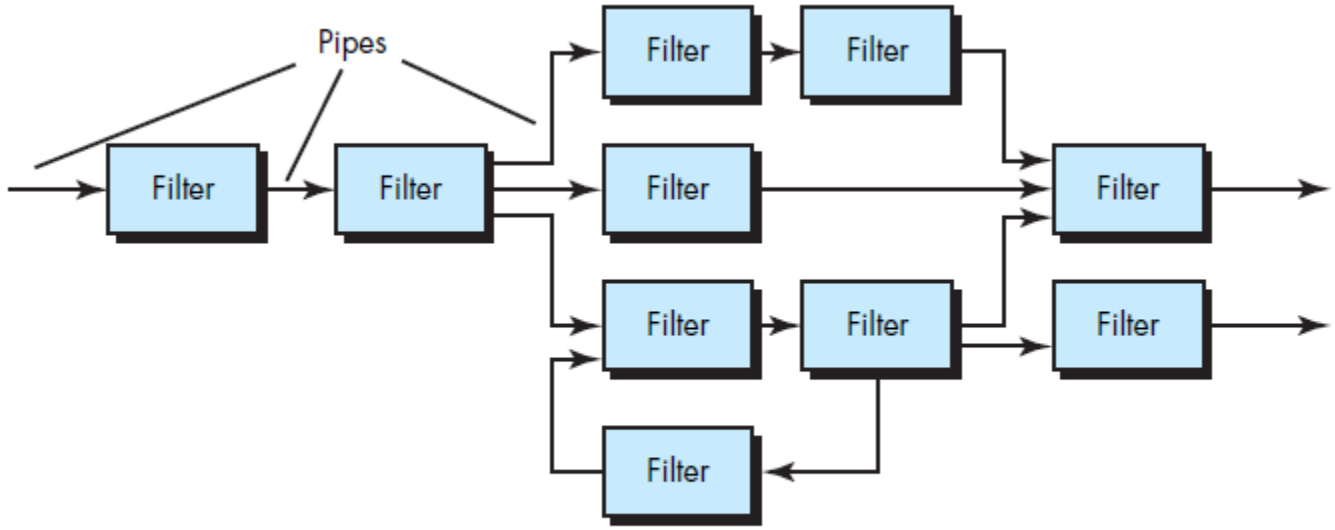
Estilos arquitectónicos

1. Arquitectura centrada en datos
   - Un repositorio de datos es accedido frecuentemente por otros componentes para actualizar, agregar, eliminar o modificar datos en el repositorio.



Estilos arquitectónicos

2. Arquitectura de flujo de datos
   - un dato de entrada es transformado en un dato de salida a través de una serie de componentes de manipulación o cálculo.

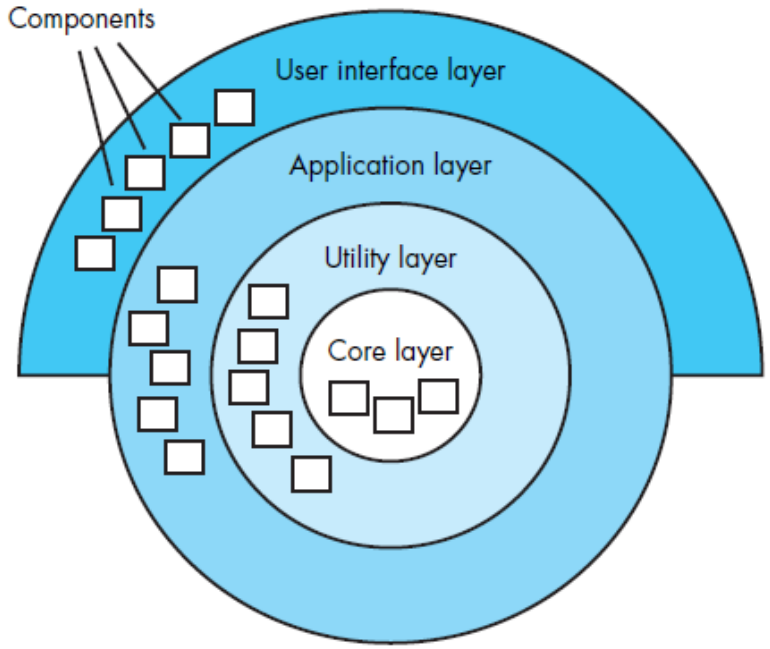


Pipes and filters

Estilos arquitectónicos

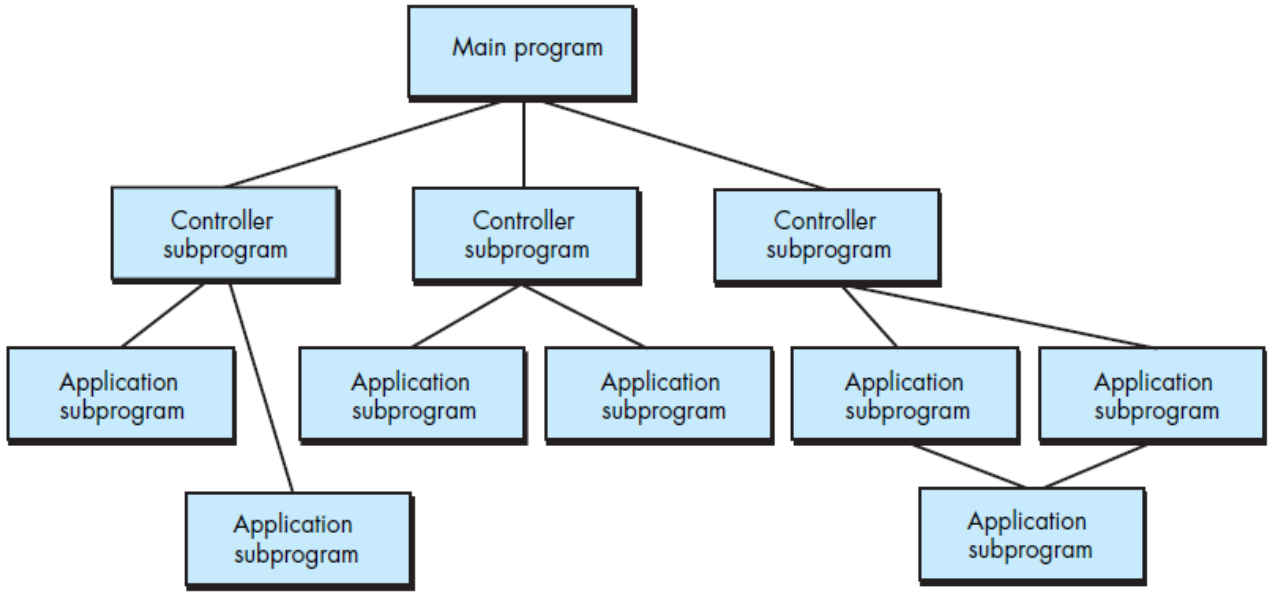
3. Arquitectura en capas

- Se define varias capas y cada una de ellas cumple operaciones que progresivamente se acercan al conjunto de instrucciones de máquina.
- En la capa más exterior, los componentes hacen operaciones de interfaz de usuario.
- En la capa más interior, componentes llevan a cabo operaciones de interfaz hacia el sistema operativo.
- Las capas intermedias proveen servicios utilitarios y funcionales.



Estilos arquitectónicos

4. Arquitectura de llamada y retorno
   - Se centra en la creación de una estructura de programa que sea fácilmente modificable y escalable. Principales subcategorías:
     - Arquitectura de programa principal /subprograma
     - Arquitectura de llamada a procedimiento remoto



Estilos arquitectónicos

- Arquitectura orientada a objetos.
  - Los componentes de un sistema encapsulan los datos y las operaciones que se deben aplicar para manipular los datos.
  - La comunicación y coordinación entre componentes debe ser realizada por medio de paso de mensajes.

Control de versiones de software

Control de versiones de software

- Manejo de versiones es el proceso de mantener un registro de las diferentes versiones de los componentes de un software y los sistemas en los cuales estos componentes se usan.
- El manejo de versiones implica asegurar que los cambios hechos por diferentes desarrolladores a estas versiones no interfieren con la anterior.
- Los sistemas de control de versiones automatizan el manejo de versiones.

Control de versiones de software

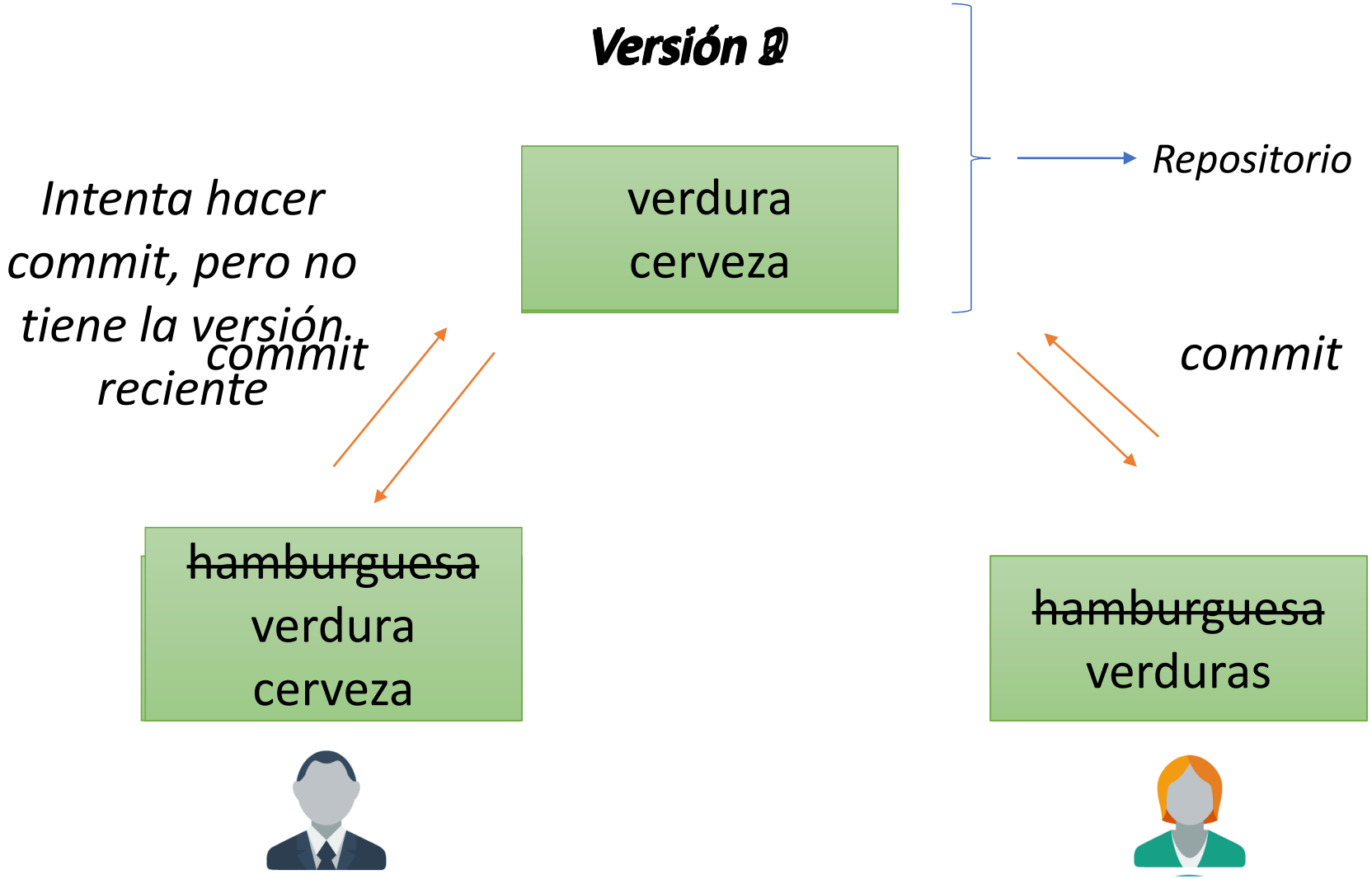
- Ventajas:
  - Almacenamiento y Backup
  - Control de acceso mediante permisos
  - Deshacer ‘ilimitado’
  - Combinar aportaciones de distintos colaboradores
  - Sincronización de desarrolladores
  - Histórico de cambios
  - Versiones en paralelo

Control de versiones de software

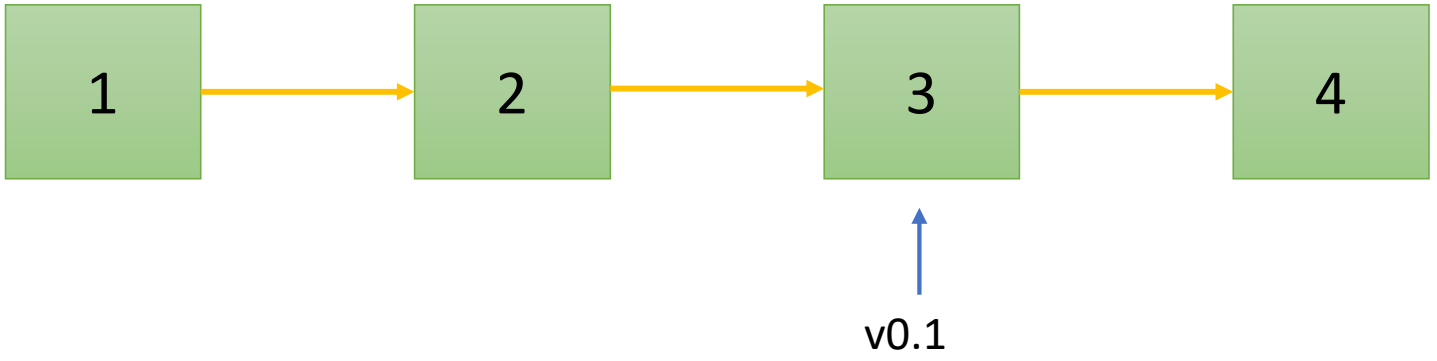
Control de versiones de software

- **Vocabulario básico:**
  - **Repositorio**: Almacén de datos que guarda cada versión de nuestro proyecto, incluyendo los datos asociados a cada commit.
  - **Commit**: Cambio de una versión a otra

Ejemplo: Lista de Compras



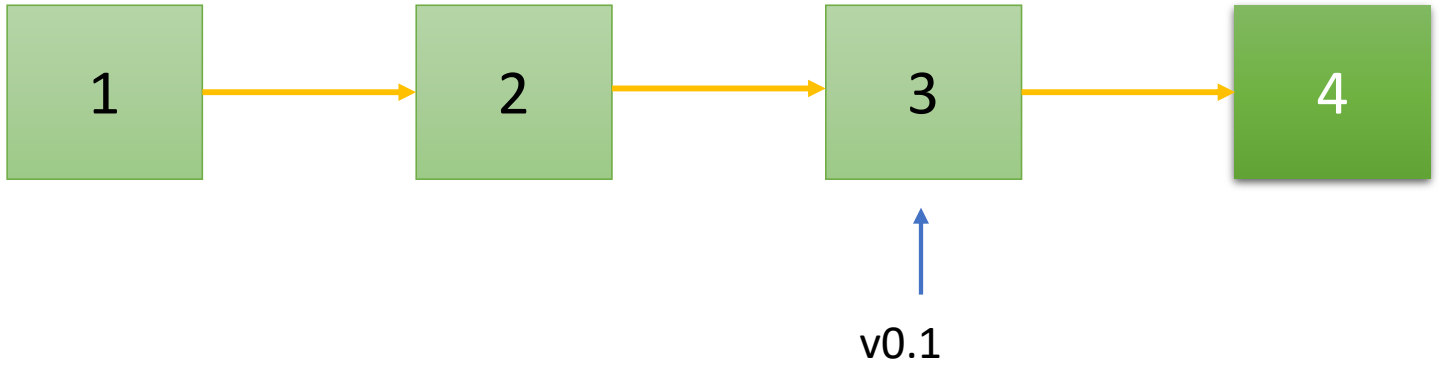
Control de versiones de software



Se pueden etiquetar versiones concretas para localizarlas fácilmente en la historia del repositorio.

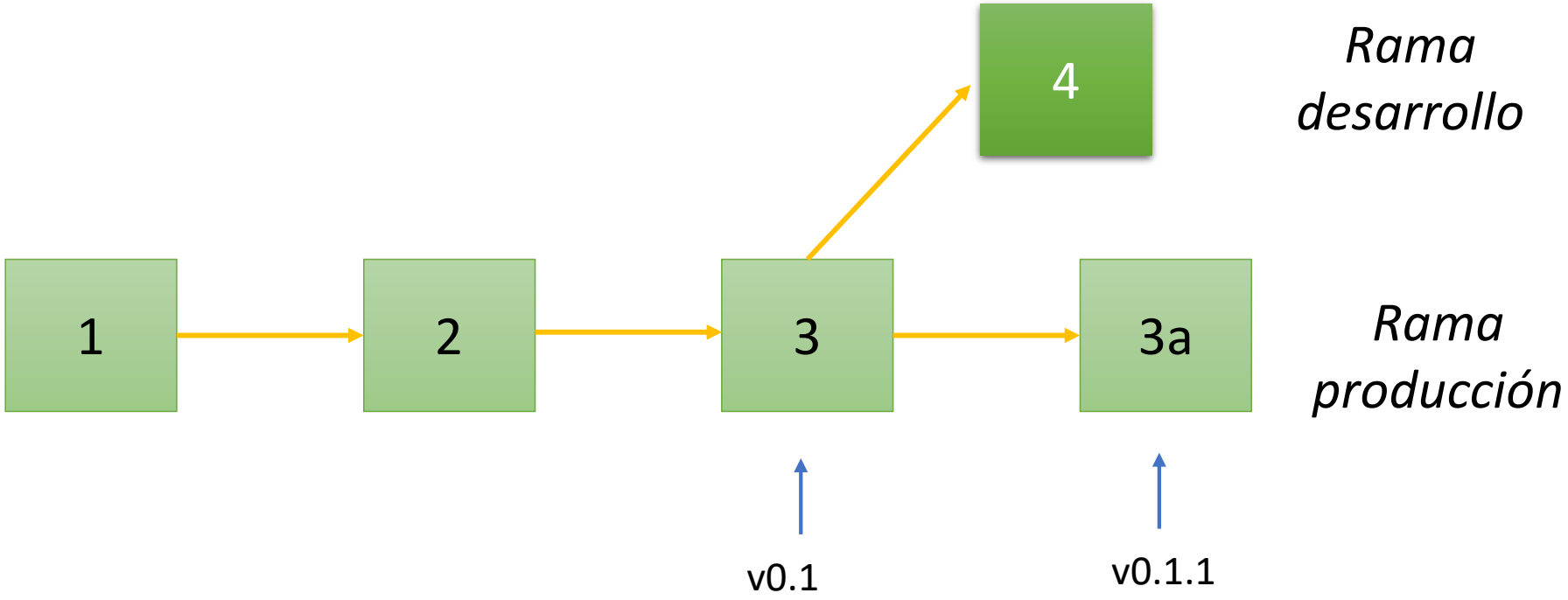
Control de versiones de software

- ¿Qué ocurre si estamos desarrollando la funcionalidad 4 y se descubre un bug en alguna de las anteriores?



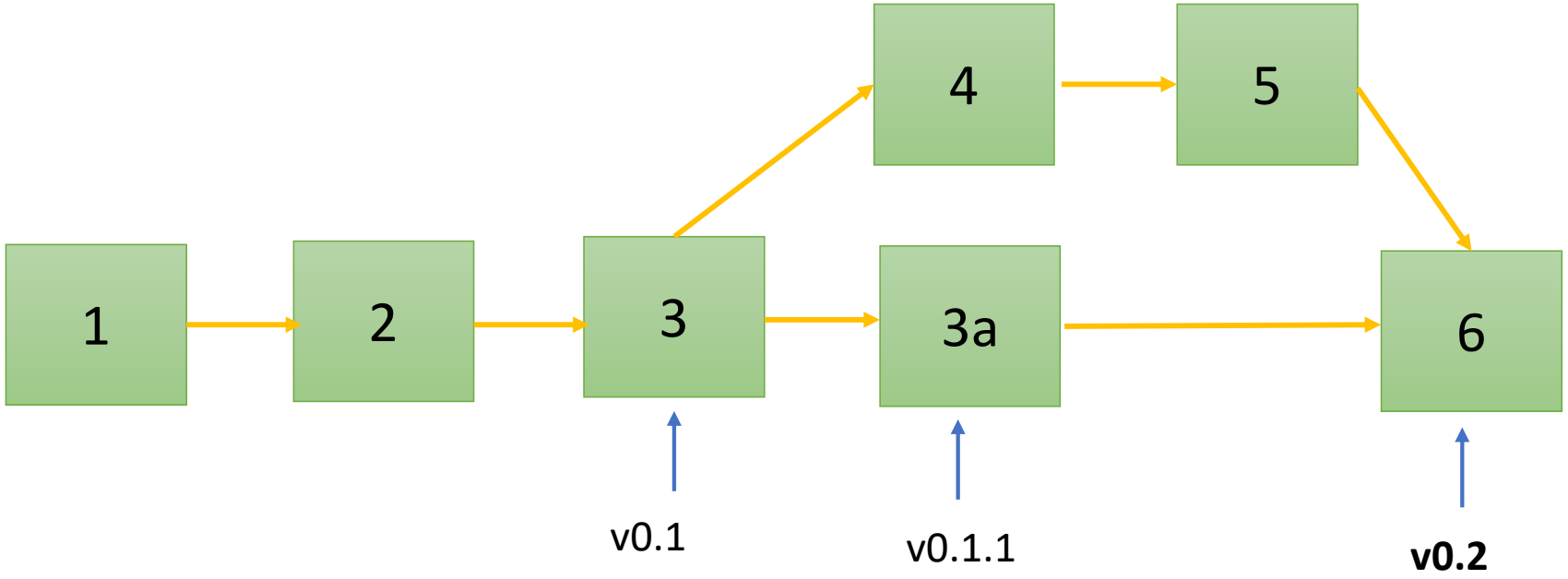
Control de versiones de software

- Para evitar esto se puede trabajar en "ramas"



Control de versiones de software

- Cuando la rama de desarrollo sea estable, la fusionaremos con la de producción



Git

Git

- Es un sistema de versionamiento distribuido.
- Aunque se trabaje con un repositorio remoto, siempre se tiene uno local.

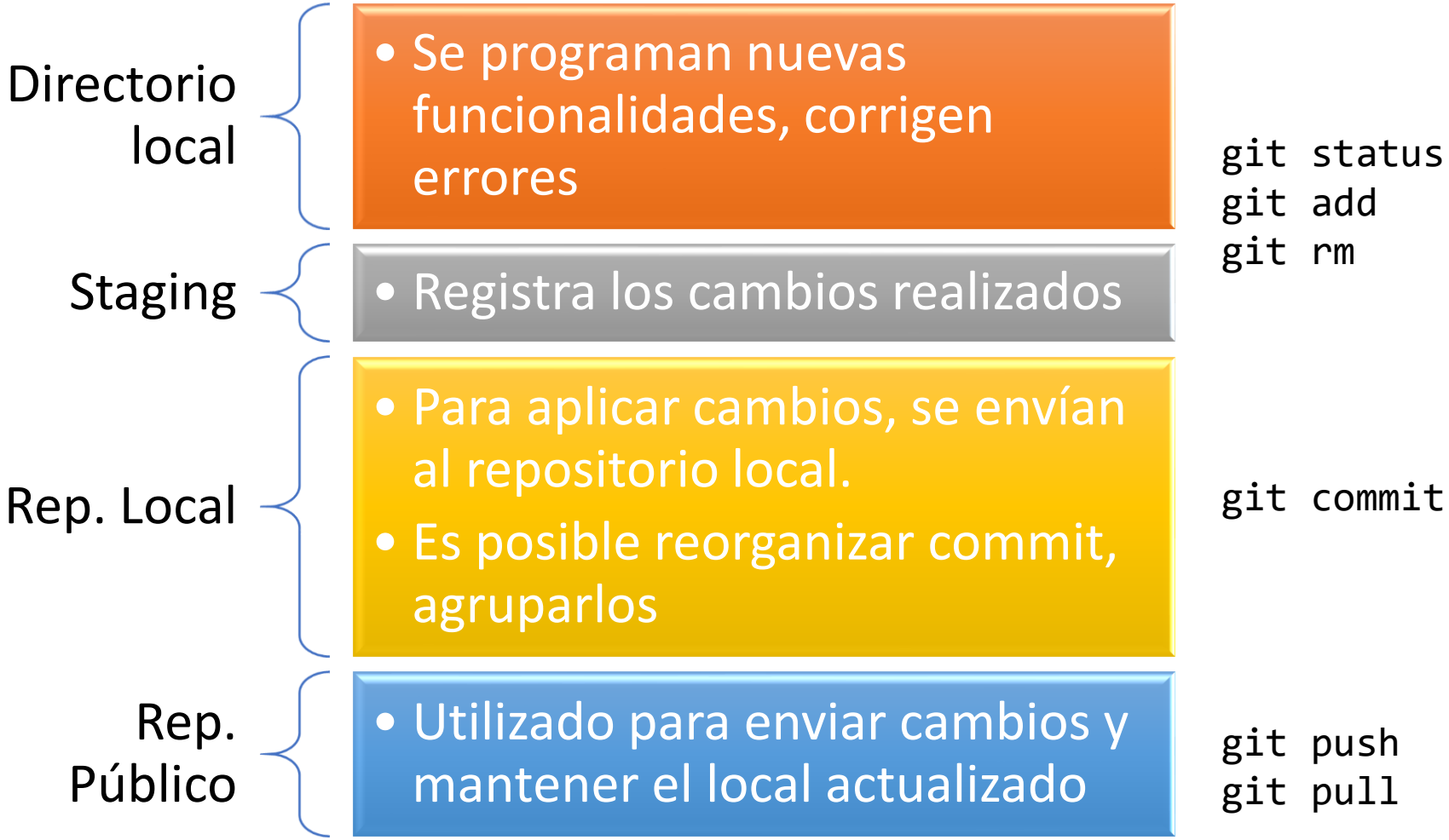
Directorio local de trabajo

Área de cambios (staging)

Repositorio local (privado)

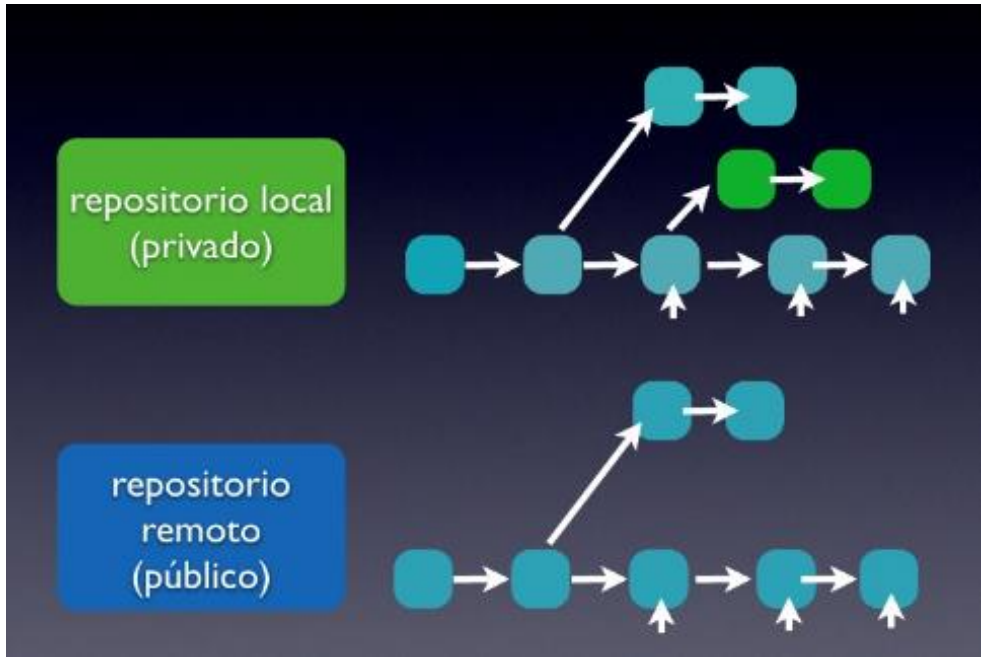
Repositorio remoto (público)

Git


Directorio local
• Se programan nuevas funcionalidades, corrigen errores
git status
git add
git rm
Staging
• Registra los cambios realizados
Rep. Local
• Para aplicar cambios, se envían al repositorio local.
• Es posible reorganizar commit, agruparlos
git commit
Rep. Público
• Utilizado para enviar cambios y mantener el local actualizado
git push
git pull

Ramas (Branches)

- Se pueden llevar ramas para desarrollar en local, otras en remoto para manejar las versiones del programa, sincronizarlas entre sí…



GitHub

- Acceso web a repositorios Git.
- Gratuito para proyectos libres.
- Antepasados: sourceforge, Savannah, gforge
- “Red social” de programadores.
- Currículo para las empresas.

Antes de finalizar

Puntos para recordar

- ¿Qué es diseño de software?
- ¿Qué se decide en el diseño de alto nivel y en el detallado?
- ¿Cuáles son los principios de diseño de software?
- Descripción de los principales estilos arquitectónicos
- Conceptos básicos de control de versiones

Lectura adicional

- IEEE, “Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK)”
  - Chapter 2: Software Design
- Pressman and Maxin, “Software Engineering”
  - Chapter 12: Design Concepts
  - Chapter 13: Architectural Design
- Robert Martin, Clean Architecture
  - Chapter 1: What is Design and Architecture
- Gui and Scott, “Measuring Software Component Reusability by Coupling and Cohesion Metrics”
- Ian Sommerville, “Software Engineering”
  - Chapter 2: Software Processes

Próxima sesión

- DevOps y Entrega Continua