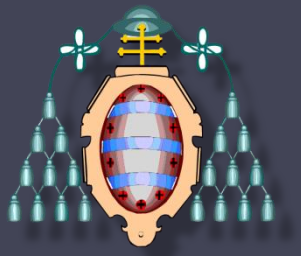
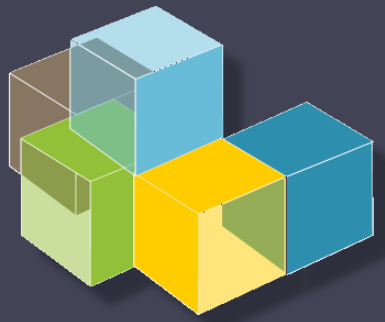
ES
Español



Universidad de Oviedo

Escuela de Ingeniería Informática



ARQUITECTURA DEL SOFTWARE

Arquitectura del Software

Clase 3 a
Tema 2: Disposición

Escuela de Ingeniería Informática

2013

Jose Emilio Labra Gayo

Disposición (Allocation)

Relación del software con el entorno
Construcción, despliegue y distribución

Esquema

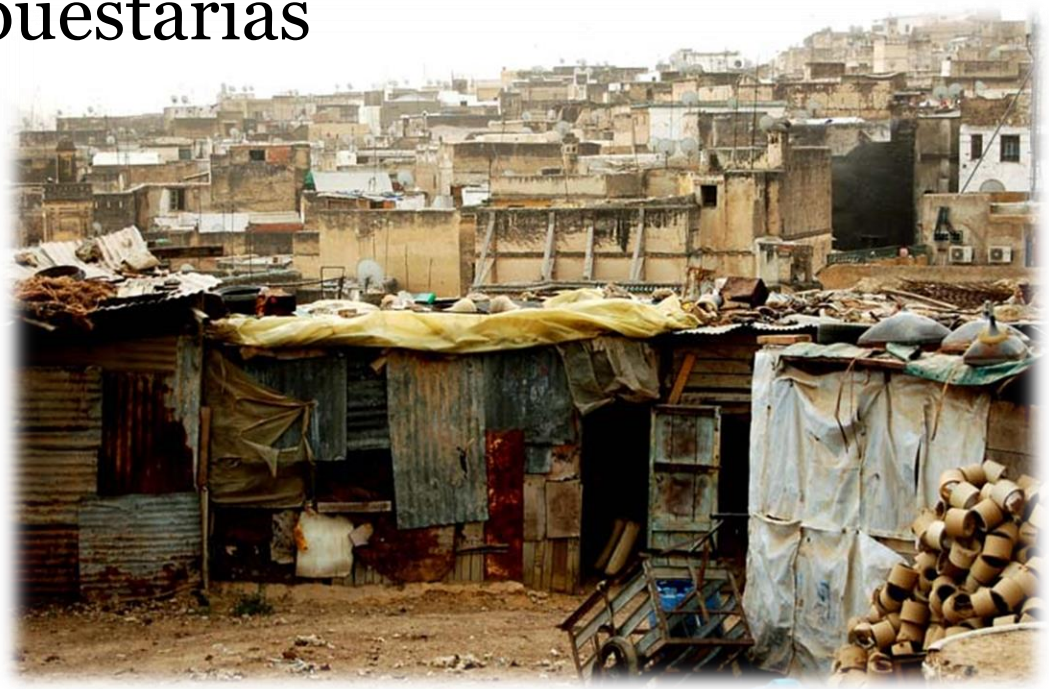
- Construcción
  - Estilos de desarrollo
    - Tradicionales, iterativos, ágiles
  - Gestión de configuraciones
    - Control de versiones
    - Gestión de dependencias
    - Despliegue e integración continua
- Distribución
  - Canales de distribución

Estilos de desarrollo

Nota: Se ha incluido una selección. Puede consultarse una extensa lista en: http://en.wikipedia.org/wiki/List_of_software_development_philosophies

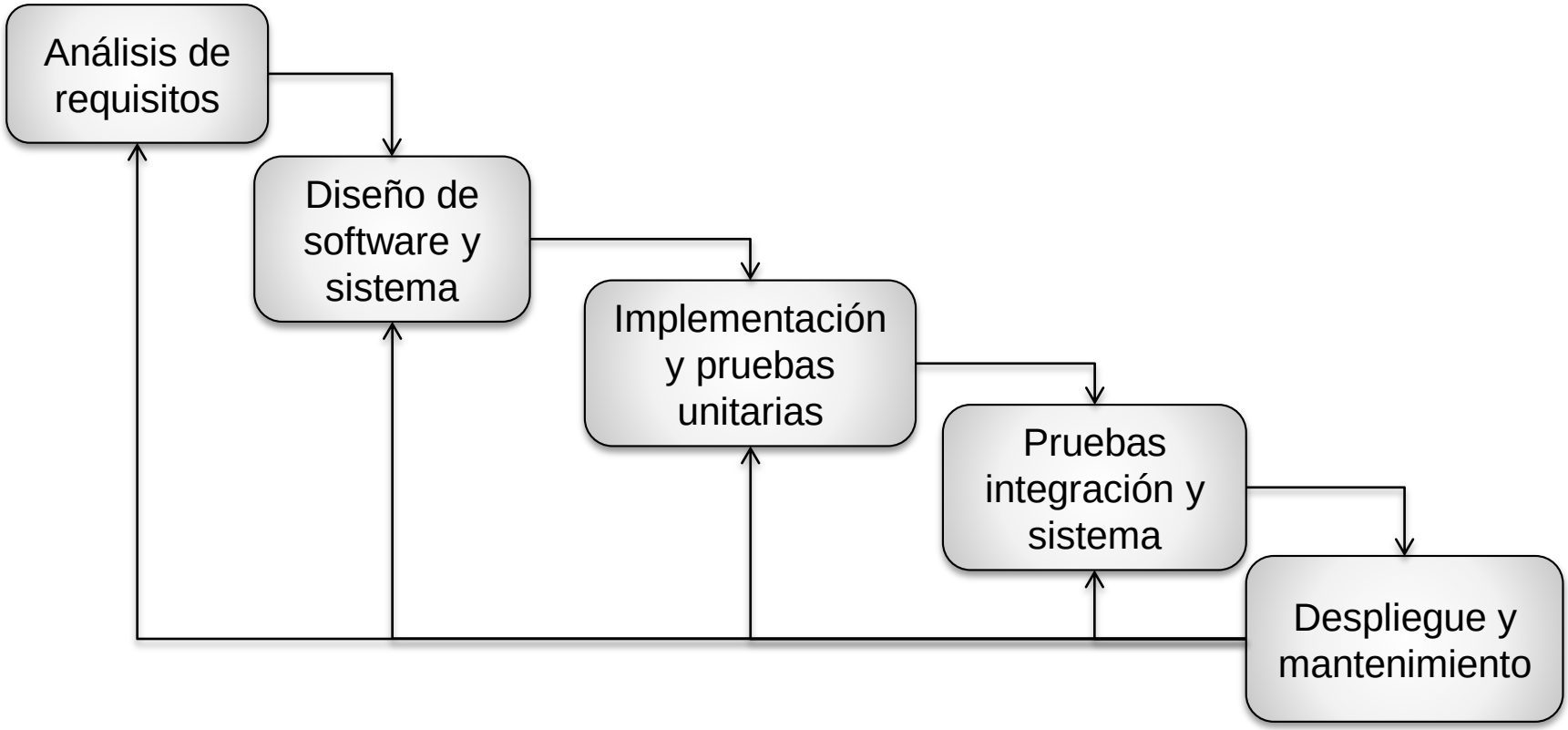
Incremental piecemeal

Crecimiento según necesidad
Codificar sin considerar la arquitectura
Software de usar y tirar
Limitaciones presupuestarias

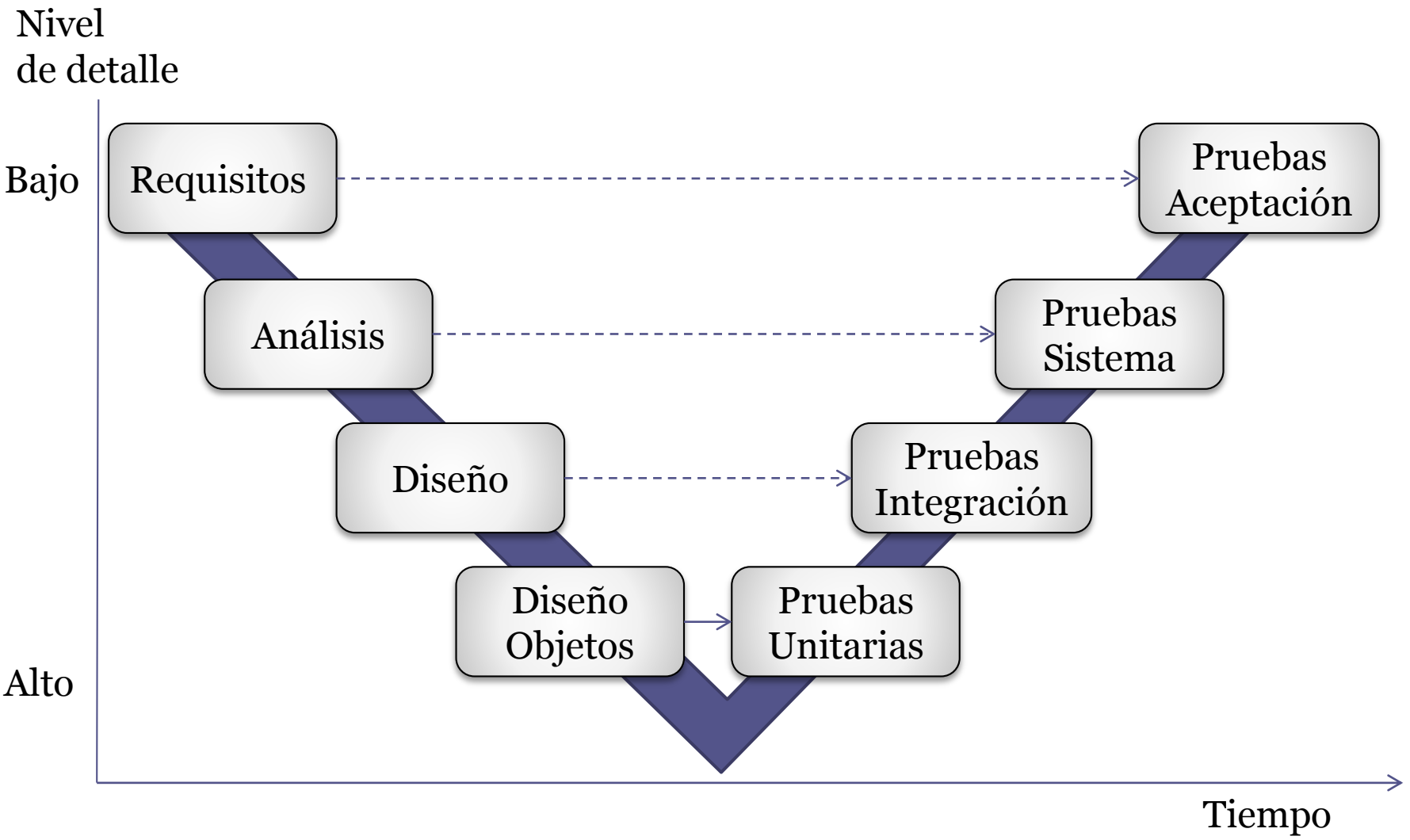


Cascada

Propuesto en años 70



Modelo en V


Nivel
de detalle
Bajo
Alto
Requisitos
Análisis
Diseño
Diseño
Objetos
Pruebas
Unitarias
Pruebas
Integración
Pruebas
Sistema
Pruebas
Aceptación
Tiempo

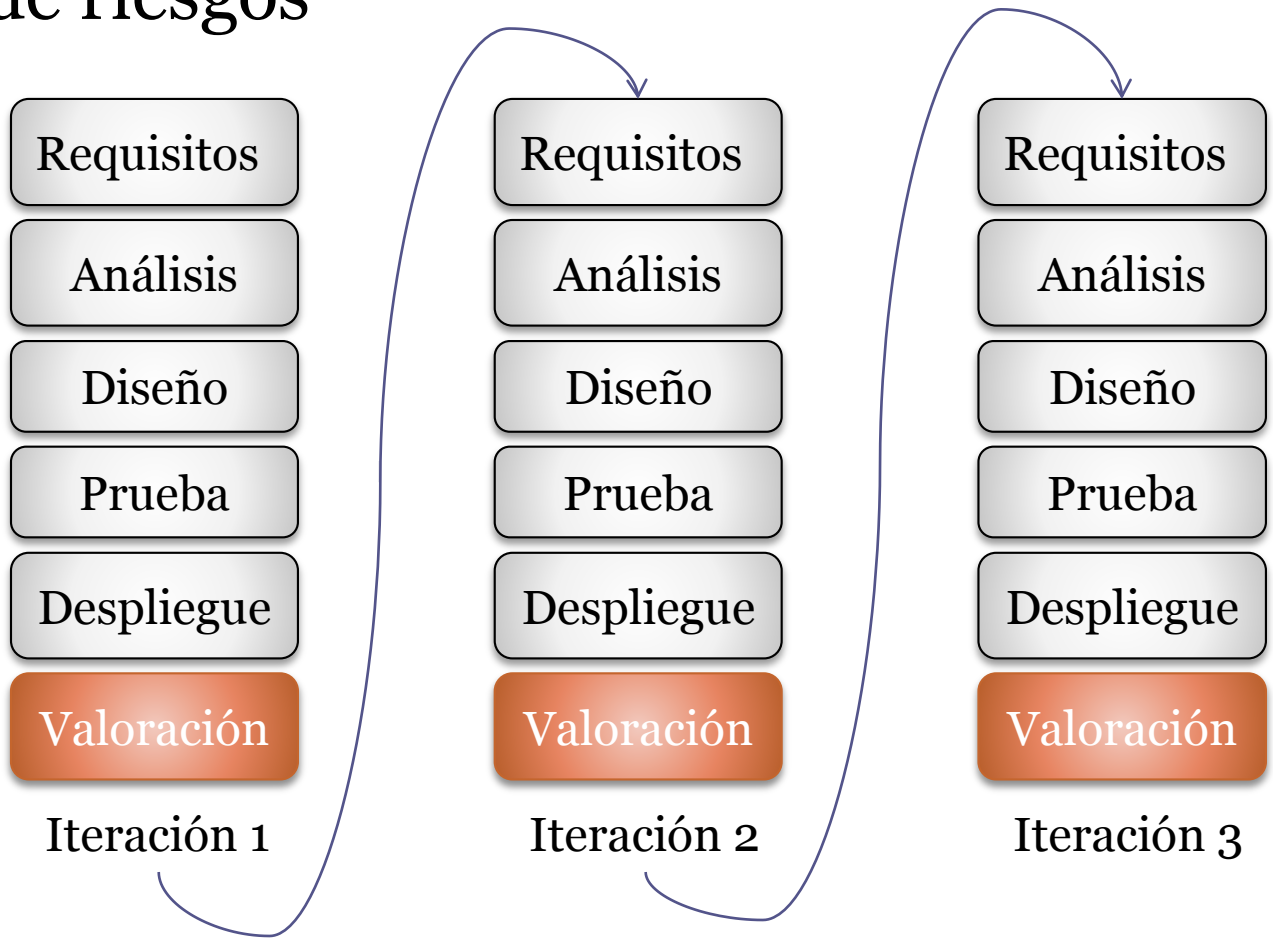
Big Design Up Front

Antipatrón de modelos tradicionales

- Demasiada documentación que nadie lee
- Documentación diferente al sistema desarrollado
- Arquitectura degradada
- Sistemas que no son usados

Modelos iterativos

Basado en prototipos
Evaluación de riesgos



Métodos ágiles

Algunas prácticas (XP)

1. Planificaciones cortas
2. Pruebas
3. Programación en parejas (revisiones de código)
4. Refactorización
5. Diseño simple
6. Propiedad de código compartida
7. Integración continua
8. Cliente en lugar de desarrollo
9. Entregas pequeñas
10. Horarios *normales*
11. Estándares de codificación

Gestión de configuraciones

Gestión de configuraciones

- Diferentes versiones de software
  - Funcionalidades nuevas o diferentes
  - Corrección de *bugs*
  - Nuevos entornos de ejecución
- Gestión de configuraciones: gestión de la evolución del software
  - Cambios del sistema = actividades en equipo
  - Costes y esfuerzo necesarios

Control de versiones

- Gestionar diferentes versiones software
  - Acceso a todas las versiones del sistema
    - Facilidad para volver atrás
    - Diferencias entre versiones
  - Código colaborativo
  - Facilidad para gestión de ramificaciones
  - Metadatos
    - Autor de la versión, fecha actualización, etc.

Releases y versiones

Versión: instancia de un sistema funcionalmente distinta de otras instancias

Release (entregable): instancia de un sistema que es distribuida a usuarios externos al equipo de desarrollo.

Puede ser considerado un producto final

Nombres habituales de versiones

- Pre-alfa
  - Antes de las pruebas
- Alfa
  - En pruebas
- Beta (o prototipo)
  - Pruebas por usuarios
    - Beta-tester: usuario que hace pruebas
- Release-candidate
  - Versión beta que podría ser producto final

Otros esquemas de nombres

- Utilizar algunos atributos
  - Fecha, creador, lenguaje, cliente, estado,...
- Nombres reconocibles
  - Ganimede, Galileo, Helios, Indigo, Juno,...
  - Precise Pangolin, Quantal Quetzal,...
- Versioneado semántico (http://semver.org)
  - MAJOR.MINOR.PATCH (2.3.5)
    - MAJOR: cambios incompatibles con versión anterior
    - MINOR: nueva funcionalidad compatible con versión anterior
    - PATH: Reparación de bugs compatible con versión anterior
  - Versión 0 (inestable)
  - Pre-release: 2.3.5-alpha

Publicación de entregables

Una *release* supone cambios de funcionalidad

Planificación

- Publicar una *release* no es barato
- Los usuarios no suelen querer nuevas *releases*
- Factores externos:
  - Marketing, clientes, hardware, …
- Modelo ágil: *releases* my frecuentes
  - Utilizando integración continua se minimiza el riesgo

Publicación de entregables

Una release no es sólo software
- Ficheros de configuración
- Ficheros de datos necesarios
- Programas de instalación
- Documentación
- Publicidad y empaquetamiento

Distribución: medios físicos (CDs, DVDs), Web (descargas), stores

Continuous delivery

- Continuous delivery/entrega continua
  - Entregas rápidas para obtener feedback lo antes posible
    - Utilización de TDD e integración continua
    - Deployment pipeline (tubería de despliegue)
  - Ventajas:
    - Afrontar el cambio
    - Minimizar riesgos de integración

Filosofía Wabi-sabi
Aceptar la imperfección
Software no finalizado: Suficientemente bueno (Good enough)

DevOps

Unir ***dev****elopment* y ***op****eration****s***

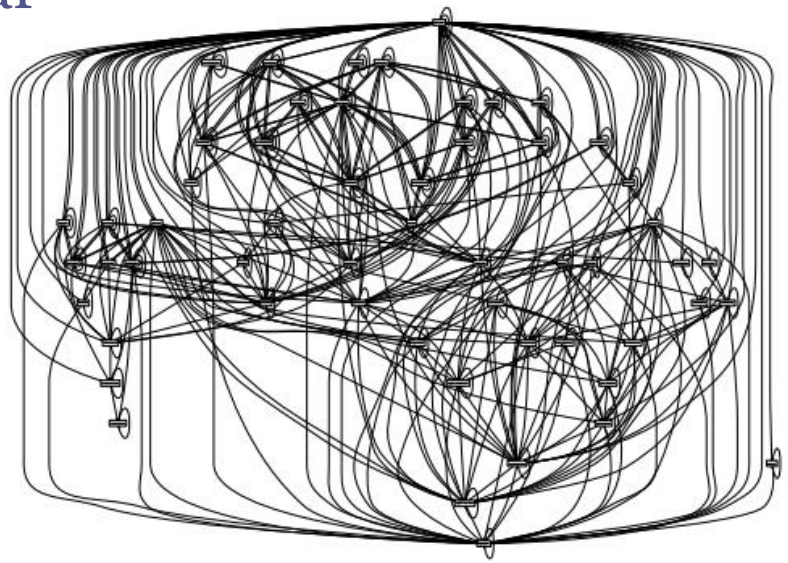
Cambio cultural en el que el mismo equipo afronta las fases:

- Codificar (code): Desarrollo y revisión de código, Integración continua
- Construir (build): Control de versiones, construcción
- Probar (test)
- Empaquetar: Gestión de artefactos
- Release: automatización de versiones
- Configurar y gestionar
- Monitorizar: Rendimiento, experiencia del usuario

Gestión de dependencias

Librería: Colección de funcionalidades utilizadas por el sistema que se desarrolla

- El sistema depende de dicha librería
  - La librería puede depender de otras librerías
  - La librería puede evolucionar
    - Versiones incompatibles
- Grafo de dependencias



Grafo de dependencias de Mozilla Firefox
Fuente: The purely functional deployment model. E. Dolstra (PhdThesis, 2006)

Grafo de dependencias

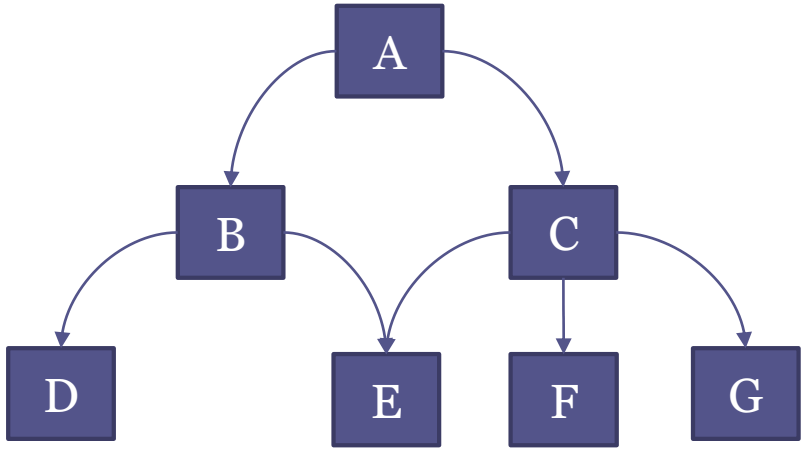
Grafo G = (V,E) donde

- V = vértices (componentes/paquetes)
- E = aristas (u,v) que indican que u depende de v

Métrica CCD (cumulative component dependency)

- Suma de dependencias de todos los componentes
  - Cada componente depende de sí mismo

En ejemplo:
CCD=7+3+4+1+1+1+1=18



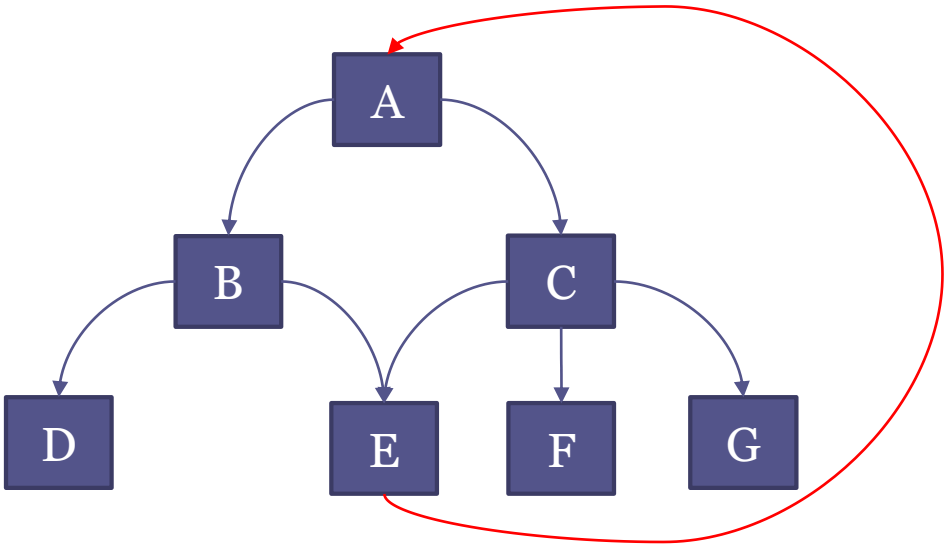
Principio de Dependencias cíclicas

El grafo de dependencias no debería tener ciclos

Añadir un ciclo puede hacer crecer la CCD

Ejemplo:

```
CCD = 7+7+7+1+7+1+1=31
```



Gestión de dependencias

- Modelos
  - Instalación local: las librerías se instalan para todo el sistema.
    - Ejemplo: Ruby Gems
  - Incluir solamente en proyecto (control de versiones)
    - Garantiza versión adecuada
  - Enlace externo
    - Repositorio con librerías
    - Dependencia de Internet y evolución de la librería

Modelos de distribución

Líneas de producto
Canales de distribución

Líneas de producto

Línea de producto: productos que comparten una serie de características comunes satisfaciendo un segmento de mercado concreto

Objetivo:

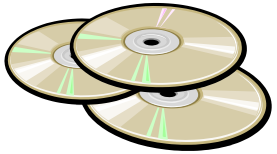
- Reducción esfuerzo desarrollo
- Mejorar productividad
- Pasar de un único producto a una línea de productos

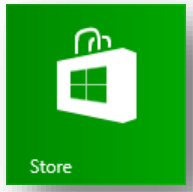
Líneas de producto

- Requisitos
  - Identificar soluciones genéricas a problemas comunes
  - Desarrollo basado en componentes
  - Plataformas genéricas
    - Reutilización de software
  - Generación automática de sistemas

Canales de distribución

- Distribución tradicional
  - CDs, DVDs, etc.
- Distribución vía Web
  - Descargas, FTP, etc.
- Mercados de aplicaciones
  - Paquetes de aplicaciones Linux
  - Apple AppStore,
  - Google Play,
  - Windows Store





Entornos de ejecución

- Software on-premises
  - Se ejecuta en el inmueble de la persona/organización que lo va a utilizar
- SaaS (Software as a Service)
  - Se ejecuta remotamente
- Contenedores (Container as a Service)
  - Ejecución local
  - Se distribuye en contenedores
    - Fácil instalación
    - Mayor rendimiento que máquinas virtuales
  - Ejemplos: Docker, Kubernetes