Tema 4. Otros aspectos de la Ingeniería del Software.

4.1. Planificación y gestión de proyectos.

4.2. Validación y verificación de software.

4.3 Mantenimiento de software.

Tema 4.1: Planificación y Gestión de Proyectos

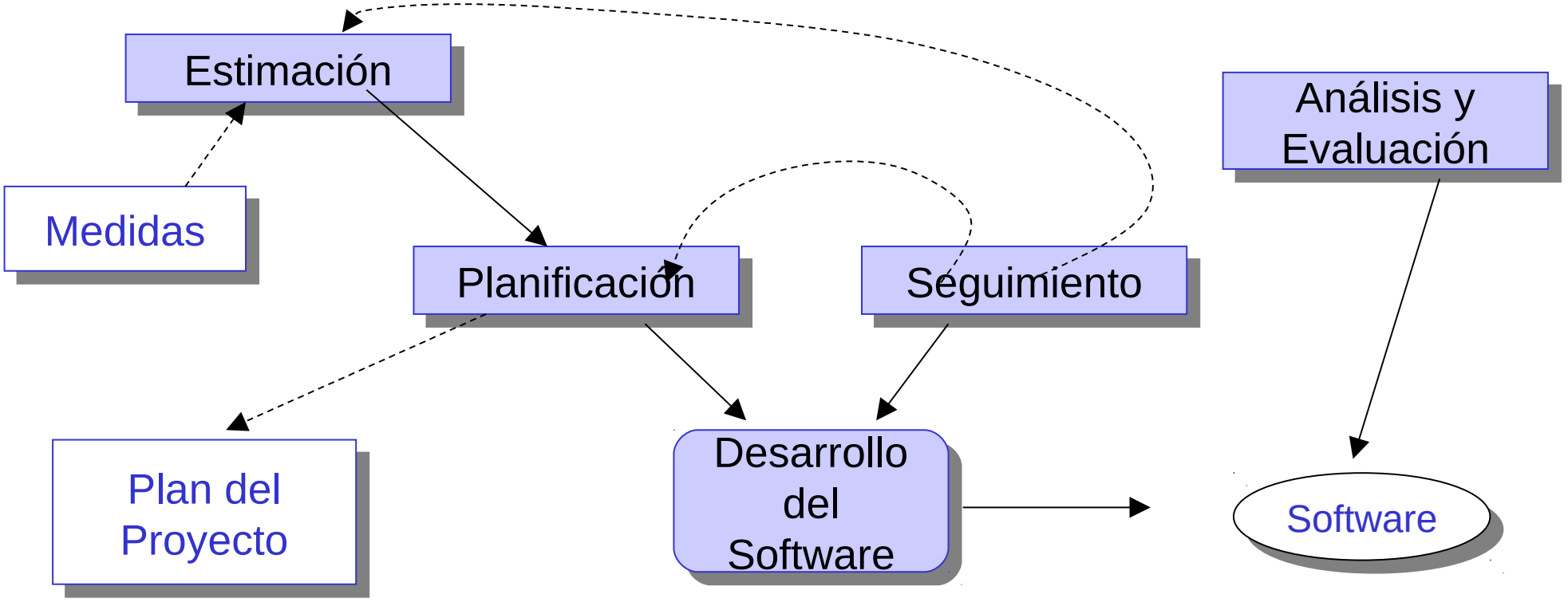
Contenido

Tema 4.1: Planificación y Gestión de Proyectos

- Introducción
- Medidas del Software: Métricas
- Estimación
- Planificación del proyecto

Bibliografía: **[PRES13 capítulos 24, 25,2 6 y 27]**
[SOMM11 capítulos 22 y 23]

Gestión de proyectos: Actividades y problemas



- La planificación de proyectos suele ser escasa y las estimaciones erróneas
- Gestores no adecuados
- Dificultades para diseñar un sistema de control/seguimiento del proyecto (vigilar el avance real del proyecto)
- No utilizamos técnicas para medir calidad y productividad del proyectos software

Planificación del proyecto

Para un **correcta gestión** de un proyecto hay que partir de una **buena planificación** del mismo, para ello hay que:

(1) Establecer el **ámbito del software**

(2) Definir las **actividades** a realizar

(3) **Estimar** los costos (tiempo, personas)

(4) Estimar **riesgos** posibles

(5) Asignar recursos a actividades

(6) Planificar temporalmente el trabajo

Resultado ---> Plan del proyecto

(Documento básico para la gestión del proyecto)

La **falta de planificación** provoca: Retrasos en la entrega, incrementos en los costes, disminución de la calidad final del producto y elevado gasto en el mantenimiento

Medidas del software: Métricas

Se usarán medidas para:

- Valorar la **calidad** del producto y del proceso de desarrollo
- Evaluar la **productividad** del equipo de desarrollo
- Facilitar las **estimaciones** en nuevos proyectos
- Ver el **estado actual** de un proyecto
- Estudiar el impacto de aplicación **de nuevas técnicas** o herramientas

Para realizar una medida hay que decidir: Qué medir, cómo medirlo, cuándo medirlo y cómo comparar las medidas

¿Qué podemos medir?

Producto: Líneas de código, velocidad de ejecución, memoria, errores, calidad, fiabilidad, complejidad, facilidad de mantenimiento...

Proceso: Coste, tiempo de desarrollo, esfuerzo, páginas de documentación, calidad, productividad, eficiencia...

Medidas del software: Métricas

Métrica: método usado para realizar medidas, estas pueden ser:

- **Orientadas al Tamaño:** Medidas directas del tamaño del software y de cada uno de sus componentes.
- **Orientadas a la Función:** Medidas indirectas basada en la funcionalidad del software
- **Orientadas a la persona:** Medida de la efectividad de las personas en el proceso de desarrollo (productividad)

Ejemplo de medidas usando métricas orientadas al tamaño, como puede ser las líneas de código (LDC oMLDC)

Esfuerzo = Personas / Mes

Productividad = MLDC / esfuerzo

Calidad = Nº Errores / MLDC

Coste medio = Euros / MLDC

Documentación = Nº Páginas de documentación /MLDC

Estimación

¿Qué necesitamos estimar?

- **Esfuerzo** requerido para desarrollar el software
- **Tamaño y complejidad** del software a desarrollar
- **Duración del proyecto** y de cada una de sus actividades
- **Coste asociado** al desarrollo

¿Cuándo la hacemos?

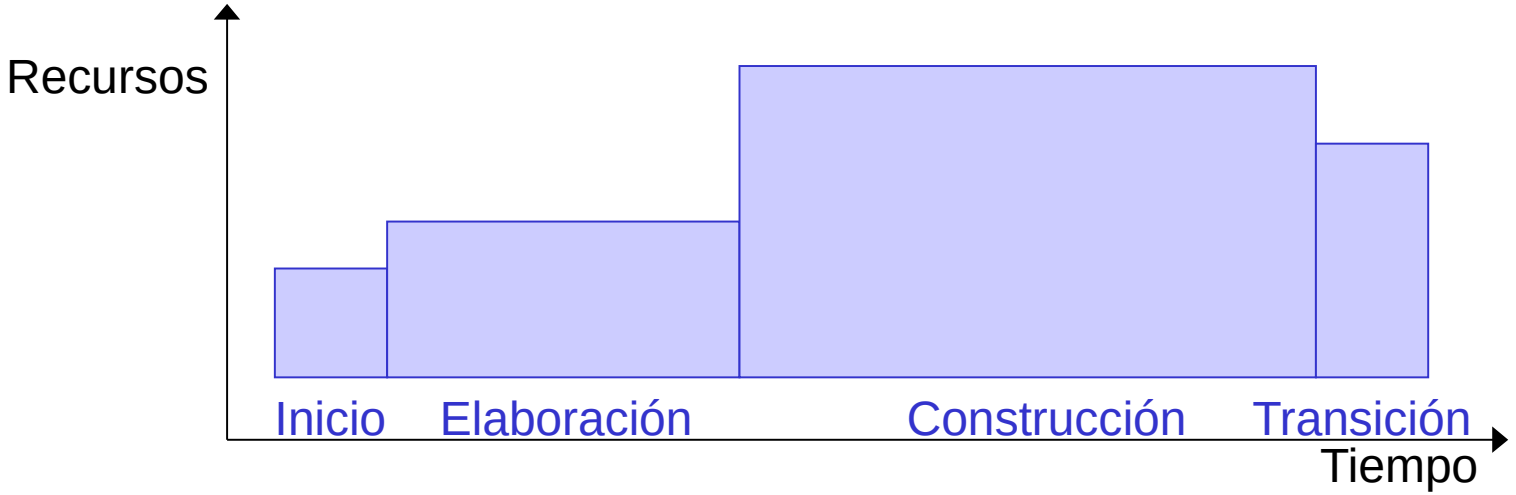
1. Durante la fase de planificación del proyecto
2. Al finalizar el análisis y la especificación de requisitos
3. Al finalizar el diseño inicial del software (diseño arquitectónico)

Una buena estimación nos va a asegurar el éxito del proyecto

La **estimación nunca será una ciencia exacta**, va a depender de factores como la complejidad y tamaño del proyecto, capacidad y composición del equipo de desarrollo...

Planificación del proyecto: **El Plan del proyecto**

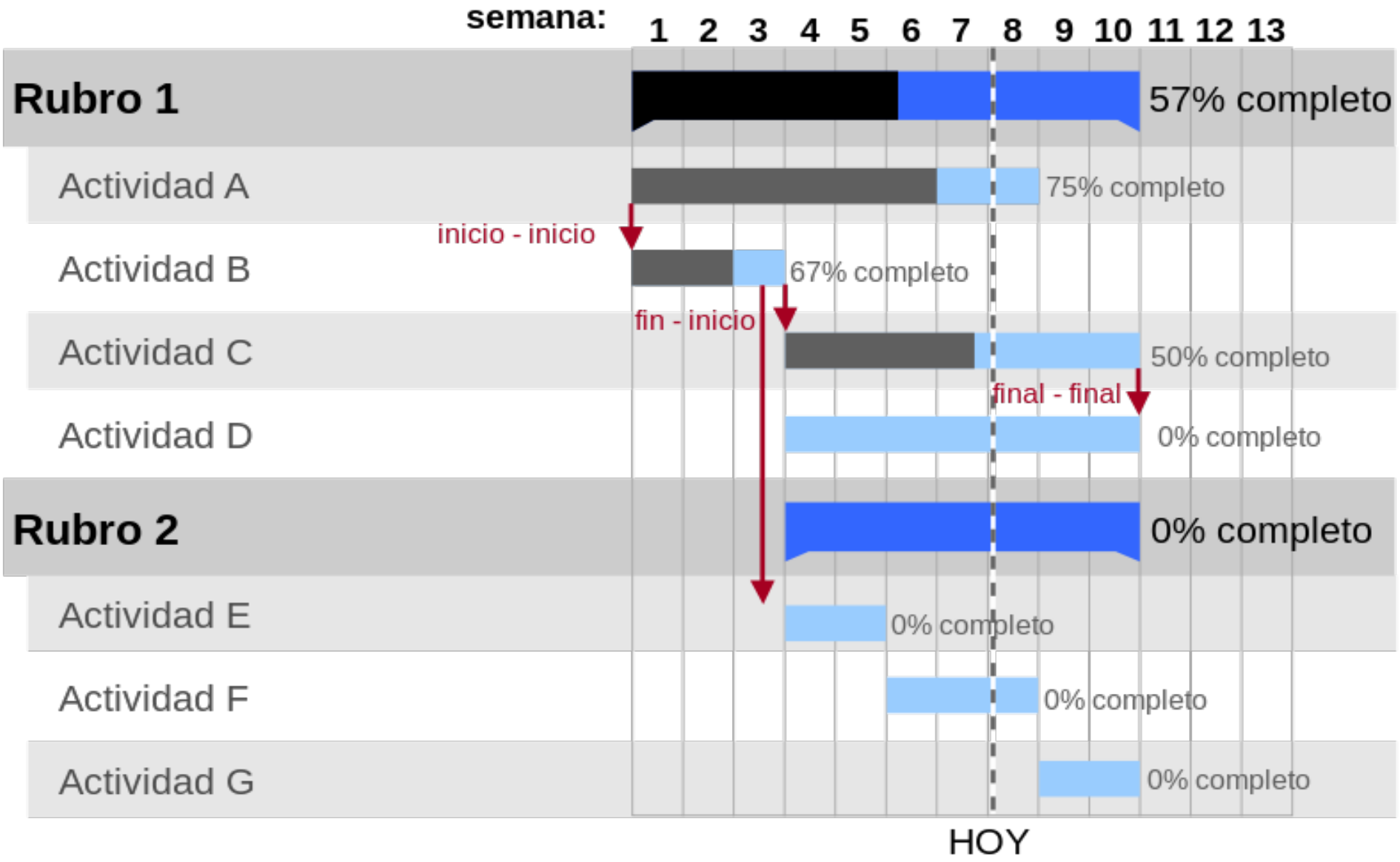
Podemos tener representaciones del siguiente tipo: Distribución de tiempo, esfuerzo y recursos en cada una de las fases del proceso de desarrollo unificado



Distribución esfuerzo(p.m.)	5%	20%	65%	10%
Distribución de tiempo	10%	30%	50%	10%

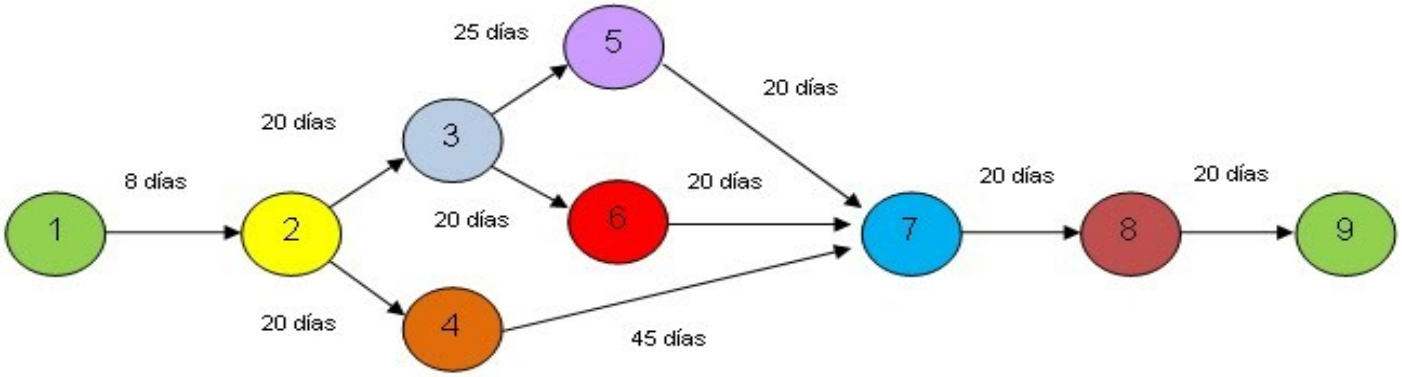
Planificación del proyecto: **El Plan del proyecto**

Diagramas de Gantt

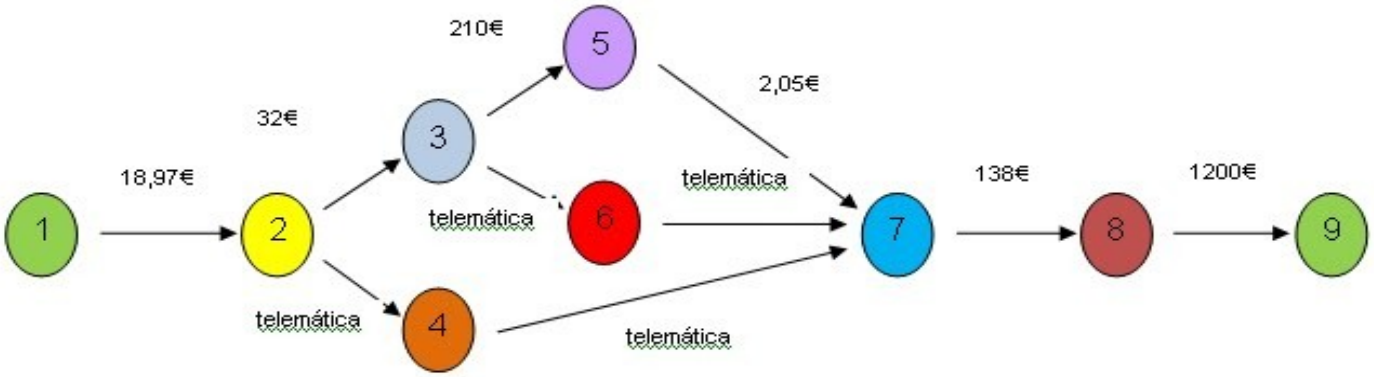


Grafos PERT

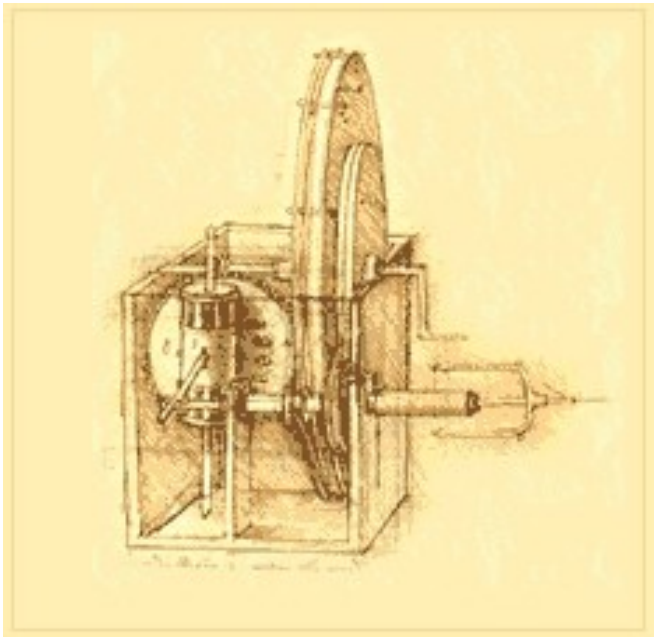
GRAFO PERT: Tiempo



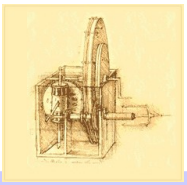
GRAFO PERT: Coste



Tema 4.2 - Validación y verificación de software (V&V)



Bibliografía: **[PRES13 capítulo 16, 17 y 18]**
[SOMM05 (7ª edición) capítulos 22 y 23]



Contenido

Tema 4.2 - Validación y verificación de software (V&V)

- Concepto de V&V
- Planteamiento
- Tareas de la V&V
- Inspecciones del software
- Prueba del software
  - Conceptos y niveles
  - Pruebas de Aceptación/Validación
  - Prueba de defectos
    - Prueba del sistema
    - Prueba de Integración
    - Prueba de Unidad
    - Técnicas de prueba de Unidad

Concepto de V&V

Conjunto de procesos de comprobación y análisis que aseguran que el software que se desarrolla está acorde a su especificación y cumple las necesidades de los clientes

Verificación → ¿Estamos construyendo el producto correctamente?
Coherencia interna y conforme con su especificación.

Validación → *¿Estamos construyendo el producto correcto?*
Producto que cumple con las expectativas del cliente/usuario.

Planteamiento

- Se trata de **encontrar los defectos** de los productos software, no de corroborar que, *como cabía esperar,* todo está bien.
- Debemos asumir que nuestro programas tienen errores, y buscarlos, a sabiendas que **no los vamos a encontrar todos**, la perspectiva no es muy atractiva.
- Es necesario acometer con una **mentalidad positiva** esta **tarea destructiva**. Una actividad de verificación y validación alcanza el éxito cuando permite encontrar errores.
- Es necesario distanciarse conceptualmente respecto a la perspectiva desde la que se abordó el desarrollo, por ello es conveniente que la realice un **equipo distinto al de desarrollo**.
- Consume entre el 30% y 40% del esfuerzo de desarrollo.

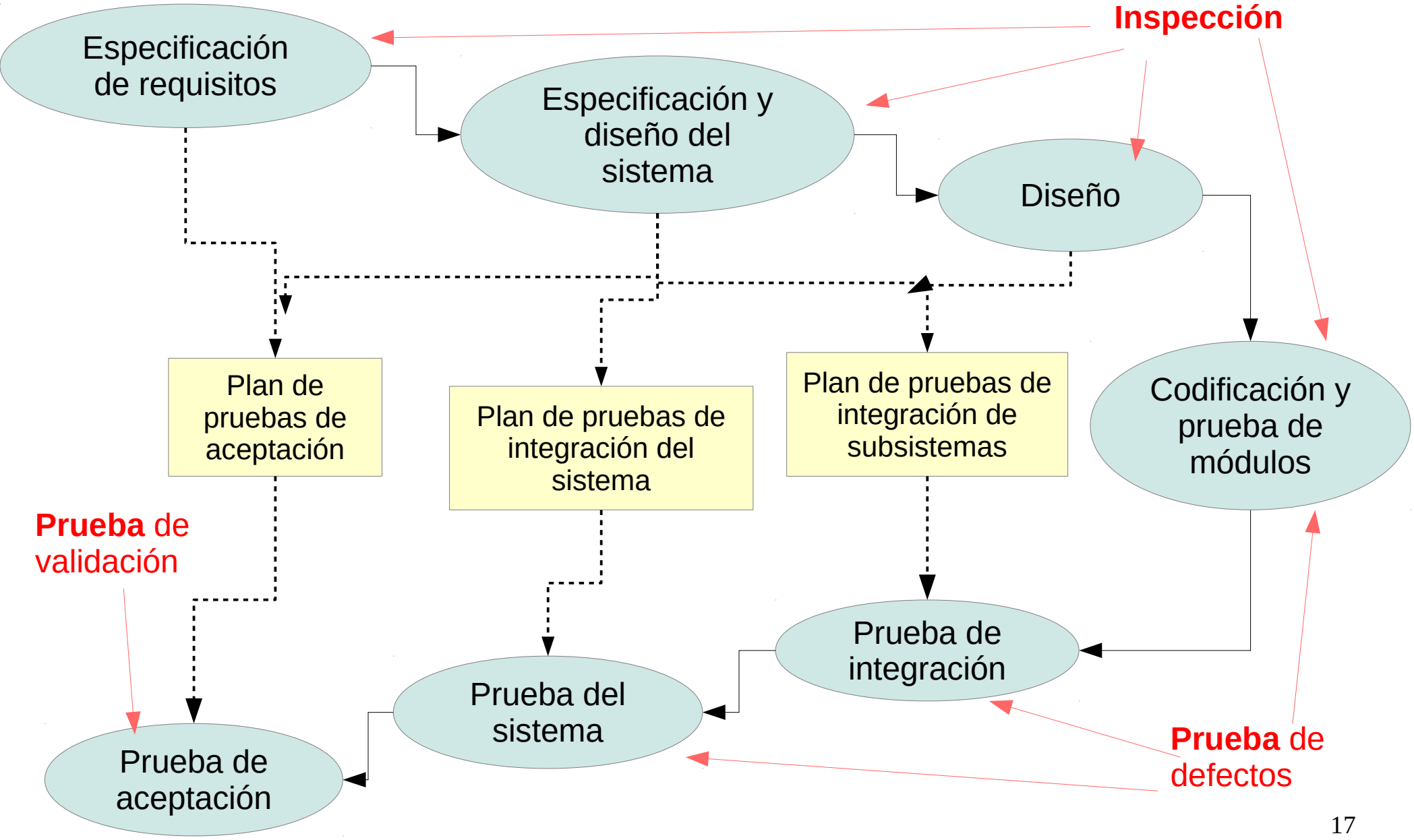
Tareas de V&V

Son complementarias y tienen lugar en cada una de las etapas del proceso de producción del software. Éstas en general son:

- **Inspección del software**: Analiza y comprueba las representaciones del sistema: documentos, diagramas, código... Son técnicas estáticas, no necesitan de la ejecución del sistema.
- **Prueba del software**: Ejecución de la implementación con datos de prueba, para comprobar que funciona cómo se esperaba que lo hiciese, son técnicas dinámicas. Pueden ser:
- **Pruebas de validación**: intentan demostrar que el software es el que el cliente quiere.
- **Pruebas de defectos:** hallan inconsistencias entre un sistema software y su especificación.

Nota: *No confundir la V&V con la depuración de errores*

Tareas de la V&V



Prueba de Aceptación/Validación

Proceso de la prueba de aceptación

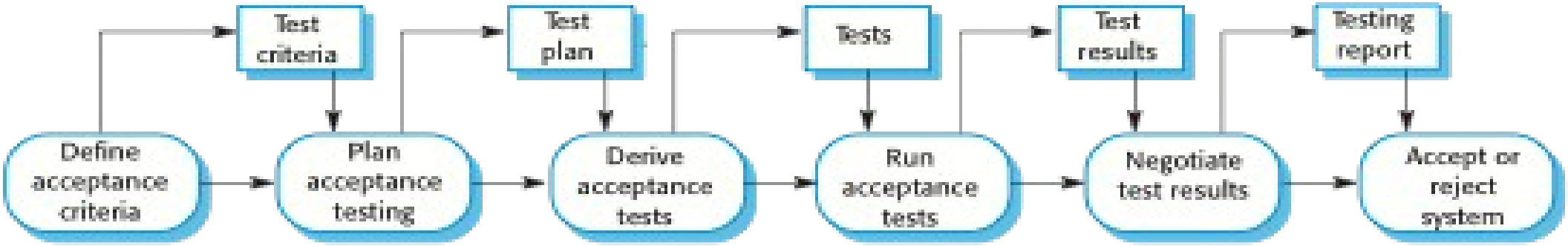


Figura de SOMM05 (7ª edicion): http://www.softwareengineering-9.com/

- Basadas en los criterios de aceptación
- Resultado de la prueba: Aceptación o rechazo del sistema
- Pueden ser:
  - Pruebas alfa (entorno de desarrollo)
  - Pruebas beta (entorno del cliente)

Prueba de defectos: Proceso

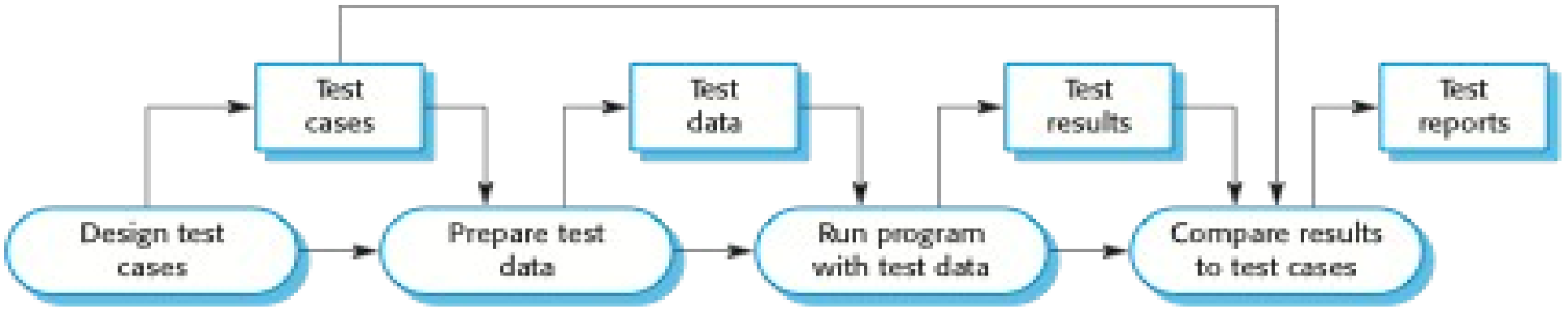


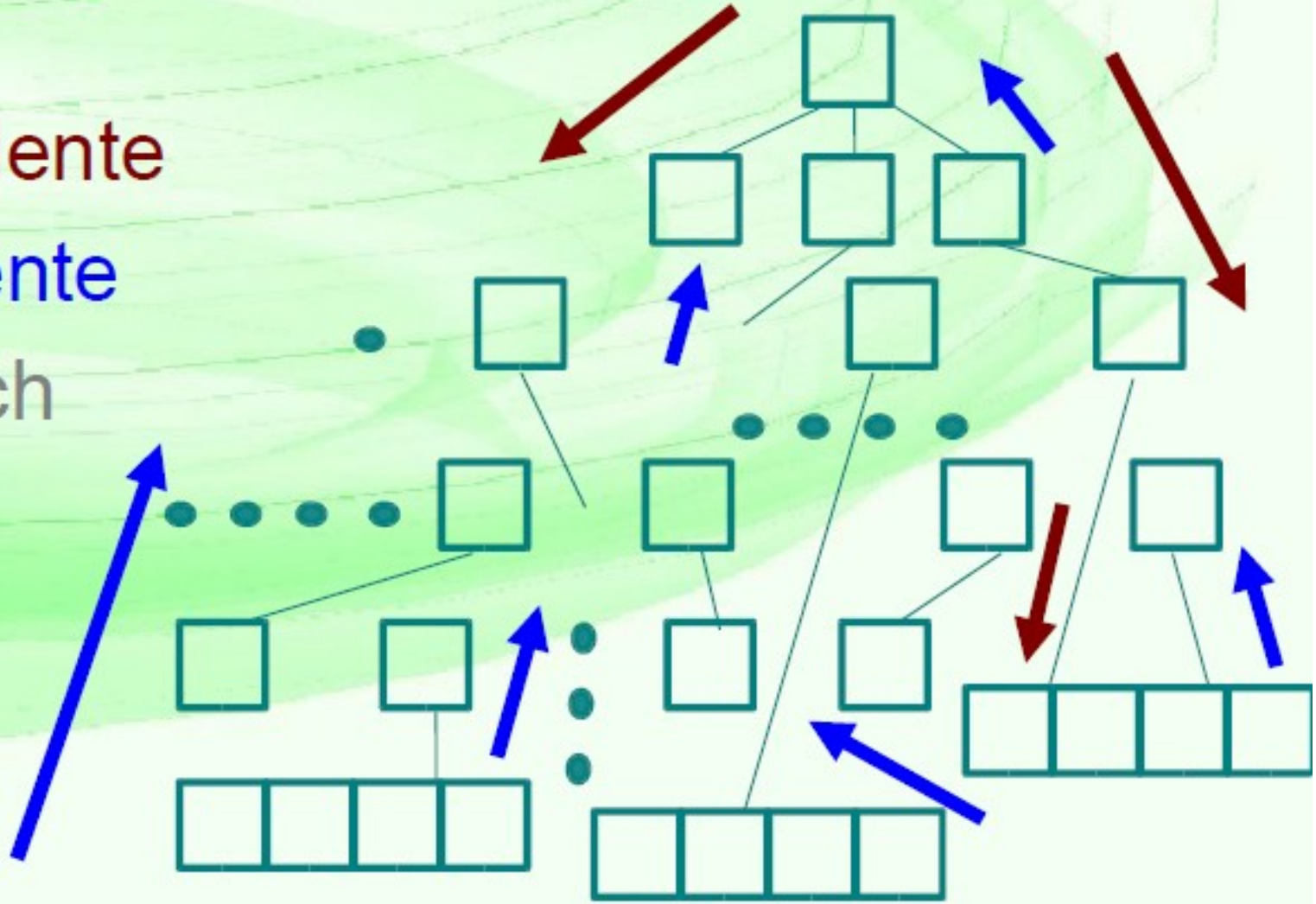
Figura de SOMM11 (7ª edicion): http://www.softwareengineering-9.com/

- Los **casos de prueba** son especificaciones de las entradas a la prueba y de la salida esperada del sistema, más una declaración de lo que se prueba.
- Los **datos de prueba** son las entradas seleccionadas, que cumplen con lo especificado en los casos de prueba.

Prueba de defectos: **Prueba de integración**

Estrategias:

- descendente
- ascendente
- sandwitch



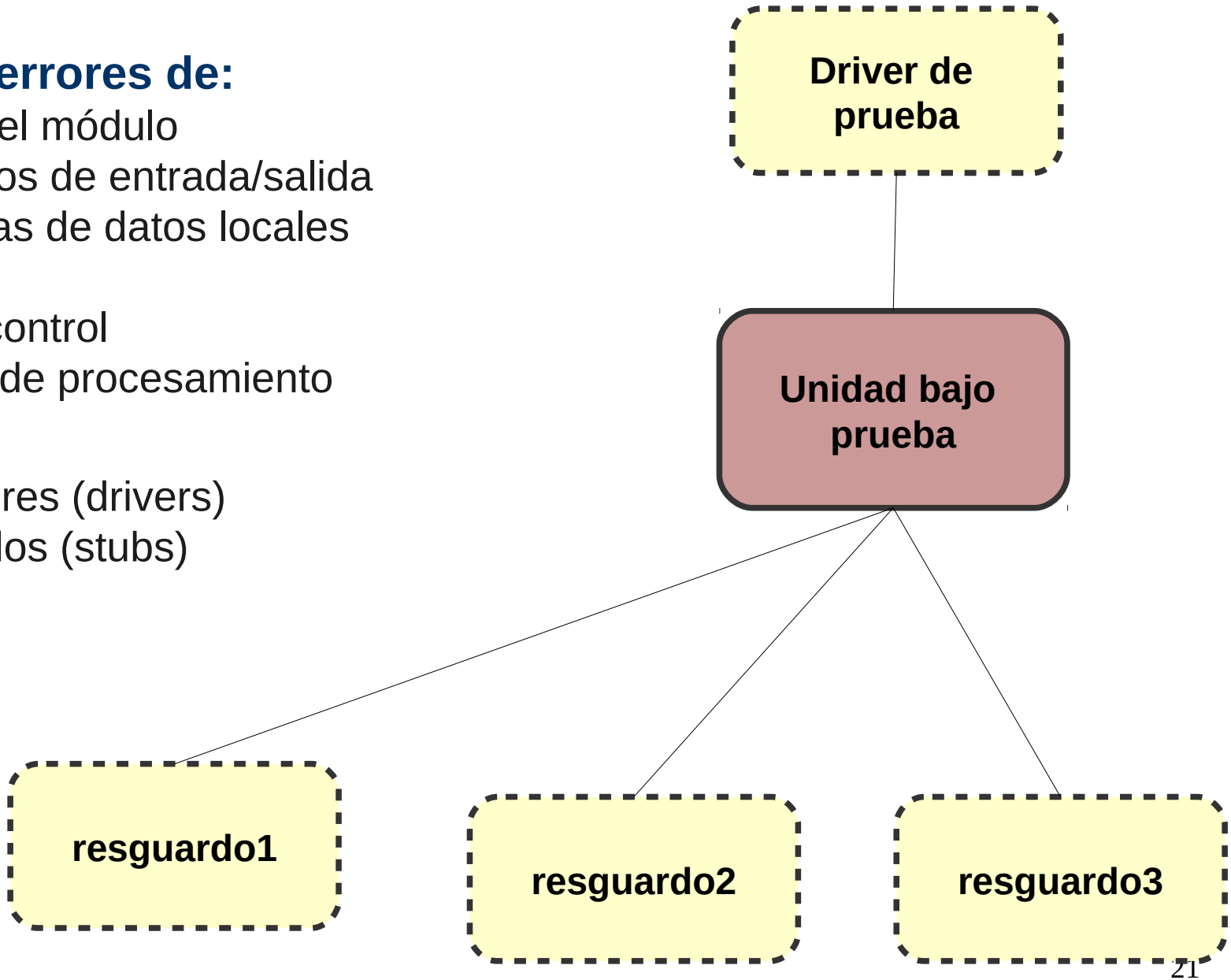
Prueba de defectos: **Prueba de unidad**

Se buscan errores de:

- Interfaz del módulo
- Parámetros de entrada/salida
- Estructuras de datos locales
- Cálculos
- Flujo de control
- Caminos de procesamiento

Requieren:

- Conductores (drivers)
- Resguardos (stubs)



Prueba de defectos: **Técnicas de Prueba de Unidad**

Las **técnicas de prueba** ayudan a definir conjuntos de casos de prueba, aplicando cierto criterio

Los Caso de prueba especifican la prueba en términos de:

- Los valores de entrada a suministrar
- Los valores de salida correctos

Tipos de técnicas de prueba:

- **Caja negra**: los casos se deducen de las interfaces y especificaciones del módulo
- **Caja blanca**: los casos se deducen del contenido o interior del módulo

Técnicas de caja negra: Particiones Equivalentes

Objetivo:

Ejecutar los módulos con todos los posibles valores distintos de los argumentos de entrada al módulo

Pasos a seguir:

1. Encontrar todas particiones equivalentes (clases de equivalencia) de todos los valores de entrada, es decir la especificación de los **casos de prueba**
2. Derivar los **datos de prueba** eligiendo al menos un valor de cada clase de equivalencia

Técnicas de caja negra: Particiones Equivalentes

Ejemplo

Si tenemos un módulo con los siguientes argumentos:

NombreArticulo: String entre 2 y 15 caracteres alfanuméricos

Peso [5]: Array de 5 elementos reales, donde cada uno de los elementos representa el peso, entre 0 y 10.000 gr., que se puede tener del artículo en cuestión. Estos pesos están ordenados de menor a mayor y si el artículo tiene sólo tres pesos los dos primeros elementos estarán a 0 y los tres últimos a valores distinto de cero

Técnicas de caja negra: Particiones Equivalentes

Ejemplo: Clases de equivalencia o casos de prueba

NombreArticulo:

1. Alfanumérico (válido): AcdEf4
2. No alfanumérico (no válido): A$%!1

Longitud NombreArticulo:

3. 2<=L<=15 (válido): afdHteKJN14
4. L<2 (no válido): a
5. L>15 (no válido):aaaaaaaaaaaaaaaaaaaaaaaaa

Rango de valores para el peso:

6. Peso < 0 gr. (invalida): -2
7. 0<= Peso <=10.000 gr. (válida): 500
8. Peso > 10.000 (invalida): 11.000

Orden:

9. Elementos ordenados (válida): [0,0,1,5,10]
10. Elementos no ordenados (inválida): [1,0,10,5,0]

Técnicas de caja negra: Particiones Equivalentes

Ejemplo: Datos de Prueba

Caso 1: (“abcd”,[0,1,2,3,4])

Resultado esperado: Ejecución del módulo sin problemas.

Caso 2: (“abcd”,[0,1000,2000,3000,11000])

Resultado esperado: Salida por error (“peso no válido”)

Caso 3: (“abcd”, [-1,0,2,3,4])

Resultado esperado: Salida por error (“peso no válido”)

Caso 4: (“abcd”, [1,0,10,5,0)])

Resultado esperado: Salida por error (“valores desordenados”)

Caso 5: (“$%&”,[0,0,0,0,1])

Resultado esperado: Salida por error (“nombre no válido”)

Caso 6: (“a”,[0,0,0,0,1])

Resultado esperado: Salida por error (“long. nombre no valida”)

Caso 7: (“aaaaaaaaaaaaaaaa”,[0,0,0,0,1])

Resultado esperado: Salida por error (“long. nombre no valida”)

Tema 4.3 : Mantenimiento del Software

Contenido

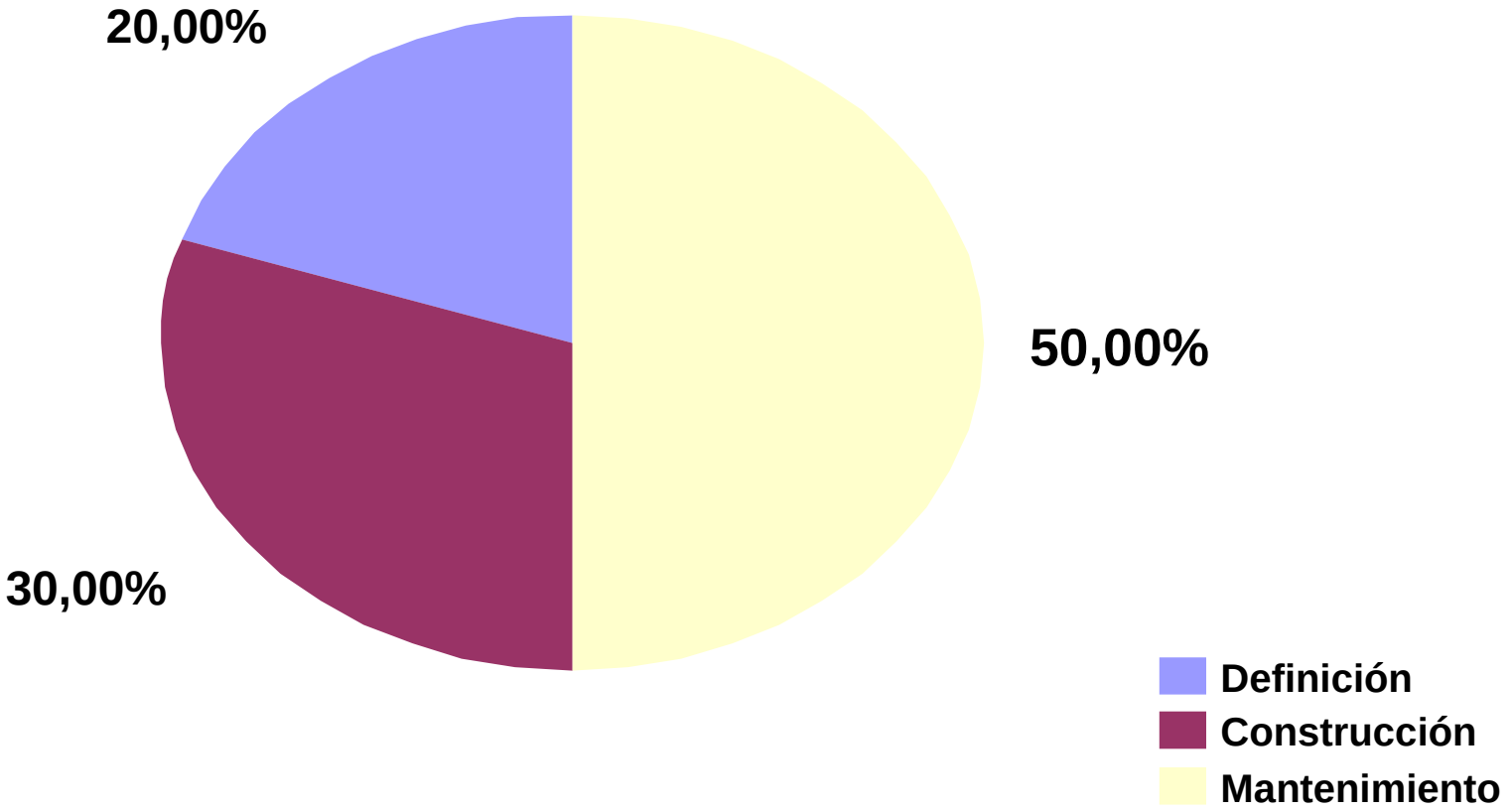
Tema 4.3 : Mantenimiento del Software

- Introducción.
- Conceptos fundamentales.
- Proceso de mantenimiento.
- Tipos de mantenimiento.
- Técnicas de mantenimiento.

Bibliografía: **[PRES13 capítulo 29]**
[SOMM11 capítulo 9]

Introducción

El mantenimiento constituye con creces la mayor parte del esfuerzo total del desarrollo, siendo una actividad que consume muchos recursos.(IEEE 1219)



Introducción

Principales causas del mantenimiento:

- Código no estructurado.
- Insuficiente conocimiento del dominio del problema.
- Documentación insuficiente.

Proceso de mantenimiento

Actividades de mantenimiento:

- Comprensión del software y de los cambios a realizar.
  - Conocer funcionalidad, estructura interna y requisitos.
- Modificación del software.
  - Crear y modificar las estructuras de datos, la lógica de los procesos, las interfaces y la documentación.
- Realización de pruebas.
  - Realizar pruebas selectivas para comprobar la corrección del software modificado.

Tipos de mantenimiento

- **Mantenimiento correctivo:**
  - Modificaciones a un producto software después de la entrega para corregir defectos descubiertos.
- **Mantenimiento adaptativo:**
  - Modificaciones a un producto software después de la entrega para permitir que ese producto se pueda seguir utilizando en un entorno diferente.
- **Mantenimiento perfectivo:**
  - Modificaciones de un producto software después de su entrega para mejorar el rendimiento o la facilidad de mantenimiento.

Tipos de mantenimiento

- **Mantenimiento preventivo:**
  - Orientado a prevenir errores que pueden surgir en el futuro.
- **Mantenimiento de emergencia:**
  - Engloba los cambios que deben realizarse sin planificación previa, pues resultan esenciales para mantener un sistema en operación.

Tipos de mantenimiento

Técnicas de mantenimiento

- **Ingeniería inversa:**
  - Consiste en analizar un sistema para identificar sus componentes y las relaciones entre ellos, así como en elaborar nuevas representaciones del sistema, generalmente de más alto nivel.
- **Reingeniería:**
  - Consiste en la modificación de componentes software mediante el uso de técnicas de ingeniería inversa para su análisis y herramientas de ingeniería directa para su reconstrucción.
- **Reestructuración:**
  - Consiste en un cambio de representación de un producto software, dentro del mismo nivel de abstracción.

Técnicas de mantenimiento