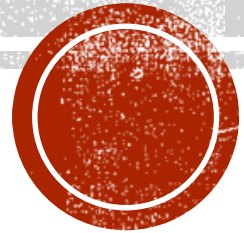
ALGORÍTMICA Y PROGRAMACIÓN II

Introducción



OBJETIVOS

- Profundizar y mejorar la capacidad de programar en el **paradigma imperativo**.
- Conocer el concepto de **TDA y sus aplicaciones** en la resolución de problemas.
- Reconocer y resolver problemas de naturaleza **recursiva**.

SE ESPERA DEL ALUMNO

- Profundizando conceptos de la **Ingeniería de Software**:
  - Análisis de Algoritmos
  - Modularización
  - Técnicas de Diseño Descendente
  - Encapsulamiento
  - Verificación
  - Otros….
- Reforzar los conceptos del **QUÉ y del CÓMO** en el desarrollo de un software.

SE ESPERA DEL ALUMNO

- Seleccionar los **TDA** y decidir su representación (estática o dinámica), pudiendo evaluar la conveniencia de una u otra implementación.
- Implementar **soluciones recursivas**, analizando ventajas y desventajas de su aplicación.
- **Desarrollar e implementar un sistema de software**, de mediana complejidad, a partir de las especificaciones dadas por la cátedra.
- **Documentar y testear** el sistema.

ES EL SEGUNDO PASO EN EL PROCESO DE APRENDIZAJE DE PROGRAMACIÓN

- Se producen dos saltos conceptuales:
  - De **abstracción**, con el manejo de los TDA y sus implementaciones (estáticas y dinámicas) y el concepto de recursión.
  - De **escala**, con la implementación de un sistema sencillo de software.

¿QUÉ ES EL SOFTWARE?

- **Es la construcción de un producto mediante un proceso complejo,** utilizando técnicas y herramientas específicas que requiere una gran capacidad intelectual.
- El software son todos los aspectos que toma este, **los entregables**.
  - Ingeniería de Requisitos
  - Análisis y Diseño
  - Programa
  - Pruebas
  - Manuales – Capacitación
- El software es conocimiento acumulado / empaquetado / ejecutable.

PROGRAMAS + PROCEDIMIENTOS + DOCUMENTACIÓN + DATOS DE LA OPERACIÓN DEL SISTEMA

¿POR QUÉ EL SOFTWARE ES ÚNICO?

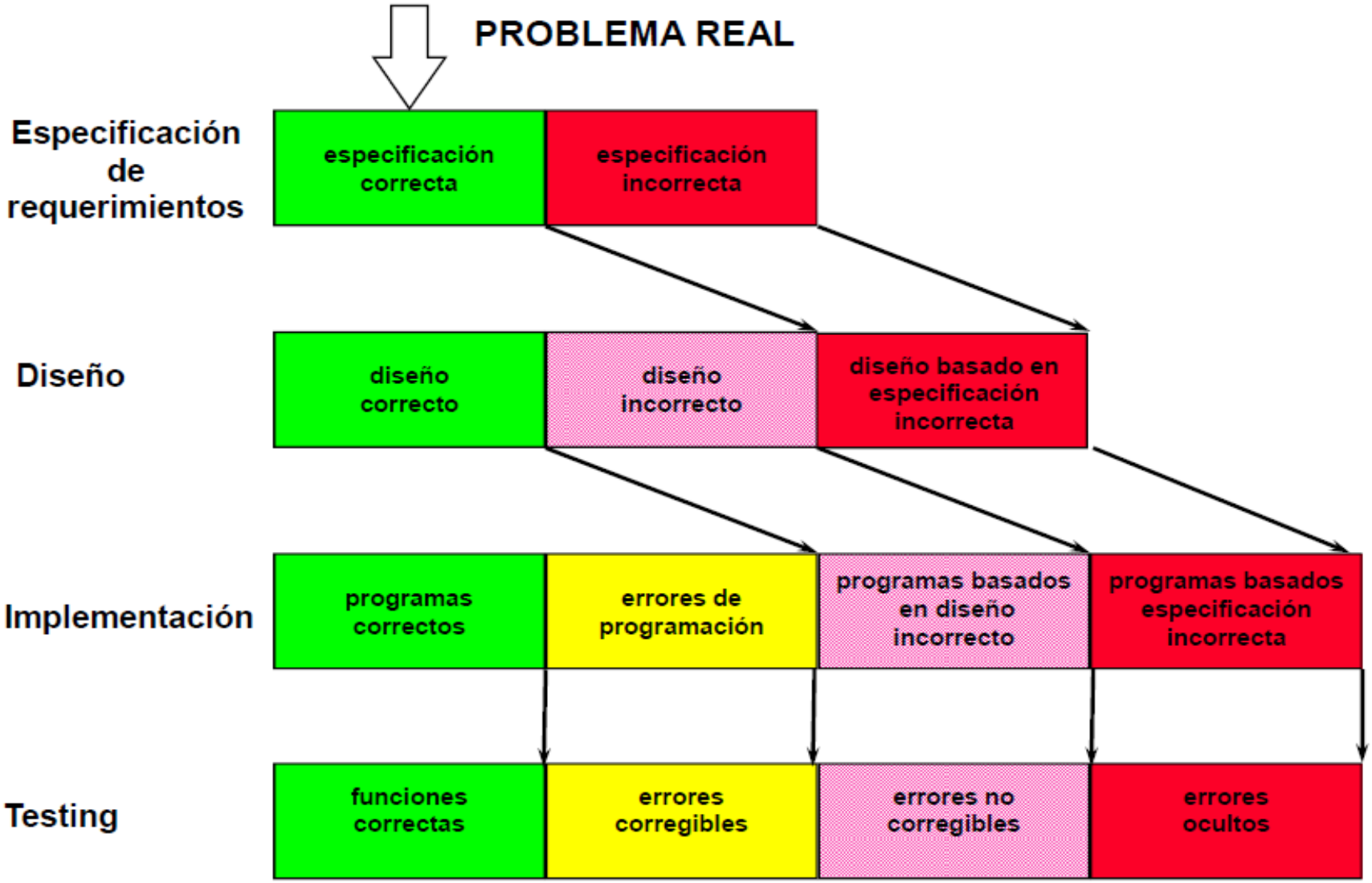
- Es intangible
- Posee un alto contenido intelectual
- Potencialmente modificable hasta el infinito
- Su proceso de desarrollo es de mano de obra intensivo, basado en equipos y por proyectos
- ….

CARACTERÍSTICAS DEL SOFTWARE

- **Corrección funcional**: se comporta de acuerdo a las especificación de requerimientos funcionales.
- **Confiabilidad**: el usuario puede depender del software
- **Robustez**: se comporta "razonablemente", incluso en circunstancias no previstas
- **Performance**: uso económico de los recursos de computación (eficiencia)
- **Amistosidad**: fácil uso por los seres humanos
- **Verificabilidad**: sus propiedades pueden verificarse fácilmente
- **Mantenibilidad**: puede repararse y evolucionar
- **Reusabilidad**: utilizar componentes en otros sistemas
- **Portabilidad**: puede correr en distintos ambientes
- **Comprensibilidad**: facilidad de ser entendidos por los usuarios (desarrolladores)
- **Interoperatividad**: capacidad de coexistir y cooperar con otros sistemas
- **Oportunidad**: capacidad de liberar un producto en tiempo

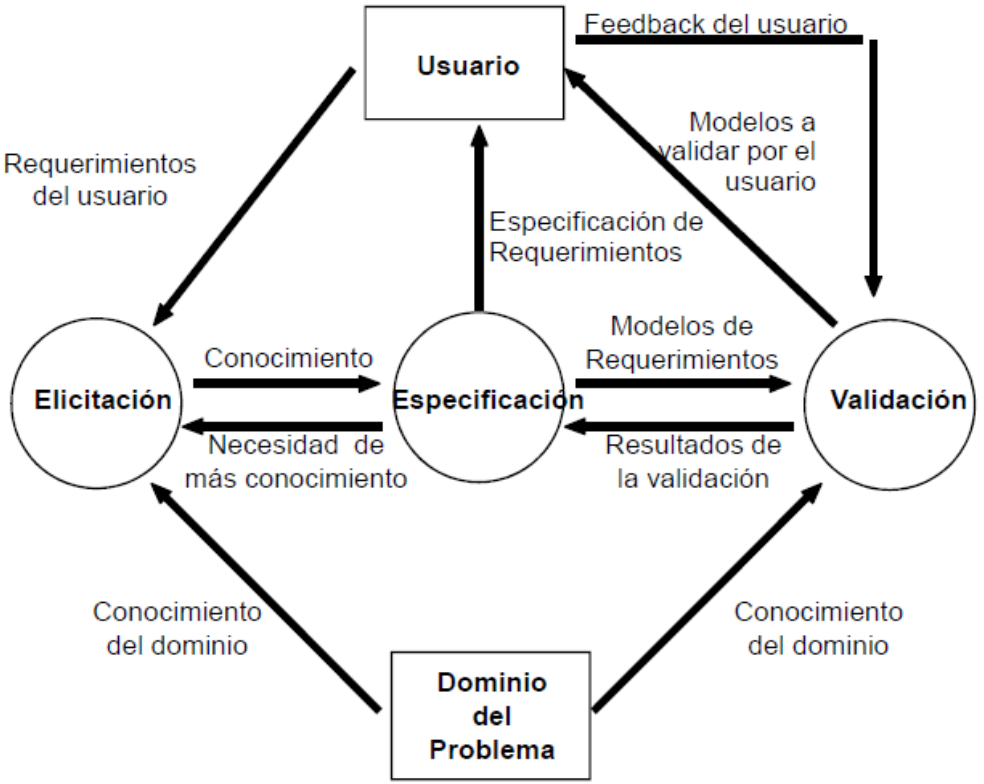
INGENIERÍA DE SOFTWARE

- Fairley
  - La Ingeniería Software es la disciplina tecnológica y de administración que se ocupa de la producción y evolución sistemática de productos de software que son desarrollados y modificados dentro de los tiempos y costos estimados
- Ghezzi
  - Ingeniería Software es el campo de la ciencia de la computación que trata con la construcción de sistemas de software que son tan grandes o complejos que son construidos por un equipo o equipos de ingenieros
- IEEE
  - El uso de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento de software, es decir, la aplicación de la ingeniería al software
- Sommerville
  - Es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software.



Integración

INGENIERÍA DE REQUISITOS (LOUCOPOULOS)



EL CONTRATO SOCIAL DE LOS REQUERIMIENTOS – DERECHOS DEL USUARIO

- Esperar del analista que **hable su lenguaje**
- Que el profesional aprenda sobre su **negocio y sus objetivos**
- Que el profesional **escriba una especificación de requerimientos** de software
- Esperar de los desarrolladores un **trato respetuoso**
- **Obtener ideas y alternativas** para los requerimientos y su implementación
- Recibir estimaciones de buena fe de **costos de los cambios**
- Recibir un sistema que satisfaga sus **necesidades funcionales y de calidad**

VERIFICACIÓN - VALIDACIÓN

Sommerville

- Verificación
  - Busca comprobar que el sistema cumple con los requerimientos especificados (funcionales y no funcionales)
  - **¿El software está de acuerdo con su especificación?**

- Validación
  - Busca comprobar que el software hace lo que el usuario espera.
  - **¿El software cumple las expectativas del cliente?**

VERIFICACIÓN - VALIDACIÓN

Boehm

- Verificación
  - ¿Estamos construyendo el producto correctamente?
- Validación
  - ¿Estamos construyendo el producto correcto?

Ghezzi

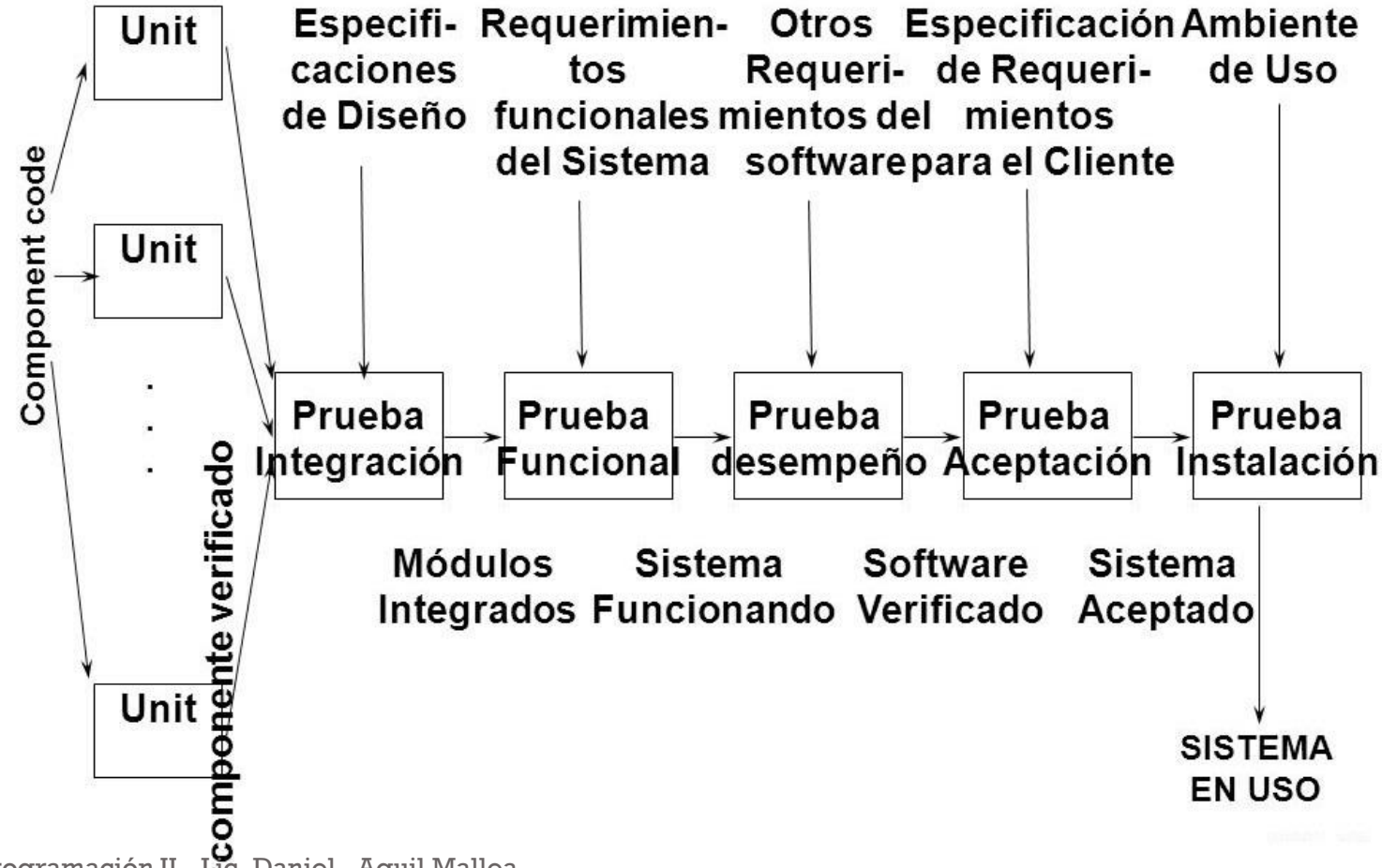
- Verificación
  - Todas las actividades que son llevadas a cabo para averiguar si el software cumple con sus objetivos

VERIFICACIÓN - VALIDACIÓN

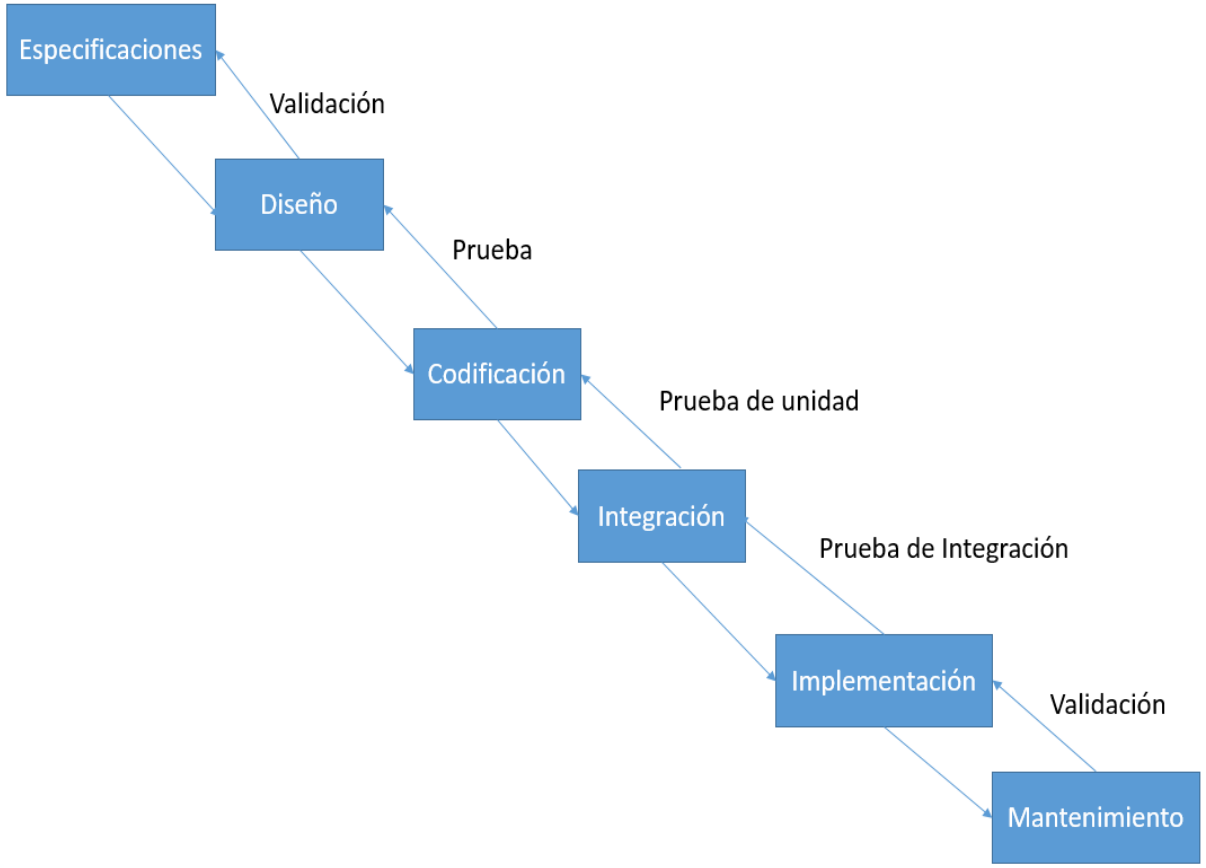
IEEE

- Verificación
  - The process of evaluating a system or component to determine whether the products of a given development phase **satisfy the conditions imposed at the start of the phase**
- Validación
  - The process of evaluating a system or component during or at the end of the development process to determine whether it **satisfies specified requirements**

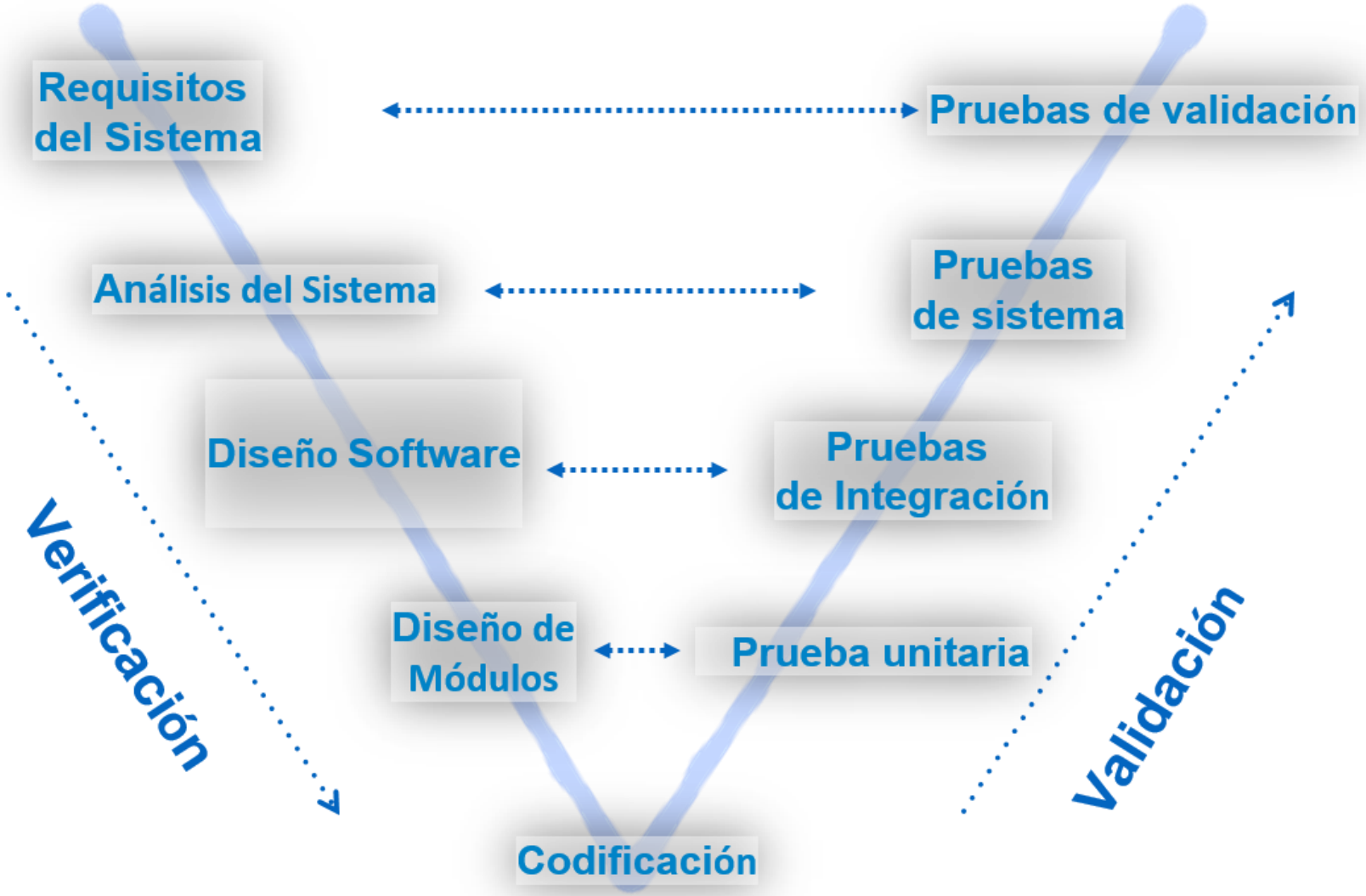
VERIFICACIÓN - VALIDACIÓN



VERIFICACIÓN - VALIDACIÓN



VERIFICACIÓN VALIDACIÓN



¿QUIÉN VERIFICA?

Pruebas Unitarias

- Las realiza el **equipo de desarrollo**. En general la misma persona que lo implementó.
- Es positivo el conocimiento detallado del módulo a probar

Pruebas de Integración

- Normalmente las realiza el **equipo de desarrollo**
- Es necesario el conocimiento de las interfaces y funciones en general

Resto de las pruebas

- En general un **equipo especializado** (verificadores)
- Es necesario conocer los requerimientos y tener una visión global

¿QUIÉN VERIFICA?

¿Por qué un equipo especializado?

- Maneja mejor las técnicas de pruebas
- Conoce los errores más comunes realizados por el equipo de programadores
- **Problemas de psicología de pruebas**
  - El autor de un programa tiende a cometer los mismos errores al probarlo
  - Debido a que es "SU" programa inconscientemente tiende a hacer casos de prueba que no hagan fallar al mismo
  - Puede llegar a comparar mal el resultado esperado con el resultado obtenido debido al deseo de que el programa pase las pruebas

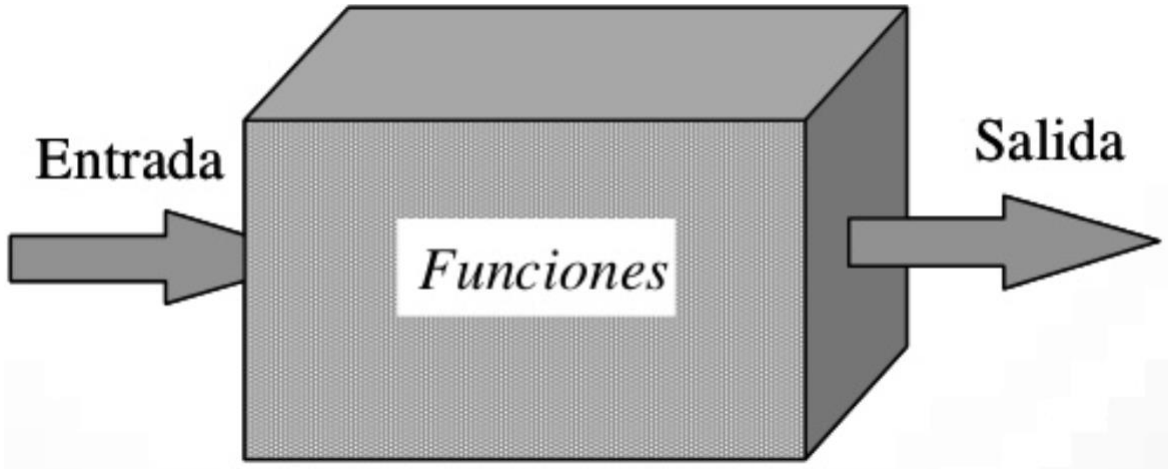
VERIFICACIÓN ESTÁTICA-DINÁMICA

- Inspecciones de software. Se ocupa del análisis de representaciones estáticas del sistema para describir problemas (verificación estática)
  - Pueden ser complementadas por documentos basados en herramientas y análisis del código
    - Análisis [automatizado] de código fuente
    - Análisis formal
- Pruebas del software. Se ocupa de la ejercitación y la observación del comportamiento del producto (verificación dinámica)
  - El sistema se ejecuta con datos de pruebas y se observa su comportamiento operativo.
    - Caja Negra
    - Caja Blanca
- Ejecución simbólica
  - Técnica híbrida

VISIÓN DE LOS OBJETOS A PROBAR

Caja Negra

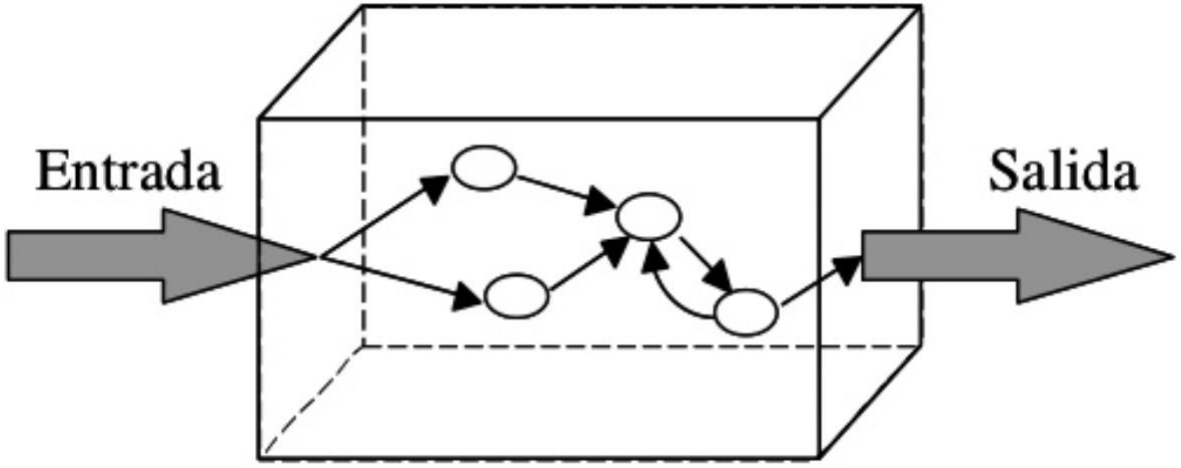
- Entrada a una caja negra de la que no se conoce el contenido y ver que salida genera
  - Casos de prueba - No se precisa disponer del código
  - Se parte de los requerimientos y/o especificación
  - Porciones enteras de código pueden quedar sin ejercitar



VISIÓN DE LOS OBJETOS A PROBAR

Caja Blanca

- A partir del código identificar los casos de prueba interesantes
  - Casos de prueba – Se necesita disponer del código
  - Tiene en cuenta las características de la implementación
  - Puede llevar a evitar la prueba de algún requerimiento no implementado

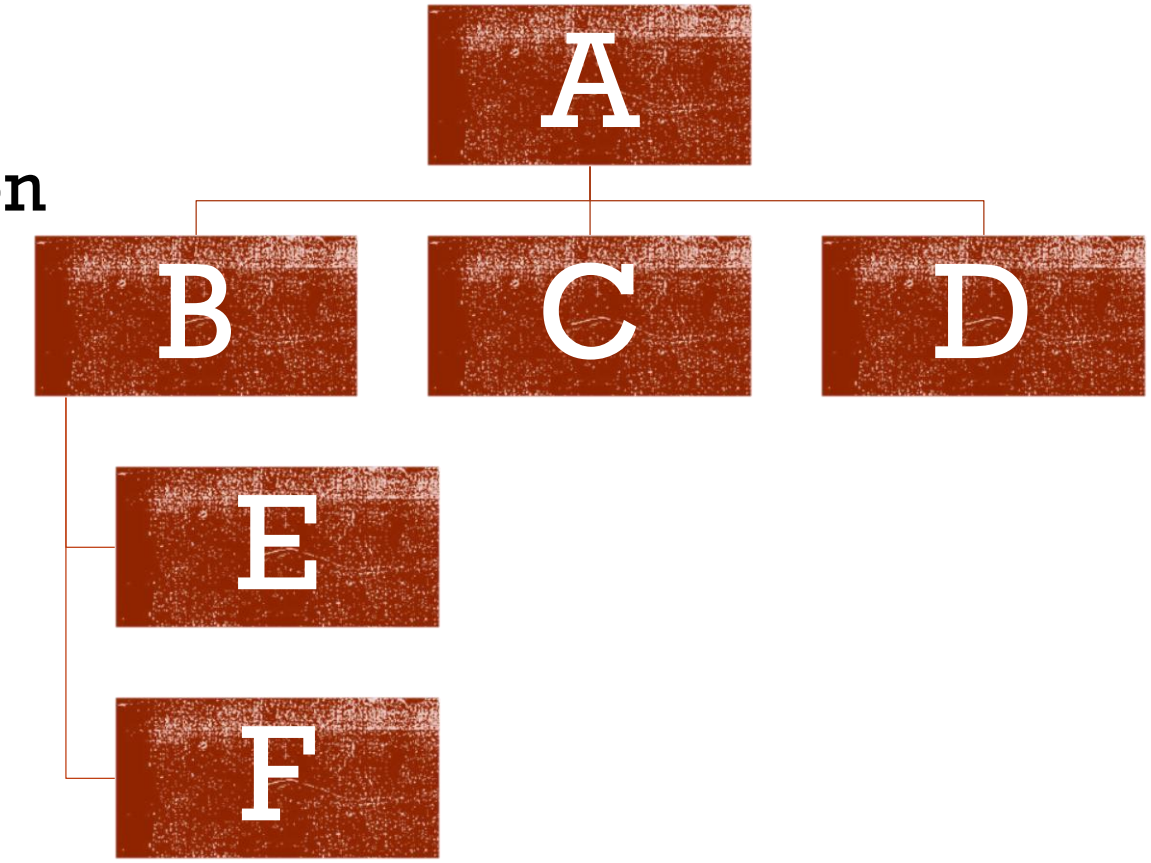


V&V

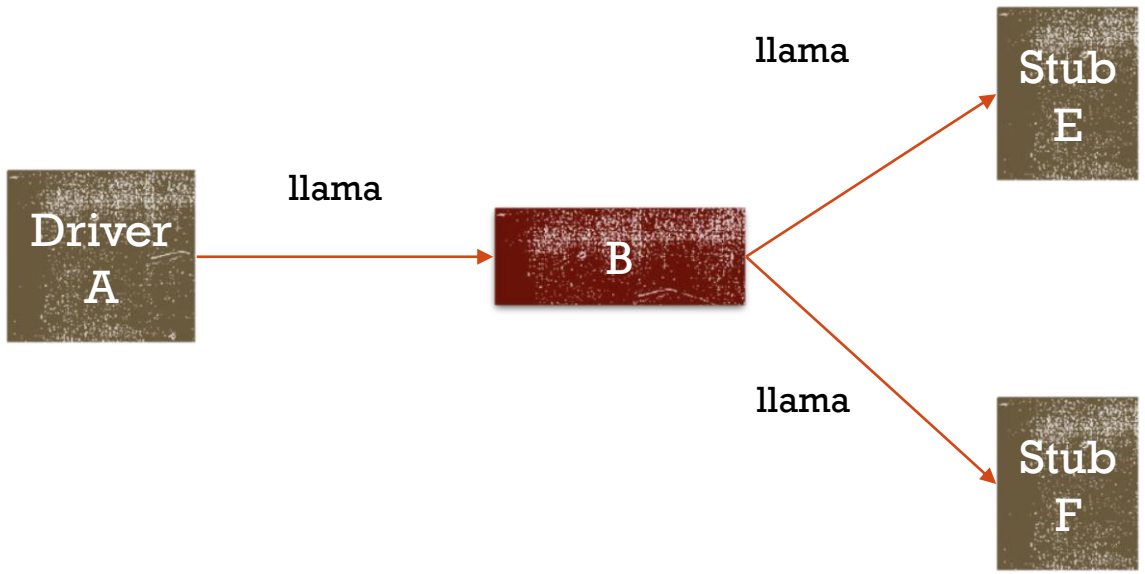
- Es imposible realizar pruebas exhaustivas y probar todas las posibles secuencias de ejecución
- La prueba (test) demuestra la presencia de fallas y nunca su ausencia (Dijkstra)
- Es necesario elegir un subconjunto de las entradas del programa para testear
  - Conjunto de prueba (test set)
- El test debe ayudar a localizar fallas y no solo a detectar su presencia
- El test debe ser repetible
  - En programas concurrentes es difícil de lograr
- **Jacobson sugiere ejecutar primero las pruebas de caja negra y cuando estas sean todas correctas completar con caja blanca.**

PRUEBAS DE INTEGRACIÓN

- Pruebas de Módulos
- Estrategias de Integración
  - No incremental
    - Big-Bang
  - Incremental
    - Bottom-Up
    - Top-Down
    - Sandwich
    - Por disponibilidad



PRUEBA DE MÓDULOS



- Quiero probar al módulo B de forma aislada – No uso los módulos A, E y F
  - El módulo B es usado por el módulo A
    - Debo simular la llamada del modulo A al B – **Driver**
    - Normalmente el Driver es el que suministra los datos de los casos de prueba
  - El módulo B usa a los módulos E y F
    - Cuando llamo desde B a E o F debo simular la ejecución de estos módulos – **Stub**
  - Se prueba al módulo B con los métodos vistos en pruebas unitarias

BIBLIOGRAFÍA UTILIZADA

- Apuntes de Tópicos I, Magister en Ingeniería de Software, Universidad Nacional de La Plata – Alejandro Oliveros y Pablo Thomas.
- Sommerville, I, Ingeniería del software. 9na Edición
- Loucopoulos, P, Karakostas, V, System Requirements Engineering, McGraw Hill, 1995.
- Sommerville, I. Sawyer, P., Requirements Engineering. A good practice guide, John Wiley & Sons, Chichester, 1997