



INGENIERÍA DEL SOFTWARE I

Tema 5

Construcción y Pruebas de Software

Univ. Cantabria – Fac. de Ciencias

Francisco Ruiz



Objetivos

- Comprender que **construir software** engloba muchas **más** actividades que la de **escribir código**.
- Conocer los **principios de la construcción** de software y las **actividades** más significativas.
- Conocer el papel que juegan la **verificación y validación** del software y, dentro de ellas, las **pruebas**.
- Tener una visión general de los **niveles, clases, técnicas y actividades de pruebas** del software.



Contenido

- Construcción
 - Principios
- Proceso de Construcción
 - Planificación
 - Manejo de Excepciones
 - Codificación
 - Pruebas
 - Aseguramiento de Calidad
 - Reutilización
 - Integración
- Lenguajes de Construcción
- Verificación y Validación
- Pruebas
 - Conceptos Básicos
 - Aspectos Clave
- Niveles de Prueba
 - Unitarias
 - Integración
 - Sistema
- Clases de Pruebas
- Técnicas de Prueba
 - Caja Blanca
 - Caja Negra
 - Basadas en Expertez
 - Basadas en Especificación
 - Basadas en Código
 - Basadas en Defectos
 - Basadas en el Uso
 - Otras
- Proceso de Pruebas
 - Actividades Principales
 - Estrategia de Aplicación



Bibliografía

- Básica
 - IEEE Computer Society (2004)
 - SWEBOK - Guide to the Software Engineering Body of Knowledge, 2004 Version.
 - Capítulos 4 y 5.
 - <http://www.swebok.org/>
 - Cap. 23 del libro de Sommerville (2005).
- Complementaria
 - Cap. 10 del libro de Piattini (2007).
 - Caps. 13 y 14 del libro de Pressman (2005).
 - Caps. 7, 8 y 9 del libro de Pfleeger (2002).
 - Cap. 22 del libro de Sommerville (2005).



Construcción

- Se refiere a la creación detallada de **software operativo** mediante una combinación de
 - Codificación,
 - Verificación,
 - Pruebas Unitarias,
 - Pruebas de Integración, y
 - Depuración.
- Este esfuerzo utiliza como entrada las salidas del **diseño** y produce salidas para las **pruebas**.



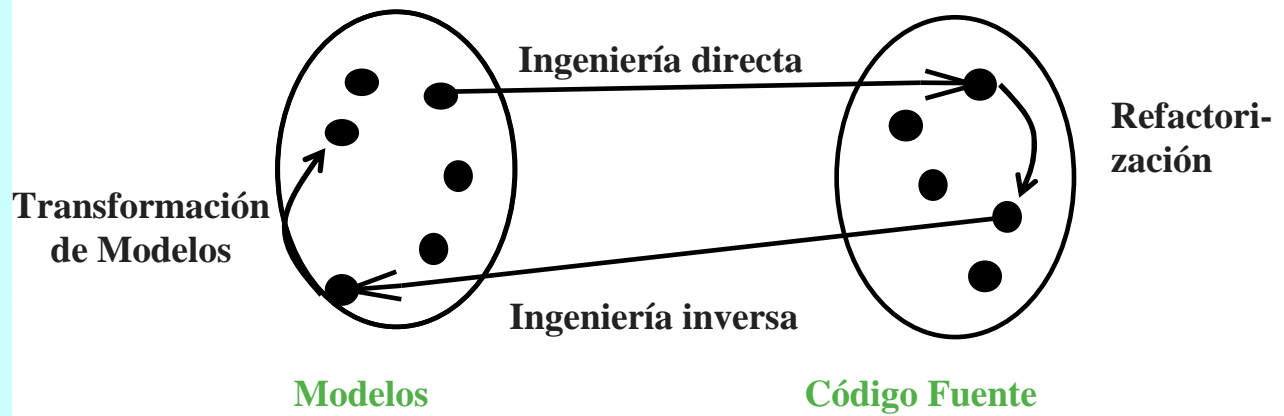
Construcción

- Los límites entre Construcción, Diseño y Pruebas varían dependiendo del ciclo de vida que se usa en cada proyecto.
 - Aunque bastante esfuerzo de **diseño** puede realizarse antes de empezar las actividades de construcción, otro debe realizar en paralelo.
 - Igualmente, algunos tipos de **pruebas** se realizan durante la construcción, mientras que otras se hacen a posteriori.
- Durante la Construcción se generan las cantidades más grandes de artefactos software (ficheros de código, contenidos, casos de prueba, etc.)
 - Esto origina necesidades de **gestión de configuración**.



Construcción

- Hay cuatro tipos de esfuerzos que vinculan los **modelos con el código fuente**



Construcción - Principios

- Los **principios fundamentales** de la construcción de software son:
 - Minimizar la **Complejidad**
 - Anticipar los **Cambios**
 - Pensar en la **Verificación** posterior
 - Aplicar **Estándares**



Construcción - Principios

- **Minimizar la Complejidad**
 - Las **Personas tienen una limitación** para retener estructuras e informaciones complejas en su cabeza, especialmente durante periodos largos de tiempo.
 - Por ello, es **necesario aplicar** el principio de **reducción de la complejidad** a todos los aspectos de la construcción de software.
 - Este objetivo se logra creando **código sencillo y fácil de leer**.



Construcción - Principios

- **Minimizar la Complejidad**
- Maneras de alcanzar este objetivo:
 - Escribiendo **código sencillo y fácil de leer**.
 - Haciendo uso de **estándares**.
 - Aplicando diversas **técnicas de codificación**.
[después]
 - Aplicando **técnicas de aseguramiento de calidad** centradas en la construcción.
[después]



Construcción - Principios

- **Anticipar los Cambios**
 - La mayor parte del software está destinado a cambiar a lo largo del tiempo.
 - El software se ve afectado de forma inevitable por los cambios en su entorno exterior.
- Este objetivo se soporta por diversas **técnicas específicas**.
[después]



Construcción - Principios

- **Pensar en la Verificación posterior**
- Aplicar este principio significa construir software de forma que **los fallos puedan ser encontrados lo antes posible** al codificar, la hacer pruebas independientes o al operar el sistema.
- **Técnicas** específicas que apoyan este principio son:
 - Seguir **estándares de codificación** para soportar las **revisiones de código**;
 - Hacer **pruebas unitarias**;
 - **Organizar el código** para soportar **pruebas automatizadas**; y
 - **Restringir** el uso de **estructuras de lenguaje complejas** o difíciles de entender.



Construcción - Principios

- **Aplicar Estándares**
- Los estándares que impactan directamente en la construcción del software se refieren a
 - Comunicación
 - Formatos de documentos y contenidos
 - Lenguajes de programación
 - Versiones estándares de lenguajes (Java, C++, C#)
 - Reglas de codificación (nombres de variables, comentarios, ..)
 - Plataformas
 - APIs para invocar al sistema operativo
 - Herramientas
 - Notaciones de diagramas (UML)
 - Formatos XML para intercambio entre herramientas diferentes



Construcción - Principios

- **Aplicar Estándares**
- En un proyecto se pueden aplicar dos categorías de estándares:
 - **Externos**: originados por organismos de estandarización (ISO, ANSI, AENOR), consorcios industriales (OMG) o asociaciones profesionales (IEEE, ACM).
 - **Internos**: creados por la propia organización para uso corporativo o para un tipo específico de proyectos.
 - Más específicos que los anteriores
 - Adaptados a la cultura de la organización y su personal



Proceso de Construcción

- Desde una **perspectiva de proceso**, los esfuerzos más significativos que se realizan durante la construcción de software son
 - Planificación
 - Manejo de Excepciones
 - Codificación
 - Pruebas
 - Aseguramiento de Calidad
 - Reutilización
 - Integración



Proceso de Construcción - Planificación

- **Planificar** la construcción consiste, entre otros esfuerzos, en
 - Elegir el **método de construcción**
 - Granularidad y alcance con el que se realizan los requisitos
 - Orden en el que se abordan
 - Establecer el **orden** en el que los **componentes** se crean e integran
 - **Asignar tareas** de construcción a **personas** específicas



Proceso de Construcción – Manejo de Excepciones

- **Manejo de Excepciones** durante la Construcción
- Mientras se construye un edificio, los albañiles deben hacer pequeños ajustes respecto de los planos
- De igual forma, los constructores de software deben hacer **modificaciones y ajustes**, unas veces pequeñas y otras no tanto, respecto de los **detalles del diseño** de un software.



Proceso de Construcción – Codificación

- La **Codificación**, es decir, la escritura del código fuente, es el principal esfuerzo de construcción de software.
- **Codificar software** incluye
 - Aplicar **técnicas** para crear **código fuente comprensible**, incluyendo reglas de asignación de nombres y de formato del código.
 - **Usar** clases, tipos enumerados, variables, constantes etiquetadas y otras **entidades** similares.
 - Usar **estructuras de control**.
 - Manejar **condiciones de error** – tanto errores previstos como excepciones (errores imprevistos).



Proceso de Construcción – Codificación

- **Codificar software** incluye (cont)
 - Prevenir **brechas de seguridad** a nivel de código (llenado de buffers, overflow de índices de vectores, ...)
 - Uso eficiente de **recursos escasos** vía mecanismos de exclusión y disciplinas de acceso a recursos reutilizables de forma serial (hilos, bloqueos en bases de datos, ...)
 - **Organizar el código** fuente (en sentencias, rutinas, clases, paquetes, ...)
 - **Documentar** el código
 - **Ajuste fino** (tuning) del código



Proceso de Construcción – Pruebas

- La Construcción de Software envuelve dos formas de **pruebas**, frecuentemente realizadas por las mismas personas que escriben el código:
 - **Pruebas Unitarias**
 - **Pruebas de Integración**
 - MAS INFO EN APARTADOS DE PRUEBAS
- El propósito de estas pruebas es reducir el tiempo entre el momento en el que los fallos se insertan en el código y el momento en que son detectados.
- Hay dos **estándares** relacionados:
 - IEEE Std 829-1998, IEEE Standard for Software Test **Documentation**.
 - IEEE Std 1008-1987, IEEE Standard for Software **Unit** Testing.



Proceso de Construcción – Aseguramiento de Calidad

- Existen múltiples **técnicas** para **asegurar la calidad** del código durante la construcción:
 - Pruebas (Unitarias y de Integración)
 - Escribir las pruebas primero (*test first development*)
 - Ejecución línea a línea (*code stepping*)
 - Uso de aserciones
 - Depuración (*debugging*)
 - Revisiones
 - Análisis estático



Proceso de Construcción - Reutilización

- **Reutilizar Software** implica más cosas que crear y usar bibliotecas de activos.
- Es necesario oficializar la práctica de reutilización integrando los esfuerzos de reutilización en los ciclos de vida y metodologías de los proyectos.
- Las tareas relacionadas con la reutilización durante la construcción son:
 - **Selección de activos** (unidades, bases de datos, procedimientos de prueba y datos de prueba) **reutilizables**.
 - **Evaluación de la reusabilidad** de código o pruebas.
 - **Documentar la información de reuso** para código nuevo, procedimientos de prueba o datos de prueba.



Proceso de Construcción - Reutilización

• Niveles de Reutilización

■ Código

- librerías de funciones, editores, inclusión de ficheros, mecanismos de herencia, **componentes**, etc.

■ Diseños

- no volver a inventar arquitecturas
 - patrones de diseño, patrones arquitecturales

■ Especificaciones

- abstracciones de dominio

■ Activos [assets]

- Elevar el nivel de reutilización al máximo posible
- Agregación de varios componentes atómicos a distintos niveles de abstracción (*mecano*).



Proceso de Construcción - Reutilización

• Tipos de Activos Reutilizables

- Un **activo** puede encapsular cualquier abstracción útil producida durante el desarrollo de software.

- Planes de proyecto.
- Estimaciones de coste.
- Arquitectura.
- Especificaciones y modelos de requisitos.
- Diseños.
- Código fuente.
- Documentación de usuario y técnica.
- Interfaces hombre-máquina.
- Datos.
- Casos de prueba.



Proceso de Construcción - Integración

- Durante la construcción es clave la **integración** de rutinas, clases, componentes y sub-sistemas construidos de forma separada.
- Además, un sistema software puede tener que ser integrado con otro(s) sistema(s).
- Para ello puede ser necesario:
 - Planificar la secuencia en que se integrarán los componentes;
 - Crear "andamios" para soportar versiones provisionales;
 - Determinar el nivel de pruebas y aseguramiento de calidad realizado sobre los componentes antes de su integración;
 - Establecer puntos del proyecto en los que las versiones provisionales serán probadas.



Lenguajes de Construcción

- **Lenguajes de Construcción** de software son todas las formas de comunicación para que un humano pueda especificar una solución de un problema ejecutable por una computadora.
- Pueden ser de varias **clases**:
 - De Configuración.
 - De Toolkits.
 - De Programación.



Lenguajes de Construcción

- **Lenguajes de Configuración**
 - Permiten elegir entre un conjunto predefinido de opciones para crear instalaciones de software nuevas o particularizadas.
 - Ejemplo: ficheros de configuración de Windows o UNIX.
- **Lenguajes de Toolkits**
 - Permiten construir aplicaciones a partir de un Toolkit (conjunto integrado piezas reutilizables de aplicación específica).
 - **Scripts**: definidos de forma explícita.
 - JavaScript
 - **APIs**: definidos de forma implícita, ya que se implican a partir de la interfaz pública de un Toolkit.
 - API de Office.



Lenguajes de Construcción

- **Lenguajes de Programación**
 - Frente a las otras dos clases:
 - Son más flexibles
 - Tiene menos cantidad de información sobre áreas de aplicación específicas.
 - Requieren mayor aprendizaje y entrenamiento y mayor habilidad para su uso eficiente.
 - Pueden usar tres **tipos de notaciones**:
 - **Lingüística** (cadenas de texto con palabras)
 - **Formal** (expresiones de tipo matemático)
 - **Visual** (símbolos gráficos)



Verificación y Validación

- Contexto General de las Pruebas.
- **VV** es un conjunto de procedimientos, actividades, técnicas y herramientas que se utilizan, paralelamente al desarrollo de software, para **asegurar que un producto software resuelve el problema inicialmente planteado**.
- Las pruebas son una familia de técnicas de **VV**
- Principio:
 - Actuar sobre los productos intermedios que se generan durante el desarrollo para detectar y corregir cuanto antes sus defectos y las desviaciones respecto al objetivo fijado.



Verificación y Validación

- Los **objetivos concretos de la VV** son:
 - Detectar y corregir los **defectos** tan **pronto** como sea posible en el ciclo de vida del software.
 - Disminuir los **riesgos**, las desviaciones sobre los presupuestos y sobre el calendario o programa de tiempos del proyecto.
 - Mejorar la **calidad y fiabilidad** del software.
 - Mejorar la **visibilidad** de la **gestión** del proceso de desarrollo.
 - **Valorar** rápidamente los **cambios** propuestos y sus consecuencias.



Verificación y Validación

- La visión del desarrollo de software como un conjunto de fases con posibles realimentaciones facilita la **VV**.
 - Al inicio del proyecto es necesario hacer un [Plan de VV del SW](#) (IEEE 1012).
 - Las actividades de VV se realizan de forma iterativo durante el desarrollo.
 - IEEE 1012-2004: IEEE **Standard** for Software Verification and Validation.

1. Propósito
2. Documentos de referencia
3. Definiciones
4. Visión general de la verificación y validación
 - 4.1 Organización
 - 4.2 Programa de tiempos
 - 4.3 Esquema de integridad de software
 - 4.4 Resumen de recursos
 - 4.5 Responsabilidades
 - 4.6 Herramientas, técnicas y metodologías
5. Verificación y validación en el ciclo de vida
 - 5.1 Gestión de la VV
 - 5.2 VV en el proceso de adquisición
 - 5.3 VV en el proceso de suministro
 - 5.4 VV en el proceso de desarrollo:
 - 5.4.1 VV de la fase de concepto
 - 5.4.2 VV de la fase de requisitos
 - 5.4.3 VV de la fase de diseño
 - 5.4.4 VV de la fase de implementación
 - 5.4.5 VV de la fase de pruebas
 - 5.4.6 VV de la fase de instalación
 - 5.5 VV de la fase de operación
 - 5.6 VV del mantenimiento
6. Informes de la VV del software
7. Procedimientos administrativos de la VV
 - 7.1 Informe y resolución de anomalías
 - 7.2 Política de iteración de tareas
 - 7.3 Política de desviación
 - 7.4 Procedimientos de control
 - 7.5 Estándares, prácticas y convenciones.
8. Requisitos de documentación para la VV



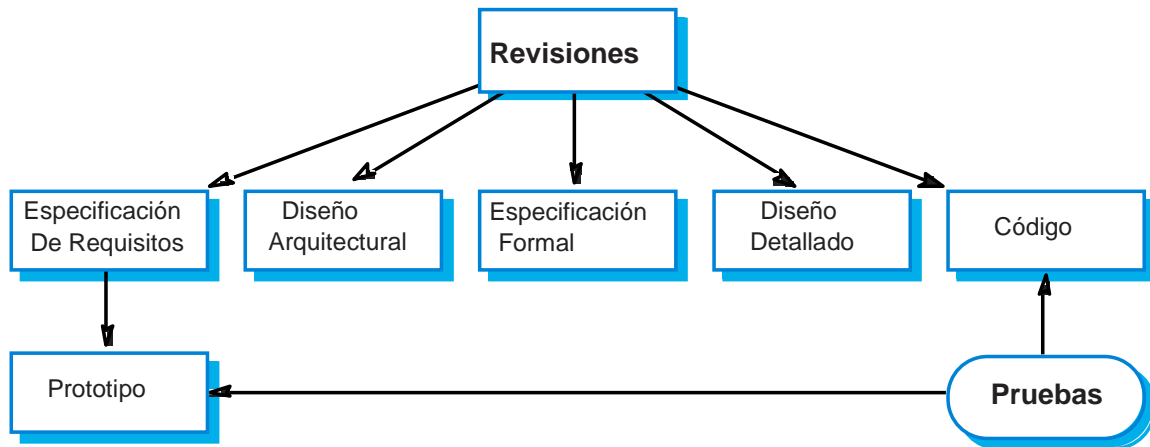
Verificación y Validación

- **Actividades de VV**
 - **Verificación:**
 - ¿Estamos **construyendo correctamente** el producto?
 - El software debe ser conforme a su especificación.
 - Objetivo: Demostrar la consistencia, compleción y corrección de los **artefectos software** entre las **fases** del ciclo de desarrollo de un proyecto.
 - Técnica más utilizada: **Revisiones SW**
 - **Validación:**
 - ¿Estamos construyendo el **producto correcto**?
 - El software debe hacer lo que el usuario realmente quiere
 - Objetivo: Determinar la corrección del **producto final** respecto a las necesidades del usuario.
 - Técnica más utilizada: **Pruebas SW**



Verificación y Validación

- **Técnicas Dinámicas (Pruebas) vs Estáticas (Inspecciones)**



Verificación y Validación

- Las **revisiones** software pueden ser
 - **Informales:**
 - No hay procedimientos definidos, por lo que la revisión se realiza de la forma más flexible posible.
 - Ventajas → menor coste y esfuerzo, preparación corta, etc.
 - Desventajas → Detectan menos defectos
 - **Semi-formales:** Se definen unos procedimientos mínimos a seguir (*walkthroughs*).
 - **Formales (Inspecciones)**
 - Se define completamente el proceso, los participantes y sus funciones, los documentos, etc.



Verificación y Validación

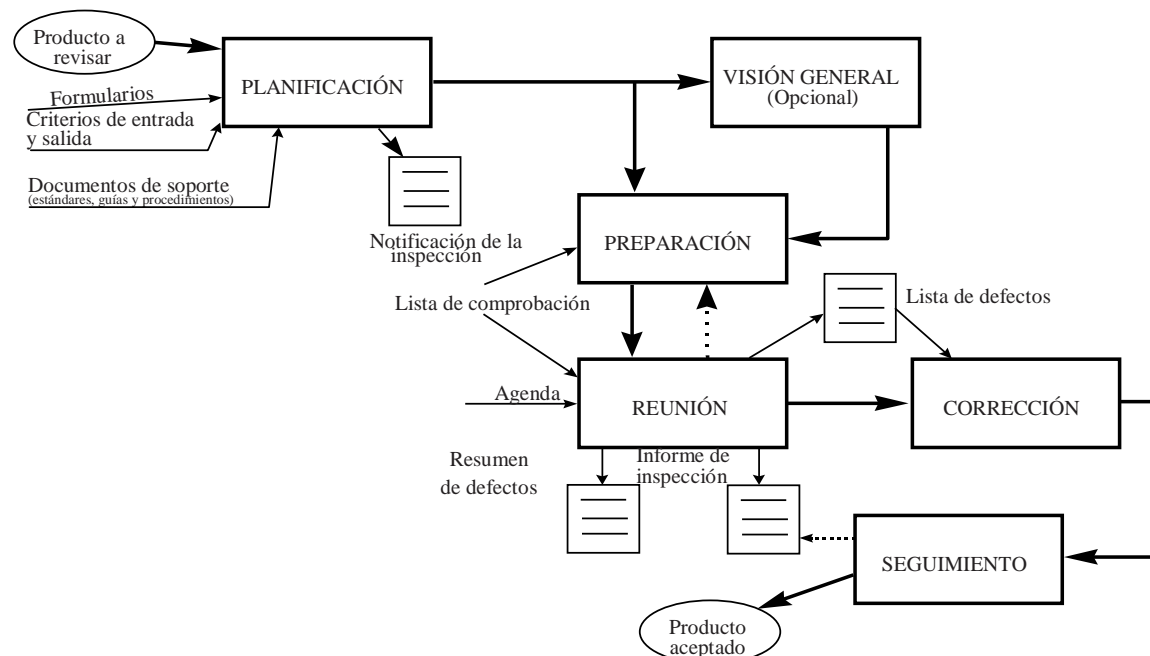
• Inspecciones

- El objetivo es detectar y registrar los **defectos de un producto intermedio**.
- Consisten en
 - Verificar si el producto satisface sus **especificaciones o atributos de calidad**
 - Verificar si el producto se ajusta a los **estándares** utilizados en la empresa.
 - Señalar las **desviaciones** sobre los estándares y las especificaciones.
 - **Recopilar datos** que realimenten inspecciones posteriores (defectos recogidos, esfuerzo empleado, etc.) y ayudar a su utilización práctica.
 - No pretende examinar alternativas o aspectos de estilo.



Verificación y Validación

• Flujo de Trabajo para realizar una Inspección





Verificación y Validación

- Informes de Inspección
 - Ejemplo de Formulario

Informe de inspección

Proyecto _____ Elemento revisado _____
 Fecha de revisión _____ Versión _____
 Documento _____
 Moderador _____
 Revisores _____

Tipo de reunión: Revisión Re-revisión Mantenimiento

Tipo de inspección:

Requisitos (ARS) Especificación (EFS) Arquitectura (DTS)
 Diseño (DTS) Código (DCS) Diseño de Pruebas (DTS)
 Casos de prueba (DCS)

Decisión: Aceptar Aceptar condicionalmente Rechazar

Tipo	Defectos graves				Defectos leves			
	Omisión	Error	Añadido	Total	Omisión	Error	Añadido	Total
Sintaxis								
Compleción								
Nombres								
Consistencia								

Esfuerzo:

- Preparación _____ horas-persona
 - Reuniones _____ horas-persona
 - Seguimiento _____ horas-persona

Firma:

Moderador _____ Secretario _____
 D/Dña. _____ D/Dña. _____



Pruebas

- La manera adecuada de enfrentarse al reto de la **calidad** es la **prevención**
 - *! Mas vale prevenir que curar !*
- Las **Pruebas** deben ser vistas como un medio para
 - Comprobar que la prevención ha sido efectiva, o
 - Identificar defectos en los casos en que no lo ha sido.



Pruebas

- En general, las **Pruebas** (*Testing*) se realizan para evaluar la calidad de un producto, y/o mejorarlo, mediante la identificación de defectos y problemas.
- Las **Pruebas de Software** consisten en la comprobación **dinámica** del comportamiento de un programa para un conjunto **finito** de casos de prueba, convenientemente **seleccionados** entre los, habitualmente infinitos, dominios de ejecución, comparándolo con el comportamiento **esperado**.



Pruebas

- **Comprobación Dinámica**
 - Las pruebas siempre implican la **ejecución** del programa con unas ciertas entradas.
 - Los **valores de entrada** no siempre son suficientes para determinar una prueba porque un sistema complejo podría reaccionar de distinta manera ante una misma entrada dependiendo del **estado del sistema** (sistema no determinista).
 - Las técnicas de verificación **estática** (sin ejecución) se consideran técnicas de aseguramiento de calidad y no pruebas.



- **Conjunto Finito de Casos de prueba**

- Incluso en los programas sencillos, son posibles teóricamente tantos casos de prueba que su realización de forma exhaustiva podría llevar meses o años.
- En la práctica se considera que el conjunto completo de posibles casos de prueba es infinito.
- Hacer pruebas implica un equilibrio entre recursos y tiempo limitados y unos requisitos de prueba virtualmente ilimitados.



- **Seleccionados entre los infinitos**

- Es necesario **seleccionar un subconjunto finito** de casos de prueba.
- La gran cantidad de técnicas de prueba propuestas difieren fundamentalmente en la manera de seleccionar el conjunto de casos de prueba.
- Diferentes **criterios de selección** pueden implicar grados muy distintos de efectividad.
- Identificar el criterio de selección más adecuado para cada situación es muy complejo. En la práctica se aplican:
 - Análisis de Riesgos
 - Experiencia del ingeniero de pruebas



Pruebas

- **Comportamiento Esperado**
 - Para que el **esfuerzo** de hacer pruebas sea **útil** debe ser posible, aunque no siempre fácil, **decidir** si los **resultados** observados al ejecutar el programa son **aceptables** o no.
 - El comportamiento esperado puede ser comparado con
 - Las **expectativas de los usuarios** (pruebas de aceptación)
 - Una **especificación** (pruebas de verificación)
 - Comportamiento anticipado desde **requisitos implícitos**.



Pruebas

- **Ideas** preconcebidas **erróneas** sobre las Pruebas:
 - La **prueba exhaustiva** del software es **impracticable** (no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos)
 - El **objetivo** de las pruebas es la **detección de defectos** en el software
 - **Descubrir un error es el éxito de una prueba**
 - Mito → un defecto implica que somos malos profesionales y que debemos sentirnos culpables → todo el mundo comete errores
 - El descubrimiento de un defecto significa un éxito para la mejora de la calidad



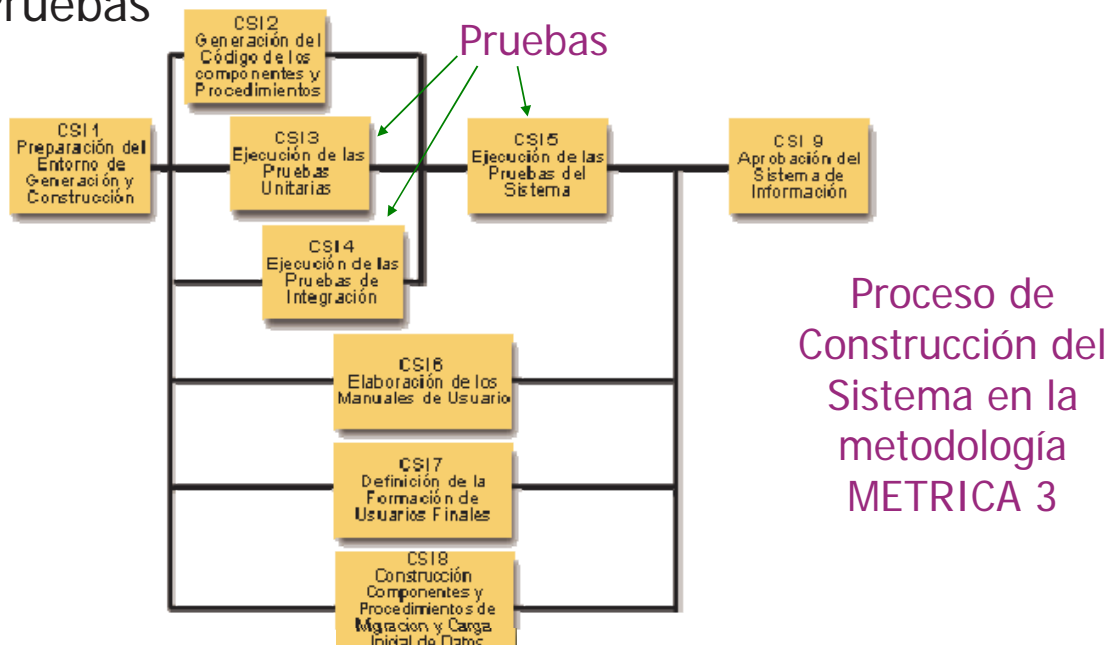
Pruebas

- La manera de pensar las pruebas de software ha evolucionado hacia una perspectiva más constructiva.
 - Son, en sí mismas, una parte clave de la **construcción del producto**.
 - Planes de prueba se llevan a cabo en las etapas iniciales (**requisitos**)
 - Planes y procedimientos de prueba se realizan de forma sistemática y continua a lo largo del **desarrollo y/o el mantenimiento**.



Pruebas

- Ejemplo de la visión integradora de Construcción y Pruebas





Pruebas – Conceptos Básicos

- **Defecto** [*defect, fault, bug*]
 - Causa de un mal funcionamiento de un sistema.
- **Fallo** [*failure*]
 - Efecto no deseado observado en el comportamiento de un sistema.
 - La causa de un fallo (defecto) no siempre puede ser identificado de forma inequívoca.
- Norma sobre terminología:
 - IEEE Standard 610.12-1990, Standard Glossary of Software Engineering Terminology.



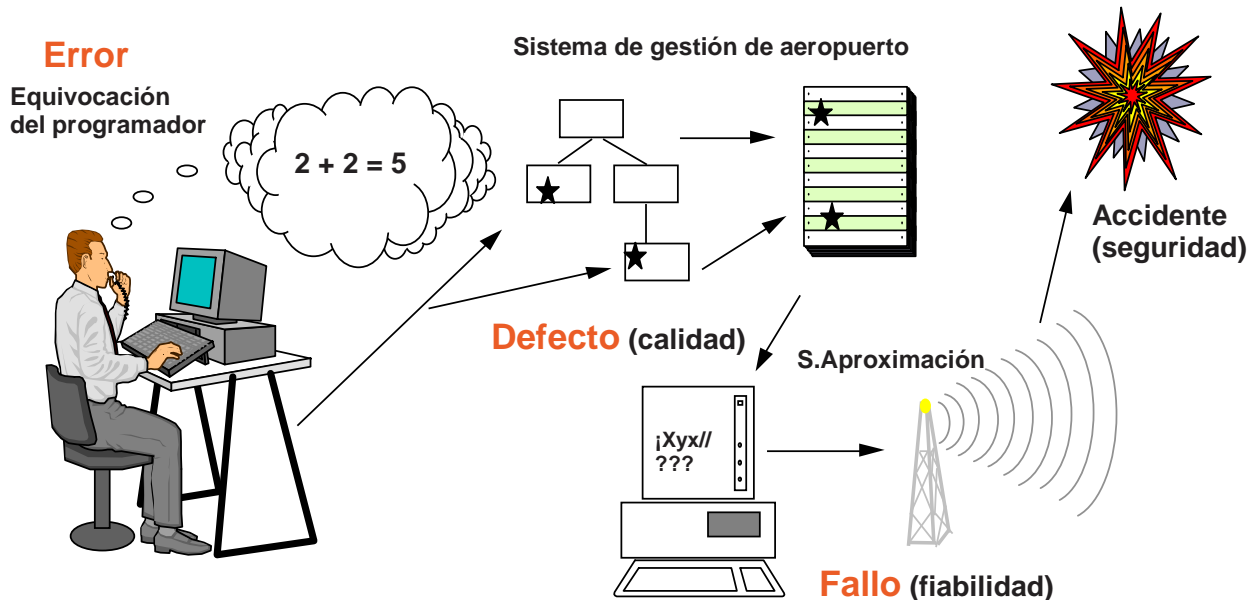
Pruebas – Conceptos Básicos

- **Error**
- Tiene varias acepciones:
 - La diferencia entre un valor calculado, observado o medido y el valor verdadero, especificado o teóricamente correcto.
 - Un defecto
 - Un resultado incorrecto
 - Una acción humana que conduce a un resultado incorrecto.



Pruebas – Conceptos Básicos

- Relación entre Error, Defecto y Fallo:



Pruebas – Aspectos Clave

- Criterio de Selección de Pruebas
 - Medio para decidir cuando un conjunto de casos de prueba es adecuado.
- Eficacia de las Pruebas
 - Probar software es observar ejemplos de ejecución de un programa.
 - La efectividad de las pruebas dependerá de los objetivos de prueba buscados.
- Pruebas para Identificar Defectos
 - En este caso las pruebas son exitosas se causan un fallo del sistema.



Pruebas – Aspectos Clave

- Oráculo
 - Agente (humano o no) que decide cuando un programa actúa correctamente en una prueba dada, emitiendo un veredicto de “correcto” o “fallo”.
- Limitaciones Teóricas y Prácticas de las Pruebas
 - La teoría de pruebas avisa contra la atribución de un nivel de confianza excesiva a una serie de pruebas pasadas.
 - Aforismo de **Dijkstra**:
 - *“Probar programas sirva para demostrar la presencia de errores, pero nunca para demostrar su ausencia”.*
 - En el mundo real no es posible hacer pruebas completas de un software.



Pruebas – Aspectos Clave

- Caminos Impracticables
 - Son opciones de ejecución del flujo de control a las que no se puede llegar con ningún dato de entrada al sistema.
- Comprobabilidad *[testability]*
 - Este término tiene dos significados relacionados:
 - a) Grado en el que es fácil para un software satisfacer un criterio de cobertura de pruebas.
 - b) Probabilidad, normalmente medida estadísticamente, de que un software falle durante las pruebas.



Pruebas – Aspectos Clave

- **Ejemplo** típico de las dificultades que se presentan

```
if (x+y+z)/3=x
    then print ("x,y y z son iguales")
    else print ("x,y y z no son iguales")
```

¿Son suficientes estos dos casos de prueba?

Caso 1: x=5 y=5 z=5

Caso 2: x=2 y=3 z=7

¿Cuáles otros serían necesarios en caso negativo?



Niveles de Prueba

- El ámbito o destino de las pruebas del software puede variar en **tres niveles**:
 - Un **módulo único**.
 - PRUEBAS UNITARIAS
 - Un **grupo de módulos** (relacionados por propósito, uso, comportamiento o estructura).
 - PRUEBAS DE INTEGRACIÓN
 - Un **sistema completo**.
 - PRUEBAS DE SISTEMA



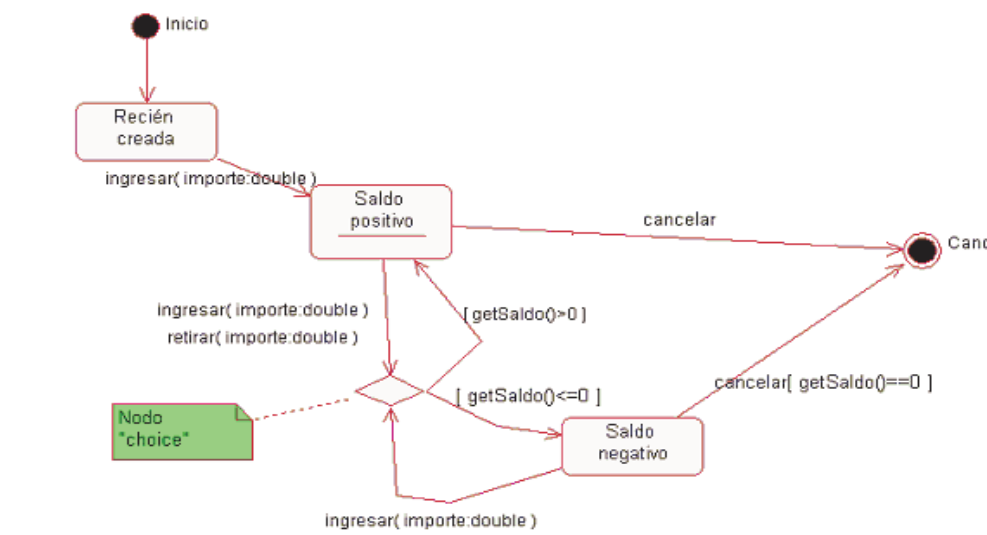
Niveles de Prueba - Unitarias

- Las **Pruebas Unitarias** verifican el **funcionamiento aislado** de piezas de software que pueden ser probadas de forma separada.
- Dichas piezas pueden ser:
 - Subprogramas/Módulos individuales
 - Componente que incluye varios subprogramas/módulos.
- Estas pruebas suelen llevarse a cabo con
 - Acceso al código fuente probado
 - Ayuda de herramientas de depuración
 - Participación (opcional) de los programadores que escribieron el código.
- IEEE Std. 1008 (2002): **Standard** for Software Unit Testing.



Niveles de Prueba - Unitarias

- ¿De dónde obtener **buenos casos de prueba unitarias**?
 - De las **máquinas** o diagramas **de estado**: considerando que definen un lenguaje sobre la unidad de prueba





Niveles de Prueba – Integración

- Las **Pruebas de Integración** verifican la **interacción entre componentes** del sistema software.
- Hay dos tipos de estrategias para llevarlas a cabo:
 - **Incremental**. Tradicionales top-down o bottom-up, para sistemas estructurados de forma jerárquica.
 - Se combina el siguiente módulo que se debe probar con el conjunto de módulos que ya han sido probados.
 - **Guiadas por la arquitectura** (los componentes se integran según “hilos” de funcionalidad).
- Es preferible ir integrando de manera incremental en vez de integrar de golpe todos los componentes de un sistema (*big bang testing*).



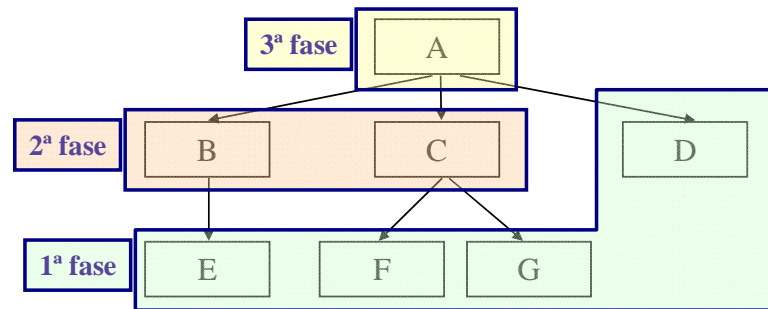
Niveles de Prueba – Integración

- Implican una progresión ordenada de pruebas que van desde los componentes o módulos y que culminan en el sistema completo.
- El **orden de integración** elegido afecta a diversos factores, como lo siguientes:
 - La forma de preparar casos
 - Las herramientas necesarias
 - El orden de codificar y probar los módulos
 - El coste de la depuración
 - El coste de preparación de casos



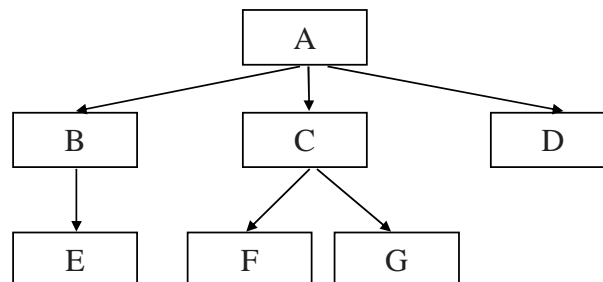
Niveles de Prueba – Integración

- Estrategia **Incremental Ascendente** (Bottom-Up)
 - Se comienza por los módulos hoja (pruebas unitarias).
 - Unitarias de E F G y D
 - Se combinan los módulos según la jerarquía y a cada grupo se le hacen pruebas de integración.
 - B con E, C con F y G
 - Se repite en niveles superiores.
 - Integración de A con B, C y D



Niveles de Prueba – Integración

- Estrategia **Incremental Descendente** (Top-Down)
 - Comienza por el módulo de control principal y va incorporando módulos subordinados progresivamente.
 - Existen dos formas básicas de hacer esta integración:
 - Primero en profundidad: completando ramas del árbol (A B E C F G D)
 - Primero en anchura: completando niveles de jerarquía (A B C D E F G)





Niveles de Prueba - Sistema

- Las **Pruebas de Sistema** verifican el **comportamiento del sistema** en su conjunto.
- Los fallos funcionales se suelen detectar en los otros dos niveles. Este nivel es más adecuado para comprobar **requisitos no funcionales**.
 - Seguridad, Velocidad, Exactitud, Fiabilidad
- En este nivel también se prueban
 - Interfaces externos con otros sistemas
 - Utilidades
 - Unidades físicas
 - Entorno operativo



Clases de Pruebas

- Dependiendo de su **finalidad**, las pruebas de software se pueden clasificar por varios criterios.
- En general, diferentes finalidades se suelen abordar a diferentes niveles de prueba.
- Ejemplo:
 - Niveles Unitario + Integración
 - Comprobar que las especificaciones **funcionales** están implementadas correctamente.
 - PRUEBAS DE CONFORMIDAD
 - PRUEBAS DE CORRECCIÓN
 - PRUEBAS FUNCIONALES
 - Nivel Sistema
 - Comprobar los requisitos **no funcionales**.



Clases de Pruebas

- De **Aceptación** [cualificación]
 - Comprueban el comportamiento del sistema frente a los **requisitos del cliente**.
 - El mismo cliente o sus usuarios suelen participar.
- De **Instalación**
 - Después de completar el software y hacer sus pruebas de aceptación, estas pruebas verifican su instalación en el entorno de destino.
 - Comprueban el comportamiento del sistema frente a los **requisitos de configuración del hardware**.



Clases de Pruebas

- **Alfa y Beta**
 - Antes de difundir una versión del software, se pasa a un **grupo pequeño de usuarios potenciales** representativos para su prueba en uso.
 - En la misma empresa (**alfa**)
 - Fuera, en casa del usuario (**beta**)
- De **Conformidad** [funcionales / corrección]
 - Validar si el comportamiento observado del software es conforme con sus especificaciones (requisitos funcionales).



Clases de Pruebas

- De **Fiabilidad** [reliability]
 - Evaluar la fiabilidad de un software mediante
 - Generación aleatoria de casos de prueba de acuerdo con un perfil operacional, y
 - Tomando medidas estadísticas de la fiabilidad.
- De **Regresión**
 - Probar de nuevo de forma selectiva un sistema o componente para verificar que las modificaciones sufridas no han causado efectos no deseados.
 - Se pueden hacer para los tres niveles.
 - Y para propiedades funcionales o no funcionales.



Clases de Pruebas

- De **Rendimiento** [performance]
 - Verificar que el software satisface los requisitos de rendimiento.
 - Tiempo de respuesta
 - Limitaciones de espacio de memoria.
- De **Estrés**
 - Probar la reacción del software con el máximo de carga prevista e incluso más aún.
- **Espalda con Espalda** [back to back]
 - Una mismo conjunto de pruebas es realizado en dos implementaciones diferentes del mismo producto y se comparan los resultados.



Clases de Pruebas

- De **Recuperación**
 - Verificar las capacidades de restauración del software después de un desastre.
- De **Configuración**
 - Analizar el software bajo configuraciones diferentes para diferentes usuarios.
- **Usabilidad**
 - Evaluar cómo de fácil es usar y aprender a usar el software para usuarios finales. Incluye:
 - Documentación de usuario,
 - Cómo de eficiente es el soporte de las tareas de usuario por las funciones del software, y
 - Abilidad para recuperarse errores del usuario.



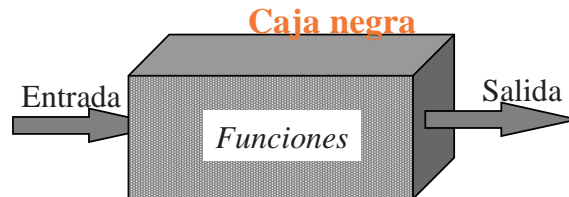
Técnicas de Prueba

- Uno de los **objetivos** de las pruebas de software es intentar **“romper” el programa**.
 - Se trata de encontrar el mayor número de fallos posible.
- El **principio** subyacente en estas técnicas es intentar ser lo más **systemático** posible en identificar un **conjunto representativo de comportamientos** del programa, determinados por
 - subclases del dominio de entradas,
 - escenarios,
 - estados y/o
 - flujos de datos.

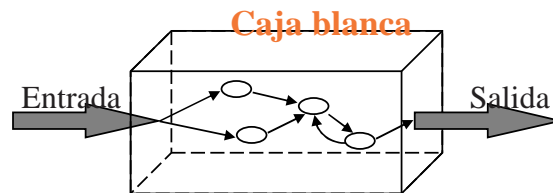


Técnicas de Prueba

- Existen multitud de técnicas de prueba del software.
- De forma general, existen dos enfoques diferentes:
 - **Caja Negra (Funcional)**. Los casos de prueba se basan sólo en el comportamiento de entrada/salida.

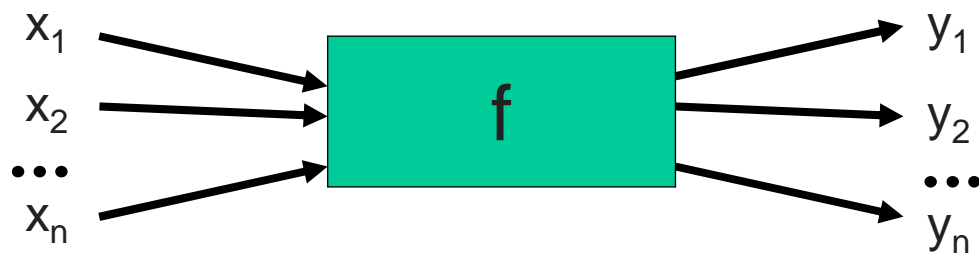


- **Caja Blanca (Estructural)**. Basadas en información sobre cómo el software ha sido diseñado o codificado.



Técnicas de Prueba

- **Caja Negra**



Es decir,

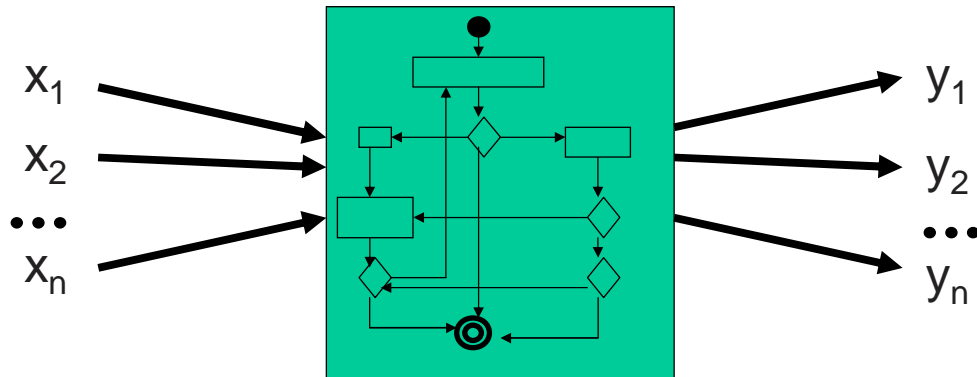
$\exists y_i = f(x_i)$, para todo i ?



Técnicas de Prueba

- **Caja Blanca**

- De alguna manera, me interesa conocer la **cobertura** alcanzada por mis casos de prueba dentro de la unidad de prueba



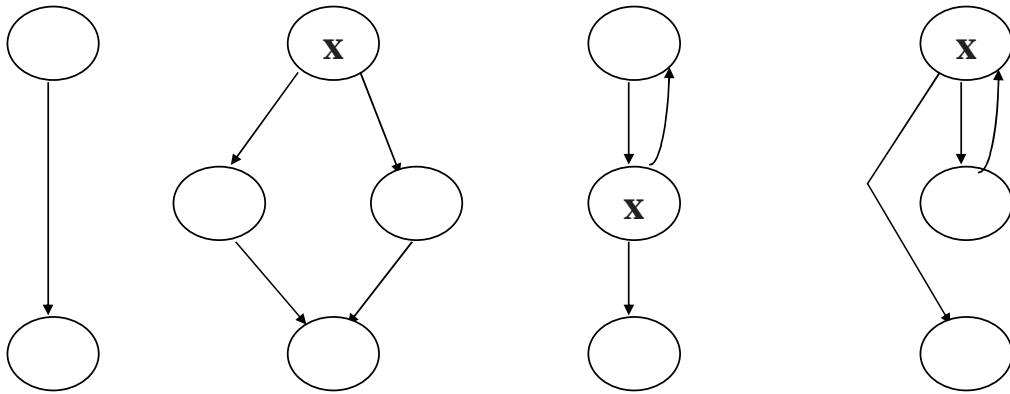
Técnicas de Prueba

- Otro criterio más preciso clasifica las técnicas de prueba según la fuente a partir de la que **son generados los casos de prueba**:
 - La **expertez** (experiencia e intuición) del ingeniero de pruebas.
 - Las **especificaciones**.
 - La **estructura del código**.
 - Los **defectos** (reales o artificiales) que se quieren descubrir.
 - El **campo de uso**.
 - La **naturaleza de la aplicación**.



Técnicas de Prueba – Caja Blanca

- La estructura de control sirve de base para obtener los casos de prueba
- Para la representación del flujo de control se utilizan **Grafos de Flujo**

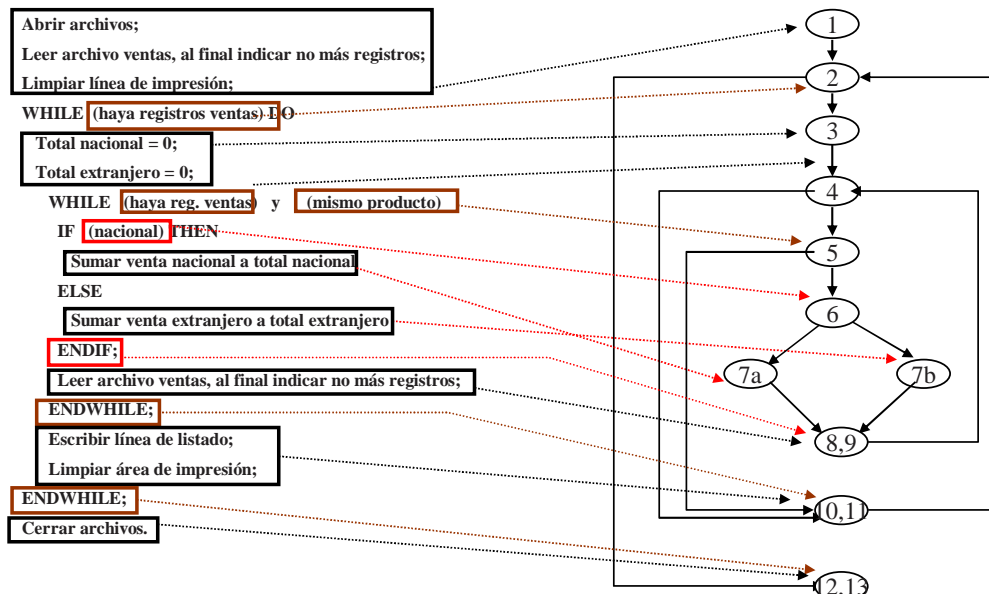


Secuencia **Si x entonces...** **Hacer... hasta x** **Mientras x hacer...**
 (If x then...else...) (Do...until x) (While x do...)



Técnicas de Prueba – Caja Blanca

- Grafo de Flujo de un Programa (Pseudocódigo)



El diseño de casos de prueba tiene que estar basado en la elección de caminos importantes que ofrezcan una seguridad aceptable de que se descubren defectos

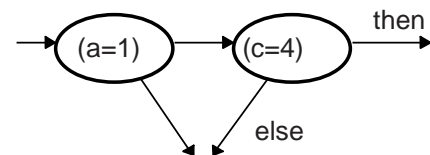
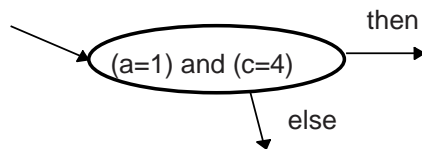


Técnicas de Prueba – Caja Blanca

- Criterios de Cobertura:

- ✓ **Cobertura de sentencias.** Que cada sentencia se ejecute al menos una vez.
- ✓ **Cobertura de decisiones.** Que cada decisión tenga, por lo menos una vez, un resultado verdadero y, al menos una vez, uno falso.
- ✓ **Cobertura de condiciones.** Que cada condición de cada decisión adopte el valor verdadero al menos una vez y el falso al menos una vez.
- ✓ **Criterio de decisión/condición.** Que se cumplan a la vez el criterio de condiciones y el de decisiones.
- ✓ **Criterio de condición múltiple.** La evaluación de las condiciones de cada decisión no se realiza de forma simultánea.

Descomposición de una Decisión Multicondicional



Técnicas de Prueba – Caja Negra

- Están orientadas a los **requisitos funcionales** del software
- Algunas **Técnicas** de Pruebas de Caja Negra [después]
 - Particiones o Clases de Equivalencia
 - Análisis de Valores Límite
 - Conjetura de Errores



Técnicas de Prueba – Basadas en Expertez

- **Ad Hoc**

- Las pruebas dependen totalmente de la habilidad, intuición y experiencia con programas similares del **ingeniero de pruebas**.

- **Exploratorias**

- A la vez, se lleva a cabo aprendizaje, diseño de pruebas y ejecución de pruebas.
- Las pruebas se diseñan, ejecutan y modifican de forma dinámica, **sobre la marcha**.
- La eficiencia depende fundamentalmente del conocimiento del ingeniero de pruebas.



Técnicas de Prueba – Basadas en Especificación

- **Particionamiento Equivalente**

- El dominio de las **entradas** se divide en una colección de subconjuntos (**clases de equivalencia**) que son equivalentes respecto de una relación especificada:
 - La prueba de un valor representativo de una clase permite suponer «razonablemente» que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro valor de la clase.
- Se realiza un conjunto representativo de casos de prueba para cada clase de equivalencia.



• **Particionamiento Equivalente**

■ **Ejemplo:** Entrada de un Programa

- Código de Área: Número de tres cifras que no comienza ni por 0 ni por 1
- Nombre: Seis caracteres
- Orden: cheque, depósito, pago factura, retirada de fondos

Condición de entrada	Clases válidas	Clases inválidas
Código área	(1) $200 \leq \text{código} \leq 999$	(2) código < 200 (3) código > 999 (4) no es número
Nombre para identificar la operación	(5) seis caracteres	(6) menos de 6 caracteres (7) más de 6 caracteres
Orden	(8) «cheque» (9) «depósito» (10) «pago factura» (11) «retirada de fondos»	(12) ninguna orden válida



• **Análisis de Valores Límite**

- Similares a las anteriores, pero cambia que los valores de entrada de los casos de prueba se eligen en las cercanías de los **límites** de los dominios de **entrada** de las variables.
- **Suposición:** Muchos defectos tienden a concentrarse cerca de los valores extremos de las entradas.
- Extensión: **Pruebas de Robustez**, con casos fuera de los dominios de entrada.



Técnicas de Prueba – Basadas en Especificación

- **Tablas de Decisión**

- Se usan tablas de decisión para representar relaciones lógicas entre condiciones (entradas) y acciones (salidas).
- Los casos de prueba se derivan sistemáticamente de cada **combinación de condiciones y acciones**.

- **Máquinas de Estados Finitos**

- Los programas se modelan como máquinas de estados finitos.
- Los casos de prueba pueden ser seleccionados de forma que cubren los **estados y las transiciones** entre ellos.



Técnicas de Prueba – Basadas en Especificación

- Basadas en **Especificación Formal**

- Disponer de las especificaciones en un **lenguaje formal** permite la derivación automática de casos de prueba.
- A la vez, se provee una salida de referencia (oráculo) para chequear los resultados de las pruebas.
 - Basadas en Modelos.
 - Basadas en Especificaciones Algebraicas.

- **Aleatorias**

- Las Pruebas son generadas de forma plenamente **aleatoria**.
- Requiere el conocimiento del dominio de las entradas.
 - Generación aleatoria de datos de entrada con la secuencia y la frecuencia con las que podrían aparecer en la práctica.



Técnicas de Prueba – Basadas en Código

- Basadas en **Flujo de Control**
 - Se usan criterios que buscan cubrir todas las sentencias o bloques de sentencias de un programa.
 - El criterio más fuerte es la prueba de caminos (*testing path*).
 - Ejecución de todos los caminos de flujo de control entrada-salida.
 - Otros criterios menos exigentes:
 - Prueba de sentencias
 - Prueba de ramas (branch)
 - Prueba de condiciones/decisiones
 - Los resultados de estas pruebas se establecen en **porcentaje de cobertura**.



Técnicas de Prueba – Basadas en Código

- Basadas en **Flujo de Datos**
 - El diagrama de flujo de control es anotado con información sobre cómo se definen, usan y eliminan las variables del program.
 - El criterio más fuerte busca probar todos los caminos de definición-uso:
 - Para cada variable, se debe ejecutar cada segmento de camino de flujo de control desde la definición de la variable a su uso.
 - Criterios menos exigentes:
 - Prueba de todas las definiciones.
 - Prueba de todos los usos.



Técnicas de Prueba – Basadas en Defectos

- Las **Pruebas** basadas en **Defectos** “inventan” los casos de prueba con el objetivo de descubrir categorías de defectos probables o previstos.
- **Conjetura de Errores** [error guessing]
 - Los casos de prueba se diseñan intentando **descubrir** los defectos más plausibles del programa.
 1. Enumerar una lista de posibles equivocaciones que pueden cometer los desarrolladores y de las situaciones propensas a ciertos errores.
 - Ejemplo: El valor cero es una situación propensa a error
 2. Generar los casos de prueba en base a dicha lista (se suelen corresponder con defectos que aparecen comúnmente y no con aspectos funcionales).
 - Fuente: Historia de defectos en proyectos previos.



Técnicas de Prueba – Basadas en Defectos

- **Mutación**
 - Un mutante es una **versión ligeramente modificada** de un programa. La diferencia es un pequeño **cambio sintáctico**.
 - Cada caso de prueba se aplica al original y a los mutantes generados.
 - Asunción: Mirando defectos sintácticos sencillos se encuentran defectos reales más complejos.
 - Para ser eficiente se requieren muchos mutantes, que deben ser generados automáticamente de forma sistemática.
 - Hay herramientas disponibles para ello.



Técnicas de Prueba – Basadas en el Uso

- **Perfil Operacional**

- Se reproduce y se prueba el entorno operativo en que deberá funcionar el programa.
- Se trata de inferir, a partir de los resultados observados, la futura fiabilidad del software cuando esté en uso.

- **SRET** (*Software Reliability Engineered Testing*)

- Método en el cual las pruebas son “diseñadas y guiadas por objetivos de fiabilidad y criticidad de las diferentes funcionalidades”.



Técnicas de Prueba – Otras

- Además de las anteriores, existen **técnicas específicas** para **ciertos tipos de software**:

- Orientado a Objetos. [LABORATORIO ASIGNATURA]
- Basado en Componentes.
- Basado en Web.
- Interfaz de Usuario.
- Programas Concurrentes.
- Conformidad de Protocolos.
- Sistemas en Tiempo Real.
- Sistemas Críticos (seguridad).

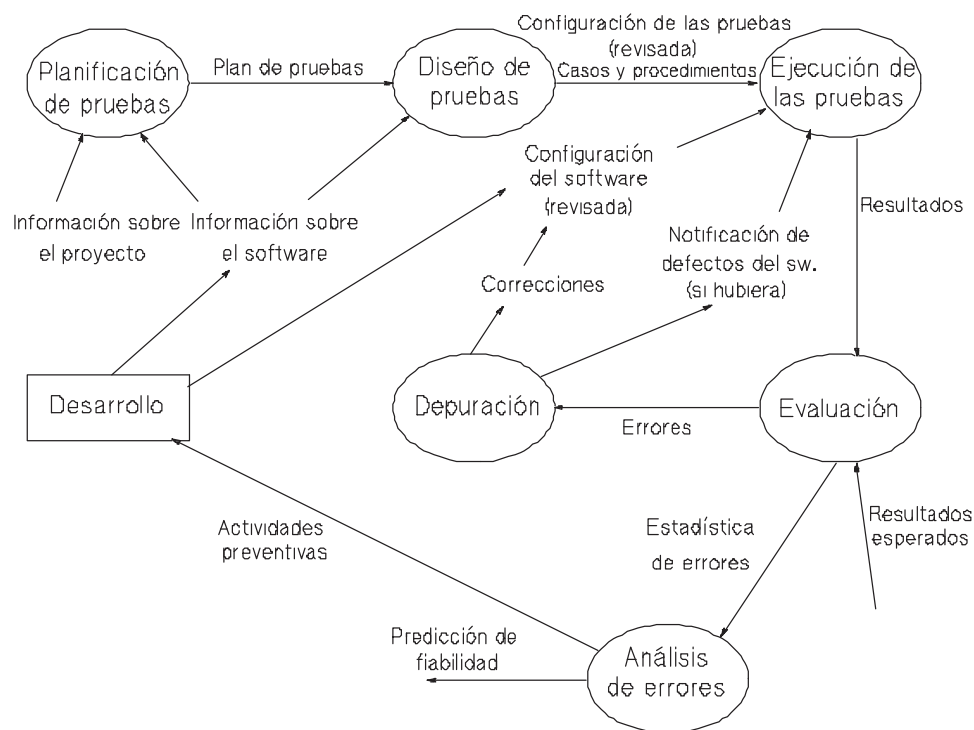


Proceso de Pruebas

- El **proceso de pruebas** establece
 - Como llevar a cabo las actividades de prueba
 - Guiando al equipo de pruebas
 - Desde la planificación de las pruebas hasta la evaluación de los resultados de las pruebas
 - Dando garantías razonables de los objetivos de las pruebas se alcanzarán de forma eficiente.
- Un componente muy importante del éxito de las pruebas es la actitud colaborativa
 - Especialmente importante es el apoyo de los directivos.



Proceso de Pruebas





Proceso de Pruebas

- El **proceso de pruebas** suele tener varias **fases** diferentes
- Cada fase puede estar guiada por objetivos de prueba diferentes.
- Ejemplos:
 - Pruebas basadas en riesgos: usa los riesgos del producto para priorizar y enfocar la estrategia de pruebas.
 - Pruebas basadas en escenarios: los casos de prueba se definen en base a los escenarios software especificados.



Proceso de Pruebas

- Las pruebas deben ser **gestionadas**
 - Actividades, personas, herramientas, políticas y mediciones deben estar organizadas en un proceso bien definido, que es parte del ciclo de vida del software.
 - ISO 12207 no establece un proceso autónomo de pruebas, sino que sus actividades se reparten entre los cinco procesos principales y los procesos de soporte.
- La **documentación** de las pruebas incluye
 - Planes
 - Especificación del Diseño
 - Especificación de Procedimientos
 - Especificación de Casos
 - Registro (Log)
 - Incidentes
 - Informes de Problemas
 - Items (Software)



Proceso de Pruebas

- El **Equipo de Pruebas** puede ser
 - Interno (miembros del equipo del proyecto),
 - Externo (de fuera del equipo, misma o distinta organización)
 - Mixto (miembros internos y externos)
- **Terminación de las Pruebas**
 - Es necesario tomar la decisión de cuando las pruebas son suficientes y cuando terminar una fase de pruebas.
 - La decisión viene determinada por
 - Indicadores de los resultados de las pruebas (% cobertura, ...)
 - Análisis de costes y riesgos
 - De los fallos remanentes vs continuar las pruebas



Proceso de Pruebas

- Para llevar a cabo pruebas software de una manera organizada y eficiente en costes es necesario que los **medios** usados para probar cada parte del software puedan ser **reutilizados** de forma sistemática.
 - El repositorio de materiales de prueba debe estar bajo el control del Sistema de **Gestión de Configuración (CMS)**.
- Un **patrón de pruebas** puede ser reutilizado en proyectos similares.
 - Soluciones adoptadas para probar algún tipo de aplicación bajo ciertas circunstancias, incluyendo las motivaciones de las decisiones adoptadas.



Proceso de Pruebas – Actividades Principales

- **Planificación**
 - Las actividades de prueba deben ser planificadas.
 - Aspectos claves de dicha planificación son:
 - Coordinación del personal
 - Gestión de los recursos y equipamiento de pruebas disponibles
 - Prever posibles resultados no deseados
- **Generación de Casos de Prueba**
 - Está basada en los niveles de pruebas a realizar y en las técnicas particulares seleccionadas.
 - Los casos de prueba deben estar bajo el control del CMS y deben incluir los resultados esperados de cada prueba.



Proceso de Pruebas – Actividades Principales

- **Preparación del Entorno de Pruebas**
 - El entorno tecnológico para hacer las pruebas debe ser compatible con las herramientas CASE usadas en el proyecto. Debe facilitar:
 - El desarrollo y control de casos de prueba
 - Registro y recuperación de resultados esperados, scripts y demás materiales de las pruebas.
- **Ejecución**
 - La realización de las pruebas debe respetar un principio básico de la experimentación científica:
 - **Replicabilidad.** Cada cosa hecha durante las pruebas debe ser realizada y documentada lo suficientemente claro para que otra persona pueda replicar los resultados.



Proceso de Pruebas – Actividades Principales

- **Evaluación de los Resultados**

- Los resultados deben ser evaluados para determinar si las **pruebas** han sido o no **exitosas**.
- En muchos casos “exitosas” significa que el software se comportó como se esperaba y no hubo ningún **resultado inesperado** importante.
- No todos los resultados inesperados son necesariamente defectos. Algunos pueden catalogarse simplemente como “**molestias**”.
- Es conveniente la **revisión formal** por el comité correspondiente de los resultados de pruebas particularmente **importantes**



Proceso de Pruebas – Actividades Principales

- **Informe de Problemas** [test log]

- Un registro de pruebas suele identificar
 - Cuando ha sido realizada una prueba
 - Quién la realizó
 - Qué items de configuración del software fueron la base para las pruebas
 - Otra información relevante
- Un **sistema de registro de incidencias** puede ser utilizado para anotar los resultados incorrectos o imprevistos.
- Las **anomalías** no clasificadas como defectos también pueden ser documentadas por si en un futuro se consideran más graves.



Proceso de Pruebas – Actividades Principales

- **Traza de Defectos** [defect tracking]
 - Los **fallos** observados durante las pruebas son a menudo debidos a **defectos** en el software.
 - Estos defectos pueden ser **analizados** para determinar:
 - **cuando** fueron **introducidos** en el software;
 - que **clase de causa** motivó que aparecieran (requisitos pobremente definidos, declaración incorrecta de una variable, error de sintáxis en el código, ...); y
 - **cuando** podrían haber sido **observados** por primera vez.
 - Además de para lo anterior, la **información de traza de defectos** sirve también para determinar los **aspectos que necesitan mejorar** y el nivel de **eficiencia** de los análisis y pruebas anteriores.



Proceso de Pruebas – Estrategia de Aplicación

- De forma general, **¿Cuándo se debe realizar cada nivel y tipo de pruebas?**
 - Se comienza en la **prueba** de cada **módulo**, que normalmente la realiza el propio personal de desarrollo en su entorno → **Pruebas Unitarias**
 - Con el esquema del diseño del software, los **módulos** probados se **integran** para comprobar sus interfaces en el trabajo conjunto → **Pruebas de Integración**
 - El software totalmente ensamblado se **prueba como un conjunto** para comprobar si **cumple o no** tanto los **requisitos** funcionales como los requisitos de rendimiento, seguridad, etc. → **Pruebas Funcionales + Pruebas de Validación**



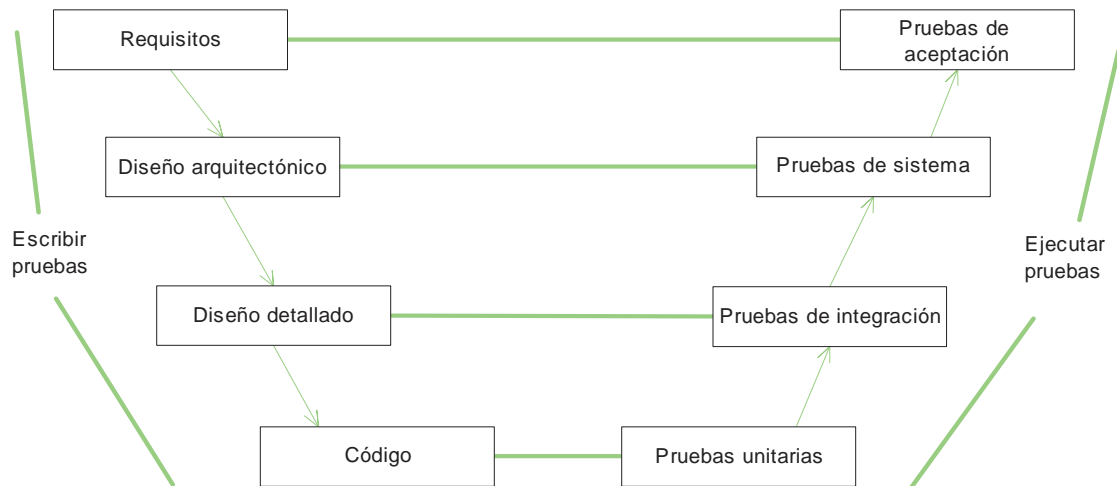
Proceso de Pruebas – Estrategia de Aplicación

- De forma general, **¿Cuándo se debe realizar cada nivel y tipo de pruebas?** (cont)
 - El software ya validado se integra con el resto del sistema de información (por ejemplo, elementos mecánicos, interfaces electrónicas, etc.) para probar su funcionamiento conjunto → **Pruebas de Sistema**
 - El producto final se pasa para que el usuario compruebe en su propio entorno de explotación si lo acepta como está o no → **Pruebas de Aceptación**



Proceso de Pruebas – Estrategia de Aplicación

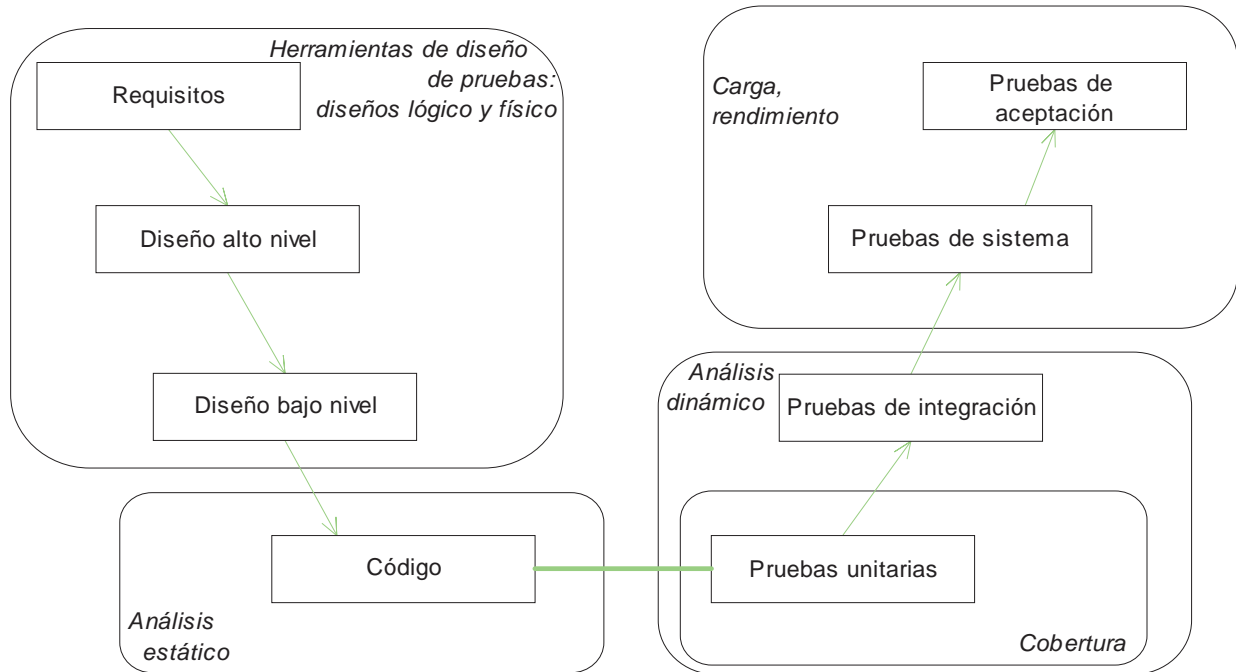
- Un **modelo en V** ilustra cuándo deben realizarse actividades de pruebas: cada actividad del desarrollo tiene su correspondiente actividad de pruebas.



Relación entre **Actividades de Desarrollo y Pruebas**



Proceso de Pruebas – Estrategia de Aplicación



Automatización de las Pruebas