CC51A – Ingeniería de Software

Pruebas de Software

Sergio Ochoa D.

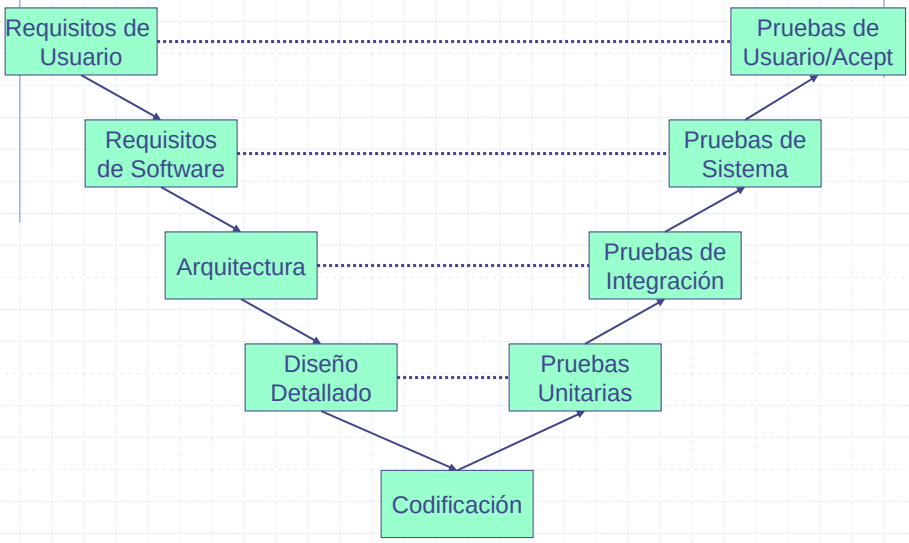
Objetivo

- Conocer los distintos tipos de pruebas a los que debe someterse un sistema de software.
- En esta clase veremos
  - Pruebas unitarias
  - Pruebas de integración
  - Pruebas de sistemas
  - Pruebas de usuario
  - Pruebas de regresión

¿Por qué probamos el software?

- Para asegurar que el sistema es consistente con los requisitos y las necesidades de los usuarios y clientes.
- Para hacer un primer diagnóstico de la calidad del producto desarrollado.
- Para ello, hay varios tipos de pruebas:
  - Como profesional hazlas a todas
  - En este curso bastan pruebas de sistema y de usuario.

Modelo de ciclo de vida en V

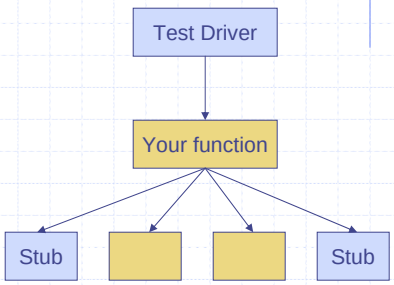


Tipos de pruebas

- Pruebas unitarias o de módulos (*unit testing*)
- Pruebas de integración (*integration testing*)
- Pruebas de sistema (*system testing*)
- Pruebas de Usuario/Aceptación (*qualification*)
- Pruebas de regresión (*regression testing*)

Pruebas unitarias

- Propósito: Encontrar defectos en un módulo o función.
- Normalmente las pruebas unitarias son hechas por el programador del módulo.
- Para probar una función debes:
  - Llamarla
  - Tener subfunciones para invocar



Diseño de pruebas unitarias

- Las pruebas de cada función se especifican junto a la especificación de cada función.
- Debemos asegurarnos de incluir pruebas para:
  - Casos habituales y casos extremos de los parámetros.
  - Todas las excepciones.
  - Todos los tipos de efectos laterales.
- Un mismo caso de prueba puede mostrar una excepción y un efecto lateral.

Especificación de un prueba unitaria

Parámetro	**Valor**
Parámetro$_1$	*Valor del parámetro 1*
Parámetro$_2$	*Valor del parámetro 2*
Parámetro$_3$	*Valor del parámetro 3*
Retorno	*Valor retornado o bien* '-'.
Estado Inicial	*Condiciones del estado inicial del sistema o bien* '-'.
Excepción	*Excepción que se devuelve en este caso o bien* '-'.
Efecto	*Efectos laterales para este caso o bien* '-'.
Observación	Qué sucedió?
Resultado	No probado/Cumple/No Cumple

Funciones de un módulo

- Lo anterior se apunta a probar cada función por separado.
  - Si las funciones de un módulo interactúan, debemos probar las interacciones
  - (Si no interactúan el módulo tiene cero cohesión)
- La estrategia de prueba de un módulo va junto a su especificación funcional.
- El límite entre la prueba de un módulo complejo y la prueba de integración es muy fino:
  - un módulo complejo es un subsistema que debe integrarse.

Herramientas

- Hay herramientas que generan casos de prueba en forma “inteligente”, de alguna manera eligiendo un número relativamente limitado de casos de prueba (robots).
- Las herramientas pueden probar los casos (generan código para el test driver).
- Las herramientas computan cobertura de código e incluso cobertura de caminos.
- Para sistemas embebidos de comunicaciones (como radios de dos vías), las herramientas se comunican con un modelo ejecutable de otros sistemas con los que se interactúa.

Pruebas de integración

- Las ***pruebas de integración*** son realizadas sobre agrupaciones de componentes/funciones.
  - Se examinan las interfaces para asegurar que estos componentes individuales son llamados cuando es necesario y que los datos que se transmiten entre dichos componentes son los requeridos.
- Propósito: Asegurar que no hay errores de interfaces; encontrar defectos en el sistema.
- Debemos planificar las pruebas de integración:
  - En el Documento de Arquitectura, o bien en el Plan de Pruebas de Sistema.
  - Hay que destinar recursos a esto (gente y tiempo).
- Típicamente se necesita andamiaje e incluso simuladores al comienzo de la integración.

Mix: Pruebas Unitarias y de Integración

- Al probar cada módulo individualmente es necesario crear módulos auxiliares que simulen las acciones de los módulos invocados por el módulo que se está probando.
- Asimismo se han de crear módulos "conductores" para establecer las pre-condiciones necesarias, llamar al módulo objeto de la prueba y examinar los resultados de la prueba.
- Debido a todo esto, a menudo se combinan ambos tipos de prueba: unitarias y de integración.
- Por ejemplo a través de la prueba llamada “desarrollo incremental”.

Mix: Pruebas Unitarias y de Integración

- **Desarrollo Incremental**
  - El tipo de prueba incremental consiste en agregar cada módulo o componente individual al conjunto de componentes existentes y el conjunto resultante se prueba.
  - Esto reduce la necesidad de crear módulos conductores, permitiendo además examinar más detalladamente las interfaces.
  - Cuando las pruebas unitarias y de integración se realizan separadamente es difícil identificar los componentes individuales o módulos que causan resultados incorrectos.
  - Lo más probable es que los problemas que surjan al incorporar un nuevo componente, surjan de este último o de las interfaces entre él y los otros componentes.

Pruebas Unitarias y de Integración

- Estrategias de integración.
  - Es importante determinar la secuencia en que se van a producir e integrar los componentes.
  - Estrategia de arriba a abajo (top-down).
    - El primer componente o módulo que se desarrolla y prueba es el que está arriba en la jerarquía.
    - Los módulos de nivel más bajo se sustituyen por módulos auxiliares que simulan el resto de la funcionalidad.
    - En este caso no son necesarios módulos conductores
    - Una ventaja de esta estrategia es que las interfaces entre los módulos son probadas en una fase temprana y con frecuencia.

Pruebas Unitarias y de Integración

- Estrategias de integración (II)
  - Estrategia de abajo a arriba (bottom-up)
    - En este caso se crean primero los componentes de más bajo nivel y se crean módulos conductores para simular las invocaciones.
    - Los módulos auxiliares son necesarios en raras ocasiones.
    - Este tipo de enfoque de las pruebas permite un desarrollo más paralelizado que el enfoque el Top-Down, pero presenta mayores dificultades a la hora de planificar y de gestionar las pruebas.

Pruebas Unitarias y de Integración

- Estrategias de integración (III)
  - Estrategias combinadas
    - A menudo es útil aplicar las estrategias anteriores conjuntamente. Así, se desarrollarán partes del sistema de forma "top-down", mientras que los componentes más críticos (de nivel más bajo) se desarrollarán "bottom-up".
    - En este caso es necesaria una planificación cuidadosa y coordinada de modo que los componentes individuales se "encuentren" en el centro.

Pruebas de Sistema

- Propósito: Asegurar que se cumplen los requisitos de software.
- Este es un proceso formal:
  - Planificado en el Plan de Pruebas de Sistema.
  - La base son los *casos de prueba* definidos en la Especificación de Pruebas de Sistema.
  - Resultados se registran en el Acta de Pruebas de Sistema.

Pruebas de Sistema

- Su alcance es el sistema completo. Las pruebas típicas que deben hacerse son:
  - *Pruebas funcionales*
    - Consisten en determinar que se han cumplido los requisitos de software del sistema, y que éste realiza correctamente todas las funciones especificadas en el SRD.
  - *Pruebas de comunicaciones*
    - Determinan si las interfaces entre los componentes del sistema funcionan adecuadamente. Esto se refiere tanto a: las comunicaciones entre dispositivos remotos, las interfaces entre diferentes módulos, interfaces hombre-máquina, etc.

Pruebas de Sistema

- *Pruebas de rendimiento*
  - Consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- *Pruebas de volumen*
  - Consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando la carga de trabajo esperada.
- *Pruebas de sobrecarga*
  - Consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndolo a cargas excesivas.
- *Pruebas de disponibilidad de datos*
  - Consisten en demostrar que el sistema puede recuperarse ante fallas, sin pérdidas indebidas de datos. Esto considera el hardware y el software.

Pruebas de Sistema

- *Pruebas de facilidad de uso*
  - Consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurarse que se acomoda al modo habitual de trabajo de éstos, como para determinar si les facilita su trabajo.
- *Pruebas de operación*
  - Consisten en comprobar los procedimientos de operación, incluyendo los procedimientos de inicio y fin de trabajo y la facilidad de operación.
- *Pruebas de entorno*
  - Consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- *Pruebas de seguridad*
  - Consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.

Prueba de Usuario/Aceptación

- Propósito: Certificar que los requisitos del cliente se han cumplido satisfactoriamente y por lo tanto se puede iniciar la producción (o la marcha blanca).
- Esta etapa –también llamada pruebas de aceptación– debe estar a cargo de un equipo independiente del de desarrollo:
  - Puede ser otra gente de la empresa.
  - Puede ser el cliente, si hay uno específico, capazde hacerlo.
  - Hay empresas que ofrecen este servicio.

Pruebas de Sistema y de Aceptación

- Pruebas de Aceptación
  - Son aquellas pruebas que realiza el usuario/cliente con el objeto de comprobar si el sistema es aceptable para él.
  - Estas pruebas son del mismo tipo que las mencionadas anteriormente, pero son determinadas por el usuario, en lugar de serlo por el equipo de desarrollo.
  - Un tipo especial de estas pruebas es el de la ejecución en paralelo con el viejo sistema (legado), para comparar los resultados producidos por ambas ejecuciones.
  - Se chequea fundamentalmente: correctitud, robustez, performance y documentación.

Pruebas de regresión

- **Propósito:** Asegurar que cambios en el software no introducen nuevos defectos.
- Se ejecutan luego de modificar el sistema.
- Consisten en *repetir las pruebas* que ya se han hecho al sistema en una versión anterior, y comparar el resultado actual con el anterior.
- Aquí hay un beneficio enorme al tener un ambiente automatizado de pruebas.

Finalización de las Pruebas

- Es difícil asegurar que el último error detectado, era el único que quedaba. Sin embargo, existen métodos para mostrar cuándo está próximo el final:
  - Terminar la prueba cuando el tiempo establecido para la misma ha expirado.
  - Terminar la prueba cuando todos los casos de prueba diseñados, se ejecutan sin detectar errores.
- Otros métodos más complicados de aplicar pero más efectivos son:
  - Estimación del número total de errores del programa (en base a estadísticas de proyectos anteriores).
  - Estimación del porcentaje de estos errores que pueden encontrarse fácilmente.
  - Estimación de qué fracción de errores se origina en ciertos procesos (áreas conflictivas).

Criterio de Finalización

- Test de unidad: aplicar un método .
- Test de Integración: se finaliza cuando se cumplieron los XXX meses programados y se han hallado y corregido XXX errores.
- Test de Sistema: se finaliza cuando se cumplieron los YYY meses programados y se han hallado y YYY errores.
- Un criterio adicional lo puede proveer la **cobertura de las pruebas**, o sea la fracción de las líneas ejecutadas en las pruebas.
- En los dos últimos casos es importante llegar a comprobar que seguir testeando es improductivo.
- Test de Aceptación: generalmente involucra algunos meses (2-6) dependiendo del nivel de criticidad del sistema.

Conclusiones