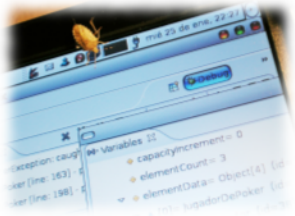
Tema 7: Introducción a la verificación y validación del Software

BLOQUE III:
VERIFICACIÓN, VALIDACIÓN Y PRUEBAS DE LOS SISTEMAS SOFTWARE.

Segundo curso de Grado en Ingeniería Informática
Curso 2014-2015

Javier Sánchez Monedero
jsanchezm@uco.es
http://www.uco.es/users/i02samoj

Índice



Índice

1. Índice 2
2. Introducción 2
3. Técnicas de V & V 4
4. Conclusiones 9
5. Bibliografía 9

1.

2. Introducción

Introducción

Advertencia

Advertencia
Estos apuntes no contienen todo el material necesario respecto al tema. Intentan dar una visión de conjunto y proporcionar los conocimientos básicos para que podáis consultar la bibliografía y manuales de referencia, así como material complementario colgado en Moodle.

Objetivos

- Conocer las actividades de verificación y validación.
- Conocer de la importancia de estas actividades.
- Conectar estas actividades de verificación y validación con las fases del desarrollo software en diferentes metodologías.

Verificación y validación

"Durante y después del proceso de implementación, el programa que se está desarrollando debe ser comprobado para asegurar que satisfacer su especificación y entrega la funcionalidad esperada por las partes de interés. La verificación y validación (V & V) es el nombre dado a estos procesos de análisis y pruebas." Sommerville [2005].

Existen actividades de V & V en cada etapa del proceso software.

Verificación

¿Estamos construyendo el producto correctamente? Se comprueba que el software cumple los requisitos funcionales y no funcionales de su especificación.

Validación

¿Estamos construyendo el producto correcto? Comprueba que el software cumple las expectativas que el cliente espera.

Objetivos de la V & V

Objetivos:

- Valorar y mejorar la **calidad** de los productos del trabajo generados durante el desarrollo y modificación del software.
- Debemos corregir todo posible fallo y alcanzar **cierto grado de perfección**.

- Debemos garantizar la consistencia, confiabilidad, utilidad, eficacia y el apego a los estándares del desarrollo del software.
- Para ello es necesario **encontrar defectos** en el sistema que estamos desarrollando y asegurarnos que el sistema será útil para el entorno de trabajo requerido.
- El **nivel de confianza** depende del propósito del sistema, las expectativas del usuario y el entorno de mercado.

Ejemplos de catástrofes causadas por software

Figura 1: Cohete explotando tras ser lanzado.

Cohete Mariner 1

- **Descripción**: investigación espacial destinada a Venus, se desvió de su trayectoria de vuelo poco después de su lanzamiento. El control de la misión destruyó el cohete pasados 293 segundos desde el despegue.
- **Causa**: Un programador codificó incorrectamente en el software una fórmula manuscrita, saltándose un simple guión sobre una expresión. Sin la función de suavizado indicada por este símbolo, el software interpretó como serias las variaciones normales de velocidad y causó correcciones erróneas en el rumbo que hicieron que el cohete saliera de su trayectoria

Hartford Coliseum (1978)

- **Coste**: 70 millones de dólares
- **Descripción**: Sólo unas horas después de que miles de aficionados al hockey abandonaran el Hartford Coliseum, la estructura de acero de su techo se desplomaba debido al peso de la nieve

- **Causa**: El desarrollador del software de diseño asistido (CAD) utilizado para diseñar el coliseo asumió incorrectamente que los soportes de acero del techo sólo debían aguantar la compresión de la propia estructura. Sin embargo, cuando uno de estos soportes se dobló debido al peso de la nieve, inició una reacción en cadena que hizo caer a las demás secciones del techo como si se tratara de piezas de dominó

Ariane 5

- **Descripción**: El cohete se desvía de su curso debido a un error en los datos procesados. El intento posterior de enderezar la trayectoria fue demasiado brusco y el cohete finalmente se destruyó a los 37 segundos
- **Causa**: el software de control realizó una conversión de un valor flotante de 64 bits en un entero de 16 bits sin comprobar que se produjeran desbordamientos

3. Técnicas de V & V

Técnicas de V & V

Técnicas de V & V

Inspecciones del Software:

- Se analizan las diferentes representaciones del sistema (diagramas de requisitos, diagramas de diseño y código fuente) en búsqueda de defectos.
- Son técnicas de validación **estáticas**: No requieren que el código se ejecute
- Debe realizarse durante todo el ciclo de desarrollo.
- Sólo sirven para la verificación, pero no garantizan software operacional.

Pruebas del Software:

- Se contrasta dinámicamente la respuesta de **prototipos ejecutables** del sistema con el comportamiento operacional esperado.
- Técnicas de validación **dinámicas** => El sistema se ejecuta
- Requiere disponer de prototipos ejecutables, por lo que sólo pueden utilizarse en ciertas fases del proceso

Las inspecciones del software y las pruebas son actividades complementarias, si bien las pruebas siempre serán la principal técnica de verificación y validación. Algunas metodologías, como *eXtreme Programming*, presentan un modelo de **programación por parejas** (*pair programming*) que facilita la inspección de código durante la programación, mejorando la calidad del código inicial.

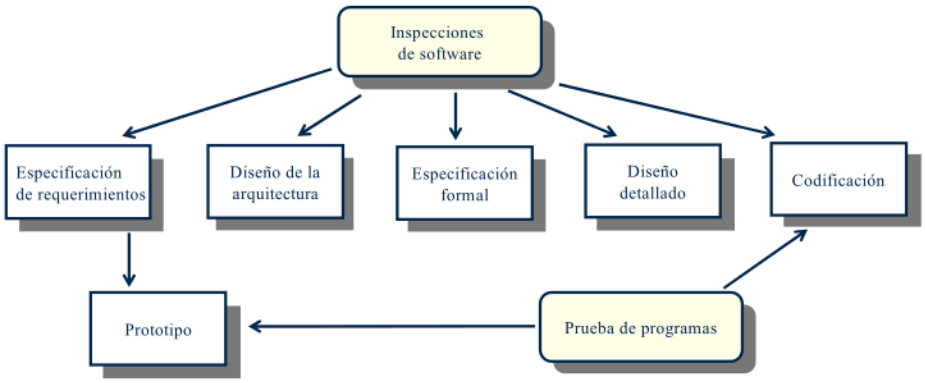


Figura 2: V & V estática y dinámica, fuente Sommerville [2005]

Esquema V & V estática y dinámica

Tipos de pruebas

Tipos de pruebas que pueden utilizarse en las distintas etapas:

- **Pruebas de validación**: intentan demostrar que el software es el que las partes interesadas quieren (satisface sus requisitos). Incluyen estadísticas de rendimiento y fiabilidad bajo condiciones concretas.
- **Pruebas de defectos**: intentan revelar defectos del sistema en lugar de simular su uso operacional. El objetivo es encontrar inconsistencias entre un programa y su especificación.

Aunque los objetivos sean distintos será común que unos tipos de pruebas revelen fallos correspondientes a otras.

La depuración

Proceso de **depuración**: Proceso que localiza y corrige los errores descubiertos durante la verificación y validación.

- Es un proceso complicado pues no siempre los errores se detectan cerca del punto en que se generaron.
- Se utilizan herramientas de depuración, que facilitan el proceso

Después de reparar el error, hay que volver a probar el sistema (pruebas de regresión). La solución del primer fallo puede dar lugar a nuevos fallos o por el contrario solucionar un conjunto de errores.

Proceso de depuración

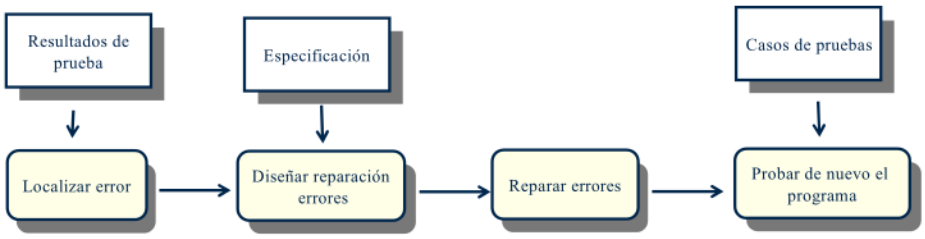


Figura 3: Esquema del proceso de depuración, fuente Sommerville [2005]

Inspecciones de software

Son un proceso de V & V estático en el que un sistema software se revisa para encontrar errores, omisiones o anomalías Sommerville [2005].

El *Software Engineering Institute* (SEI) señaló las inspecciones del software como una de las prácticas industriales esenciales para la gestión del proceso software. Las revisiones formales son una actividad en grupo para estudiar sistemáticamente los productos del desarrollo del software.

Las inspecciones:

- Generalmente se centran en el código fuente, pero puede inspeccionarse cualquier otro artefacto (por ejemplo documento de requisitos o diseño).
- Las inspecciones se pueden aplicar a la detección de fallos en cualquiera de los documentos generados.
- Algunos estudios indican que pueden detectar más del 60 % de los fallos a unos costes mucho más bajos que las pruebas dinámicas.
- Permiten detectar múltiples defectos en una simple inspección, mientras que las pruebas solo suelen detectar un fallo por prueba.
- Permite utilizar el conocimiento del dominio y del lenguaje, que determinan los principales tipos de fallos que se suelen cometer.
- Las inspecciones son útiles para detectar los fallos de módulos, pero no detectan fallos a nivel de sistema, que ha de hacerse con pruebas.
- La inspecciones no son útiles para la detección de niveles de fiabilidad y evaluación de fallos no funcionales.

Comprobaciones de inspección

Sommerville [2005] nos da un listado orientativo de comprobaciones en la inspección (orientadas principalmente al código):

- Defectos de datos:

- ¿se inicializan todas las variables antes de que se utilicen sus valores?
- ¿tienen nombre todas las constantes?
- ¿existe alguna posibilidad de que un búfer se desborde?

- Defectos de control:
  - Para cada sentencia condicional, ¿es correcta la condición?
  - ¿Se garantiza que termina cada bucle?
  - En las sentencias compuestas, ¿las llaves están puestas de forma correcta?
  - En estructuras *case*, ¿se controlan las alternativas?
- Defectos de entrada/salidad:
  - ¿Se utilizan todas las variables de entrada?
  - ¿Se asigna valor a todas las variables de salida?
  - ¿Pueden provocar corrupciones de datos las entradas no esperadas?
- Defectos de la interfaz:
  - ¿Las llamadas a funciones y a métodos tienen el número correctos de parámetros?
  - ¿Están en orden correcto los parámetros?
- Defectos de gestión de almacenamiento:
  - Si una estructura enlazada se modifica, ¿se reasignan correctamente todos los enlaces?
  - ¿Se realiza correctamente el almacenamiento dinámico?
  - ¿Se libera explícitamente el espacio de memoria?
- Defectos de manejo de excepciones:
  - ¿Se tienen en cuenta todas las condiciones de error posibles?

Reflexión sobre lenguajes de programación

En las transparencias anteriores se anima a hacer una serie de comprobaciones. ¿Qué herramientas nos dan algunos lenguajes de programación para evitar algunos de los errores?¿Qué lenguajes son menos propensos a algunos de los defectos enumerados? Qué nos puede ayudar (a veces a costa de la eficiencia):

- Usar `calloc` en vez de `malloc`.

- Utilizar macros para valores de inicialización o constantes.
- Utilizar iteradores.
- Configurar los compiladores para que muestren más advertencias (por ejemplo la opción `-Wall` en `gcc`)
- Indentación y limpieza en el código.
- ...
- En general un conocimiento avanzado de los lenguajes de programación y herramientas que automaticen el proceso de inspección (por ejemplo Valgrind para comprobar memoria no liberada en las aplicaciones).

Verificación y métodos formales

- Los **métodos formales** son una forma de expresar matemáticamente la especificación, desarrollo y verificación de sistemas hardware o software.
- Estos métodos se ocupan principalmente:
  - Del Análisis matemático de la especificación
  - De transformar la especificación a una representación más detallada semánticamente equivalente
  - De verificar formalmente que una representación del sistema es semánticamente equivalente a otra representación.

Los **métodos formales** pueden utilizarse en diferentes etapas en el proceso de V & V:

1. Puede desarrollarse una especificación formal del sistema y analizarse matemáticamente para buscar inconsistencias.
2. Puede verificarse formalmente mediante argumentos matemáticos que el código de un sistema software es consistente con su especificación.

A favor
La verificación formal demuestra que el programa desarrollado satisface su especificación.

En contra

- El uso de la especificación formal requiere notaciones especializadas y complejas de aprender, y no pueden ser comprendidas por expertos del dominio. Así, la distancia entre ingenieros del software y expertos del dominio puede aumentar.
- Aunque la especificación puede ser matemáticamente consistente, puede no especificar las propiedades del sistema que son realmente necesarias.

4. Conclusiones

Conclusiones

Conclusiones

Conexión con temas anteriores:

- ¿Qué sistemas de captura y documentación de requisitos obliga a establecer criterios de aceptación del cliente?
- Los criterios de aceptación en las historias de usuario, ¿se refieren a criterios de verificación o de validación?
- Si una empresa o equipo tiene unos parámetros de calidad que comprobar en el software, ¿qué tipo de pruebas se deberían hacer?

5. Bibliografía

Fuentes

El contenido de estos apuntes se basa a su vez en otros apuntes [Irene T., 2013], y en las referencias citadas, fundamentalmente el Capítulo 22 del libro de Ingeniería del Software de Ian Sommerville [Sommerville, 2005].

Bibliografía y enlaces

Referencias

L. R. Irene T. Apuntes de la Asignatura Ingeniería del Software. Tema 8: Introducción a la Verificación, Validación del Software. 2013.

I. Sommerville. *Ingeniería del Software*. Pearson Education. Addison-Wesley, 2005.