UNIDAD 4.
DISEÑO Y REALIZACIÓN DE PRUEBAS

VICENT MARTÍ

OBJETIVOS

- Lo que se busca en esta unidad es que conozcas la importancia que tiene para un software la fase de pruebas, que, lejos de ser un mero trámite, ha de tenerse muy en cuenta.
- Es importante que comprendas los conceptos de procedimientos y casos de pruebas, así como los tipos de pruebas de un software (regresión, funcionales, estructurales,etc.).
- También es fundamental para un desarrollador conocer la diferencia entre una prueba de caja blanca y otra de caja negra, así como conocer herramientas de depuración de código.
- Por último, y para tener una visión más completa de un desarrollo software, tendrás que aprender los conceptos de planificación de pruebas y calidad del software.

MAPA CONCEPTUAL

CONTENIDO

1. Introducción.
2. Procedimientos de pruebas y casos de prueba.
3. Tipos de pruebas: funcionales, estructurales y regresión.
4. Pruebas de caja blanca.
5. Pruebas de caja negra.
6. Herramientas de depuración.
7. Planificación de pruebas.
8. Calidad del software.

1. INTRODUCCIÓN

Siendo realistas, es prácticamente imposible realizar pruebas exhaustivas a un programa. Las pruebas son generalmente demasiado costosas. Salvo que el programa sea tan importante como para realizarlas, lo que se hace es llegar a un punto intermedio en el cual se garantiza que no va a haber defectos importantes o muchos defectos y la aplicación está completamente operativa con un funcionamiento aceptable.

El objetivo de las pruebas es convencer tanto a los usuarios como a los propios desarrolladores, de que el software es lo suficientemente robusto como para poder trabajar con él de forma productiva.

Cuando un software supera unas pruebas exhaustivas, las probabilidades de que ese software de problemas en producción se atenúan y, por tanto, su fiabilidad aumenta.

2. PROCEDIMIENTOS DE PRUEBAS Y CASOS DE PRUEBA I

Procedimiento de prueba

Es la definición del objetivo que desea conseguirse con las pruebas, qué es lo que va a probarse y cómo.

Objetivo de las pruebas

No siempre es detectar errores. Muchas veces lo que quiere conseguirse es que el sistema ofrezca un rendimiento determinado, que la interfaz tenga una apariencia y cumpla unas características determinadas, etc.

Por lo tanto, la ausencia de errores en las pruebas nunca significa que el software las supere, pues hay muchos parámetros en juego.

Cuando se diseñan los procedimientos, se deciden las personas que hacen las pruebas y bajo qué parámetros van a realizarse.

Siempre tiene que haber personal externo al equipo de desarrollo para hacer las pruebas, pues los propios programadores solo probarán lo que funciona (si supieran dónde están los errores, los corregirían).

2. PROCEDIMIENTOS DE PRUEBAS Y CASOS DE PRUEBA II

En los planes de pruebas generalmente se cubren los siguientes aspectos:

1. Introducción. Breve introducción del sistemas.
2. Módulos o partes del software a probar. Detallar cada objeto que se va a probar.
3. Características del software a probar.
4. Características del software que no se prueba.
5. Enfoque de las pruebas. Donde se detallan las personas responsables, la planificación, entre otros.
6. Criterios de validez o invalidez del software. Se especifican los criterios para dar por válido o no un software.
7. Proceso de pruebas. Especificación del proceso y procedimientos de pruebas.
8. Requerimientos del entorno.
9. Homologación o aprobación del plan. Debe estar firmado por los interesados y responsables.

CASOS DE PRUEBA

En la fase de pruebas, se diseñan y preparan los casos de prueba, que se crean con el objeto de encontrar fallos.
Si se prueba un software y funciona, la mayoría de las veces no hace falta probar lo mismo. Hay que crear otro tipo de pruebas, no repetirlas.

Hay que considerar que la prueba no sea demasiado sencilla, no va a aportar nada. Lo más recomendable es simular situaciones reales para ver como responde el software en su uso habitual. También deberemos poner a prueba el software, va a ser más difícil de ver el origen de los errores si los hay, pero es necesario hacerlo.

CODIFICACIÓN Y EJECUCIÓN DE PRUEBAS

Una vez diseñados los casos de prueba, hay que generar las condiciones necesarias para poder ejecutar dichos casos de prueba.

Habrá que codificarlos en muchos casos generando set o conjuntos de datos. En estos set de datos, hay que incluir tanto datos válidos como inválidos como algunos datos fuera de rango o disparados.

También habrá que preparar las máquinas sobre las que van a hacerse las pruebas instalando el software necesario, los usuarios de sistema, realizar carga del sistema, etc.

3. TIPOS DE PRUEBAS: FUNCIONALES, ESTRUCTURALES Y REGRESIÓN.

Vamos a ver los tipos de pruebas más frecuentes, pero existen muchas categorías.

- Pruebas funcionales. Buscan que los componentes software diseñados cumplan con la función correspondiente. **El tester** (ingeniero de pruebas) para la realización de las pruebas, se basa en la documentación existente. También suelen realizarse pruebas en conjunto con los usuarios, puesto que ellos saben cómo tiene que funcionar el sistema. No se evalúa cómo el sistema funciona internamente, pero sí qué es lo que hace. Y se conocen como **pruebas de caja negra.**
- Pruebas de seguridad. Dónde se evalúan aspectos de seguridad.
- Pruebas no funcionales. Aquellas más técnicas que se realizan al sistemas. Estas siguen siendo de caja negra, pues nunca se examina la lógica interna. (de carga, estrés, rendimiento, fiabilidad, etc.)
- Pruebas estructurales. Son de caja blanca ya que en algún momento se utilizan técnicas de análisis del código. Generalmente con herramientas especializadas.
- Pruebas de regresión. No suele probarse lo que ya se ha probado, pero si el software se modifica se realizan este tipo de pruebas.

4. PRUEBAS DE CAJA BLANCA

Son el tipo de pruebas que tiene en cuenta el código que quiere probarse. También conocidas como *clear box testing*. El objetivo será probar el código de cada uno de los elementos que forman el *software* total. Encontraremos algunas clases dentro de este tipo de pruebas:

Pruebas de cubrimiento: En este tipo de pruebas, el objeto es ejecutar, al menos una vez, todas las sentencias o líneas del programa. Puede que no siempre se pueda comprobar el 100% del código porque haya alguna condición que nunca se cumpla o también puede haber excepciones o notificaciones de error en un código que nunca va a fallar.

Pruebas de condiciones: Aquí necesitaremos varios casos de prueba. En una condición puede haber varias condiciones simples y habrá que generar un caso de pruebas por cada operando lógico o comparación. La idea es que, en cada expresión, se cumple en un caso y en otro no.

Pruebas de bucles: Los bucles son estructuras que se basan en la repetición, por lo tanto, la pruebas de bucles se basará en la repetición de un número especial de veces.

5. PRUEBAS DE CAJA NEGRA I

En el caso de las pruebas de caja negra (pruebas interfaz) también tendremos en cuenta las siguientes:
Pruebas de clases de equivalencia de datos: El objetivo de esta prueba es comprobar todas las clases válidas y las inválidas al menos una vez. Cada vez que se diseña un caso de prueba con datos inválidos, se introducirá solamente una clase inválida. De esa manera, se conocerá si el programa está funcionando correctamente. Muchas veces, al utilizar varias clases inválidas, los errores se enmascaran y no puede conocerse si todas las clases funcionan.
Pruebas de valores límite: Complementarias a las pruebas de particiones, cuyo objetivo es generar valores que puedan probar si la interfaz y el programa funcionan correctamente. Por ejemplo, para testear en la web de un banco que el máximo dinero que se puede transferir en una operación es X y los programadores no se han equivocado con los símbolos

5. PRUEBAS DE CAJA NEGRA II

Prueba de interfaces: Una interfaz de usuario, generalmente, se testea con una técnica que se denomina prueba de interfaces.

Dependiendo de las entradas, la interfaz proporcionará una salida determinada. Esa salida debería ser la esperada. Muchas veces cuando se testea un programa hay que conocer su funcionalidad.

A. *Cómo testear una interfaz*

Una primera prueba puede consistir en seguir el manual de usuario. Si el software pasa esta prueba, entonces podrá pasar a sufrir un testeo más completo.

B. *Testear la usabilidad*

Tiene por objeto evaluar si el producto generado va a resultar lo esperado por el usuario. Hay que ver y trabajar con la interfaz desde el punto de vista del usuario. Además, en su testeo, deberían utilizarse datos reales.

5. PRUEBAS DE CAJA NEGRA III

C. *Testear la accesibilidad*

Mucha gente no sabe que la accesibilidad no solamente es que el software esté diseñado para usuarios con discapacidad, sino que también sea accesible por *frameworks* de test automatizados.

Algunos software son testeados por expertos en accesibilidad que realizan auditorías de este, de tal manera que determinan si ese software supera los requerimientos exigibles.

PREGUNTA (RESPONDE Y SUBE A AULES)

¿Qué es una auditoría informática y cuál es el perfil de un auditor informático?

EJERCICIO

Realiza: P01-Preguntas presentación

- Contesta las preguntas que van saliendo en la presentación.
- Súbelo al aula virtual.

6. HERRAMIENTAS DE DEPURACIÓN I

Cada IDE (Integrated Development Environment) incluye herramientas de depuración como: inclusión de puntos de ruptura, ejecución paso a paso de cada instrucción, ejecución por procedimiento, inspección de variables, etc.

Durante el proceso de desarrollo de software, se pueden producir dos tipos de errores: **errores de compilación o errores lógicos.**

- Cuando ocurre un error de compilación, el entorno nos proporciona información de donde se produce y como poder solucionarlo. El programa no puede compilarse hasta que el programador o programadora no corrija ese error.
- En los errores son lógicos, comúnmente llamados bugs, estos no evitan que el programa se pueda compilar con éxito, pero pueden provocar que el programa devuelva resultados erróneos, que no sean los esperados o pueden provocar que el programa termine antes de tiempo o no termine nunca.

Para solucionar este tipo de problemas, los entornos de desarrollo incorporan una herramienta conocida como **depurador.**

6. HERRAMIENTAS DE DEPURACIÓN II

El **depurador** permite supervisar la ejecución de los programas, para localizar y eliminar los errores lógicos. Un programa debe compilarse con éxito para poder utilizarlo en el depurador. El depurador nos permita analizar todo el programa, mientras éste se ejecuta. Permite suspender la ejecución de un programa, examinar y establecer los valores de las variables, comprobar los valores devueltos por un determinado método, el resultado de una comparación lógica o relacional, etc.

EJERCICIO

Realiza la práctica: P02-Debugger

Solo debes entregar:

Guarda una captura de pantalla donde se vea toda la perspectiva de Debug con el programa LlenarNumeros y el breakpoint con la condición i = 3 como se indica en el texto. Guárdala con el nombre DebugEclipse.jpg y súbela al aula virtual.

7. PLANIFICACIÓN DE PRUEBAS I

Mediante la realización de pruebas se software, se van a realizar las tareas de verificación (*Proceso por el que se comprueba que el software cumple los requisitos especificados*) y validación (*Proceso que comprueba si el software hace lo que el usuario deseaba. Tiene que estar verificado*) del software.

Para llevar a cabo el proceso de pruebas, de manera eficiente, es necesario implementar una estrategia de pruebas. Siguiendo el **Modelo en Espiral**, las pruebas empezarían con la **prueba de unidad**, donde se analizaría el código implementado y seguiríamos en la **prueba de integración**, donde se prestan atención al diseño y la construcción de la arquitectura del software. El siguiente paso sería la **prueba de validación**, donde se comprueba que el sistema construido cumple con lo establecido en el análisis de requisitos de software y se alcanza la prueba de sistema que verifica el funcionamiento total del software y otros elementos del sistema. También se establecerá **automatizar pruebas** o repetirlas después de correcciones y en la fase de mantenimiento del software.

7. PLANIFICACIÓN DE PRUEBAS II

La planificación de las pruebas es un punto importante en la toma de decisiones de un proyecto. Qué tipo de pruebas y cuándo van a realizarse son preguntas que hay que tener en cuenta desde el principio.
Vamos a ver las pruebas y el momento en el que se realizan.

1. Pruebas unitarias

En la primera fase de diseño y desarrollo. No hay que demorar mucho la realización de estas pruebas ya que luego hay que integrar todo el software y los fallos van acumulándose y la localización y diagnóstico se complican. Con las pruebas unitarias se debe probar todas las funciones o métodos no triviales de forma que cada caso de prueba sea independiente del resto.
En el diseño de los casos de pruebas unitarias, habrá que tener en cuenta los siguientes requisitos:

- **Automatizable**: no debería requerirse una intervención manual.
- **Completas**: deben cubrir la mayor cantidad de código.
- **Repetibles o Reutilizables**: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez.
- **Independientes**: la ejecución de una prueba no debe afectar a la ejecución de otra.
- **Profesionales**: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

7. PLANIFICACIÓN DE PRUEBAS III

1. Pruebas unitarias (continuación)

El objetivo de las pruebas unitarias es aislar cada parte del programa y demostrar que las partes individuales son correctas. Las pruebas individuales nos proporcionan cinco ventajas básicas:

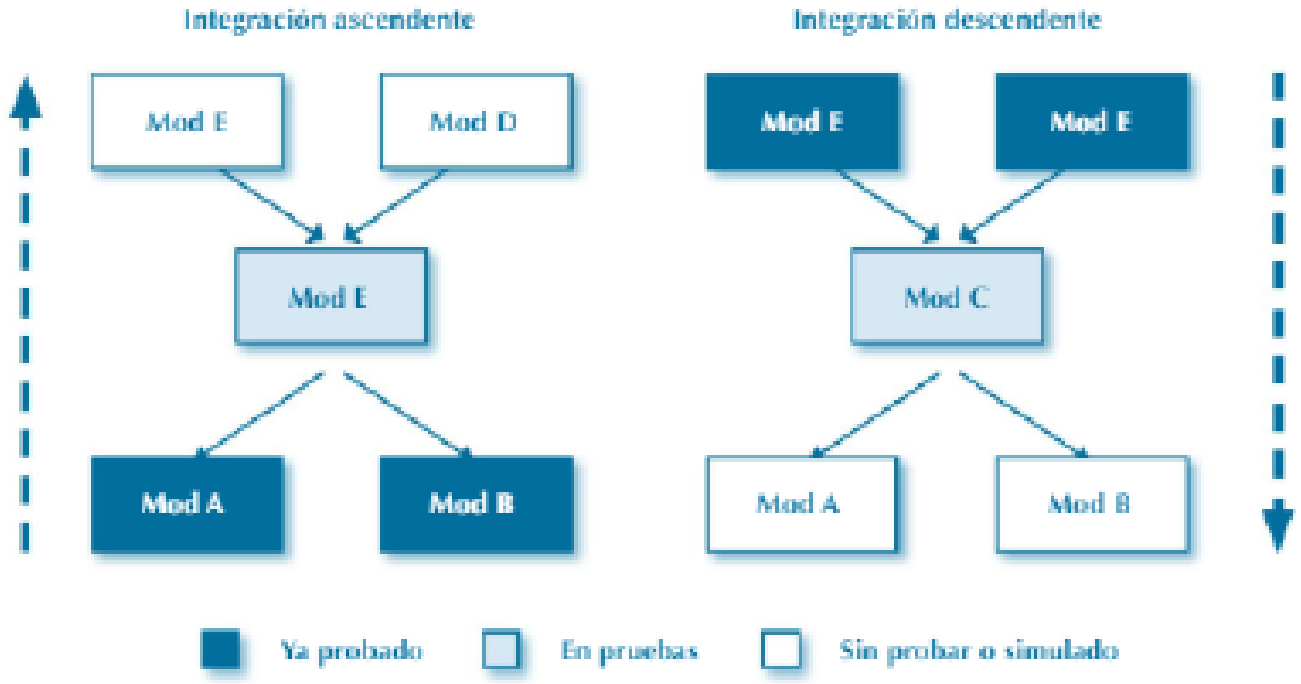
- **Fomentan el cambio**: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- **Simplifica la integración**: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.
- **Documenta el código**: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- **Separación de la interfaz y la implementación**: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- Los **errores están más acotados** y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos.

7. PLANIFICACIÓN DE PRUEBAS IV

2. Pruebas de integración

Una vez que los componentes individuales se han probado, es momento de ir integrando módulos. Existen pruebas de integración ascendentes y descendentes.

Pueden probarse los módulos más generales y luego ir a los más específicos o al contrario.



7. PLANIFICACIÓN DE PRUEBAS V

3. Pruebas de aceptación o validación

Este tipo de pruebas tratan de probar el sistemas completo. Además de probar que los requisitos del programa se cumplen uno por uno, el equipo de pruebas mirará también si técnicamente el programa es estable o no tiene ningún fallo.

Existen en este estado pruebas alfa, las cuales se realizan en un entorno controlado y bajo unas especificaciones concretas, y pruebas beta, en las que los usuarios prueban el sistema en un entorno no controlado por los desarrolladores.

4. Automatización de pruebas

Muchas veces, es necesario automatizar pruebas o repetir las mismas pruebas tras realizar mantenimientos, modificaciones o correcciones.

Es siempre bueno conservar los datos, set de pruebas, programas y módulos de prueba, puesto que no se sabe si van a ser necesarios en un futuro.

HERRAMIENTAS PARA PRUEBAS JAVA I

Entre las herramientas que nos podemos encontrar en el mercado, para poder realizar las pruebas, las más destacadas serían: **Jtiger, TestNG y Junit**

Jtiger:

- Framework de pruebas unitarias para Java
- Es de código abierto.
- Capacidad para exportar informes en HTML, XML o texto plano.
- Es posible ejecutar casos de prueba de Junit mediante un plugin.
- Posee una completa variedad de aserciones como la comprobación de cumplimiento del contrato en un método.
- Los metadatos (*Conjunto de datos que se utilizan para describir otros datos*) de los casos de prueba son especificados como anotaciones del lenguaje Java
- Incluye una tarea de Ant para automatizar las pruebas.
- Documentación muy completa en JavaDoc, y una página web con toda la información necesaria para comprender su uso, y utilizarlo con IDE como Eclipse.
- El Framework incluye pruebas unitarias sobre sí mismo.

HERRAMIENTAS PARA PRUEBAS JAVA II

Entre las herramientas que nos podemos encontrar en el mercado, para poder realizar las pruebas, las más destacadas serían:

TestNG:

- Esta inspirado en JUnit y NUnit.
- Está diseñado para cubrir todo tipo de pruebas, no solo las unitarias, sino también las funcionales, las de integración ...
- Utiliza las anotaciones de Java 1.5 (desde mucho antes que Junit).
- Es compatible con pruebas de Junit.
- Soporte para el paso de parámetros a los métodos de pruebas.
- Permite la distribución de pruebas en maquinas esclavas.
- Soportado por gran variedad de plug-ins (Eclipse, NetBeans, IDEA ...)
- Los clases de pruebas no necesitan implementar ninguna interfaz ni extender ninguna otra clase.
- Una vez compiladas la pruebas, estas se pueden invocar desde la linea de comandos con una tarea de Ant o con un fichero XML.
- Los métodos de prueba se organizan en grupos (un método puede pertenecer a uno o varios grupos).

HERRAMIENTAS PARA PRUEBAS JAVA III

Entre las herramientas que nos podemos encontrar en el mercado, para poder realizar las pruebas, las más destacadas serían:

Junit:

- Framework de pruebas unitarias creado por Erich Gamma y Kent Beck.
- Es una herramienta de código abierto.
- Multitud de documentación y ejemplos en la web.
- Se ha convertido en el estándar de hecho para las pruebas unitarias en Java.
- Soportado por la mayoría de los IDE como eclipse o Netbeans.
- Es una implementación de la arquitectura xUnit para los frameworks de pruebas unitarias.
- Posee una comunidad mucho mayor que el resto de los frameworks de pruebas en Java.
- Soporta múltiples tipos de aserciones.
- Desde la versión 4 utiliza las anotaciones del JDK 1.5 de Java.
- Posibilidad de crear informes en HTML.
- Organización de las pruebas en Suites de pruebas.
- Es la herramienta de pruebas más extendida para el lenguaje Java.
- Los entornos de desarrollo para Java, NetBeans y Eclipse, incorporan un plugin para Junit.

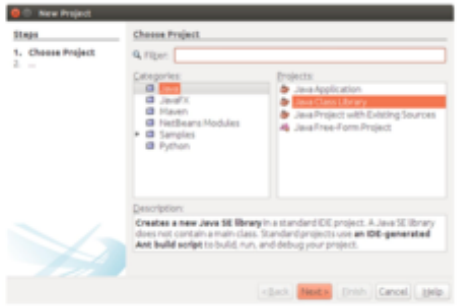
HERRAMIENTAS PARA PRUEBAS JAVA IV

Va a tomarse como ejemplo Junit, que permite realizar test repetibles, Disponemos en el aula virtual **JUnit – Presentación** que nos permite conocer este *framework.*

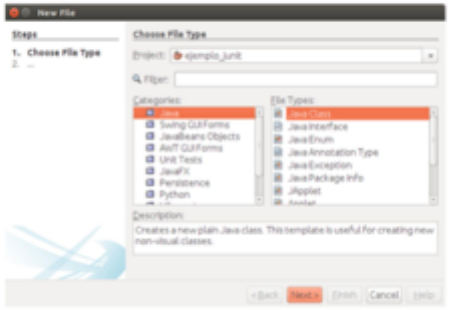
Podemos practicar realizando los ejercicios guiados de la presentación (estos no deben entregarse)

Junit. Creando una primera clase de test

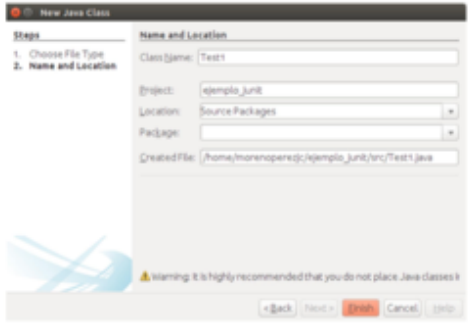
Crear un nuevo proyecto desde File -> New Project.



Elegir en la categoría Java "Java Class Library"



Crear una clase Test1.



Para saber más:

En el siguiente enlace nos encontramos con un ejempo completo de prueba de la unidad con NetBeans. Creación de Casos de Prueba en NetBeans con Junit http://netbeans.org/kb/docs/java/junit-intro.html#Exercise_21

HERRAMIENTAS PARA OTROS LENGUAJES.

En la actualidad, nos encontramos con un amplio conjunto de herramientas destinadas a la automatización del prueba, para la mayoría de los lenguajes de programación más extendidos en la actualidad. Existen herramientas para C++, PHP, etc.
Cabe destacar las siguientes herramientas:

- **CppUnit**: Framework de pruebas unitarias para el lenguaje C++. Basado en el diseño de xUnit.
- **Nunit**: Framework de pruebas unitarias para la plataforma .NET. Basado en el diseño de xUnit.
- **SimpleTest**: Entorno de pruebas para aplicaciones realizadas en PHP.
- **PHPUnit**: framework para realizar pruebas unitarias en PHP.
- **FoxUnit**: framework OpenSource de pruebas unitarias para Microsoft Visual FoxPro
- **MOQ**: Framework para la creación dinámica de objetos simuladores (mocks).

EJERCICIO

Realiza las prácticas: P03 Y P04 de JUnit

- Documenta todo el proceso de las prácticas con portada, índice, bibliografía.
- Realiza los pantallazos de los diferentes pasos más importantes. Explicándolos.
- Súbelo al aula virtual.

8. CALIDAD DEL SOFTWARE I

La calidad es un tema que, desde hace años, tiene una importancia en el mundo de la comercialización de productos. El mercado actual es muy competitivo y la calidad es uno de los aspectos diferenciales.
En los años noventa, se vivió una crisis del software. Fueron en esos años en los que la calidad y el proceso de desarrollo no se le daba importancia, lo que llevo a la falta de profesionalidad en muchos casos. Otro gran problema que genera engaños y precariedad son las '**cárnicas del software**'. También otro momento crítico para el software fue el efecto 2000.

PREGUNTA (RESPONDE Y SUBE A AULES)

- Elabora una lista con las razones por las que se originó la crisis del software. ¿Cómo piensas que podría haberse solucionado este problema?
- ¿Qué son las cárnicas del software y por qué han contribuido a que la calidad de los programas informáticos haya descendido?
- ¿En qué consistió el efecto 2000? ¿Tuvo mucha repercusión en las empresas?

8. CALIDAD DEL SOFTWARE II

Para evaluar el software, es necesario contar con criterios adecuados que permitan analizar el software desde diferentes puntos de vista.

Las pruebas de carga se realizan sobre el sistema simulando una serie de peticiones esperadas o un número de usuarios esperado trabajando de forma concurrente, realizando un número de transacciones determinado. En estas pruebas, se evalúan los tiempos de respuesta de las transacciones. Generalmente, se realizan varios tipos de carga (baja, media y alta) para evaluar el impacto y poder graficar el rendimiento del sistema.

La prueba de estrés se utiliza normalmente para romper la aplicación. Se va doblando el número de usuarios que se agregan a la aplicación y se ejecuta una prueba de carga hasta que se rompe. Este tipo de prueba se realiza para determinar la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada.

Prueba de estabilidad se hace para determinar si la aplicación puede aguantar una carga esperada continuada. Generalmente esta prueba se realiza para determinar si hay alguna fuga de memoria en la aplicación.

8. CALIDAD DEL SOFTWARE III

Las métricas de calidad sirven para medir los criterios de calidad de un software. Algunas métricas son:

- Tolerancia a errores. Efectos de los errores.
- Facilidad de expansión. Nuevas funcionalidades.
- Independencia de plataforma del hardware.
- Modularidad. Número de componentes independientes de un programa.
- Estandarización de los datos.

La norma ISO/IEC 29119 de prueba de software, pretende unificar en una única norma, todos los estándares, de forma que proporcione vocabulario, procesos, documentación y técnicas para cubrir todo el ciclo de vida del software. Desde estrategias de prueba para la organización y políticas de prueba, prueba de proyecto al análisis de casos de prueba, diseño, ejecución e informe. Con este estándar, se podrá realizar cualquier prueba para cualquier proyecto de desarrollo o mantenimiento de software.

NORMA ISO/IEC 29119

La norma ISO/IEC 29119 la compone las siguientes partes:

Parte 1. Conceptos y vocabulario:

- Introducción a la prueba.
- Pruebas basadas en riesgo.
- Fases de prueba (unidad, integración, sistema, validación) y tipos de prueba (estática, dinámica, no funcional, …).
- Prueba en diferentes ciclos de vida del software.
- Roles y responsabilidades en la prueba.
- Métricas y medidas.

Parte 2. Procesos de prueba:

- Política de la organización.
- Gestión del proyecto de prueba.
- Procesos de prueba estática.
- Procesos de prueba dinámica.

Parte 3. Documentación

- Contenido.
- Plantilla.

Parte 4. Técnicas de prueba:

- Descripción y ejemplos.
- Estáticas: revisiones, inspecciones, etc.
- Dinámicas: Caja negra, caja blanca, Técnicas de prueba no funcional (Seguridad, rendimiento, usabilidad, etc) .

Para saber más:
En el siguiente enlace podrás visitar la página internacional, donde se detallan las normas a seguir, por las pruebas de software:
Normas para la prueba de software
http://softwaretestingstandard.org/

DOCUMENTACIÓN DE LA PRUEBA I

Como en otras etapas y tareas del desarrollo de aplicaciones, la documentación de las pruebas es un requisito indispensable para su correcta realización. Unas pruebas bien documentadas podrán también servir como base de conocimiento para futuras tareas de comprobación.
Las metodologías actuales, como Métrica v.3 (*Metodología de Planificación, Desarrollo y Mantenimiento de sistemas de información. Métrica v3 se puede usar libremente, con la única restricción de citar la fuente de su propiedad intelectual, que es el Ministerio de Presidencia*), proponen que la documentación de la fase de pruebas se basen en los **estándares ANSI / IEEE** sobre verificación y validación de software.

Un documento de pruebas estándar puede facilitar la comunicación entre desarrolladores al suministrar un marco de referencia común. La definición de un documento estándar de prueba puede servir para comprobar que se ha desarrollado todo el proceso de prueba de software.

Para saber más:
En el siguiente enlace podrás visitar la página de Ministerio, dedicada a Métrica v.3
https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

DOCUMENTACIÓN DE LA PRUEBA II

Los documentos que se van a generar son:

- **Plan de Pruebas**: Al principio se desarrollará una planificación general, que quedará reflejada en el "Plan de Pruebas". El plan de pruebas se inicia el proceso de Análisis del Sistema.
- **Especificación del diseño de pruebas.** De la ampliación y detalle del plan de pruebas, surge el documento "Especificación del diseño de pruebas".
- **Especificación de un caso de prueba**. Los casos de prueba se concretan a partir de la especificación del diseño de pruebas.
- **Especificación de procedimiento de prueba**. Una vez especificado un caso de prueba, será preciso detallar el modo en que van a ser ejecutados cada uno de los casos de prueba, siendo recogido en el documento "Especificación del procedimiento de prueba".
- **Registro de pruebas**. En el "Registro de pruebas" se registrarán los sucesos que tengan lugar durante las pruebas.
- **Informe de incidente de pruebas**. Para cada incidente, defecto detectado, solicitud de mejora, etc, se elaborará un "informe de incidente de pruebas".
- **Informe sumario de pruebas**. Finalmente un "Informe sumario de pruebas" resumirá las actividades de prueba vinculadas a uno o más especificaciones de diseño de pruebas.