***Resumen Tema 8***

***Autor:*** *Jorge Muñoz García - 2ºDAM*

***Asignatura:*** *Desarrollo de Interfaces*

***Introducción***

Preparar una fase de desarrollo de software de manera incorrecta puede suponer que el proyecto contenga errores determinantes… Una aplicación no debe llegar al usuario final con fallos, sin mencionar que debe ser legible y sencilla para este. Esto provocaría una falta de profesionalidad y confianza de los desarrolladores entre los usuarios.

Por estos motivos, las pruebas en la fase de desarrollo software son básicas y necesarias antes de realizar la transferencia del proyecto a cualquier usuario…

Los perfiles habituales de un equipo de trabajo en un proyecto de desarrollo software son: Jefe de proyecto, analista, diseñador, programador y tester.

***Validación de la corrección en el desarrollo de una aplicación***

Un error no detectado en el inicio del desarrollo normalmente necesita de más esfuerzo para solucionarlo posteriormente. En función de la importancia del error se podrá solucionar modificando algo de código, en la peor situación habría que rehacer el proyecto.

Hay dos conceptos clave (verificación y validación), estos están vinculados con las pruebas que se realizan en un proyecto. Son el nombre que se da a los procesos que revisan el software y aseguran que se cumplen las expectativas y necesidades del cliente.

1. ***Validación:*** Actividad que asegura que la aplicación obtenida es la que se deseaba programar. Esto confirma que la aplicación funciona correctamente, de la manera que el cliente quería. ***“¿El sistema hace lo que quiere el cliente?”.***
2. ***Verificación:*** Actividad que comprueba el correcto funcionamiento del código. Comprueba que la aplicación se haya desarrollado correctamente. ***“¿El sistema hace lo que debe hacer?”.***

A lo largo del desarrollo del software el sistema debe ser verificado y validado en cada fase del desarrollo, aportando a la siguiente fase documentación para volver a realizar estas dos acciones. Estos dos pasos determinan la validación de la corrección del proyecto en cada fase del proceso.

***Objetivos, importancia y limitaciones del proceso de prueba / Estrategias***

El proceso de prueba se considera una fase independiente del desarrollo software. Se realiza al finalizar la codificación y antes de la transferencia al cliente.

Un módulo es una parte independiente del programa que se diseña, codifica y prueba por separado / trabajo, actividad o función bien definida y clara.

El proceso de prueba trata de ejecutar repetidamente una aplicación, usando todas las posibilidades variantes en sus datos con el objetivo de hallar todos los casos posibles en la búsqueda de errores. Este indica también las pruebas independientes de cada módulo o componente de la aplicación.

La fase de pruebas verifica y valida que el producto final sea eficaz y funcional. Para llevar a cabo este proceso se debe pensar a la inversa que en el desarrollo; hay que ir de problemas más pequeños y concretos (pruebas unitarias) hasta los más grandes y generales (pruebas de integración). Esta fase se realiza antes de entregar una aplicación para su finalización y transferencia y también en el mantenimiento de esta (además de todas las pruebas que sean necesario realizar en cada fase independiente del proyecto); es una fase muy importante y costosa aunque está infravalorada por la mayoría de desarrolladores. El número de pruebas no puede valorarse en función del número de líneas de código o variables del proyecto.

Cuanto más tiempo se tarda en detectar y solucionar errores en el código, más costará solucionarlo, tanto económicamente como a nivel lógico. La mayoría de ellos suelen concentrarse en las primeras fases del proyecto.

Un ingeniero de pruebas debe disponer de buenas habilidades comunicativas, concentración, curiosidad e intuición, saber trabajar en equipo y bajo presión.

***Clasificaciones de los tipos de pruebas***

El proceso en la fase de pruebas suele ser muy repetitivo y trabajoso, por lo que se intenta automatizar los procesos mediante otros programas software especializados.

Tipos de pruebas según la fase de desarrollo: Pruebas unitarias, pruebas de integración, pruebas de carga y pruebas de aceptación. También se pueden clasificar en función de cómo se realizan: Pruebas unitarias, pruebas de caja negra, pruebas de caja blanca y revisiones.

Existen otras pruebas llamadas pruebas de sistema, estas validan la integración del software en el sistema del usuario final. Estas pruebas están compuestas por: Prueba de rendimiento, de resistencia, de robustez, de seguridad, de usabilidad y de instalación.

***Pruebas unitarias***

Las pruebas unitarias se plantean a pequeña escala. Estas consisten en probar unidad a unidad los módulos que constituyen la aplicación, es decir, son el procedimiento para probar el correcto funcionamiento de un módulo por separado. Antes de estas pruebas se realiza una fase informal que consiste en ejecutar el código para validar que funciona. Existen herramientas de software que permiten automatizar las pruebas como NUnit. Estos entornos de trabajo / conjuntos de clases realizan pruebas unitarias asignándoles los valores deseados para cada una. Estos tipos de pruebas deben ser automáticas, completas, reutilizables e independientes.

***Pruebas de caja negra***

Las pruebas de caja negra llevarán a cabo la comprobación del funcionamiento de un componente de la aplicación mediante su interfaz, sin analizar el código. Se basa en comprobar que el componente funciona correctamente cuando se introducen datos correctos, erróneos, equivalentes o extremos superiores e inferiores (todas las posibles combinaciones). Evalúa el funcionamiento a través de la respuesta que obtiene, sin entrar en detalles ni depurar código. Para estas pruebas se han diseñado diferentes técnicas:

1. Clases de equivalencia: Determina los tipos de entrada y salida, y los agrupa.
2. Análisis de los valores límite: Estudian los valores iniciales y finales.
3. Estudio de errores típicos: Diseño de casos en los que se podría dar un error típico.
4. Manejo de interfaz gráfica: Caso de prueba que descubre errores del manejo de componentes de la interfaz como ventanas, botones, etc…
5. Datos aleatorios: Es una técnica complementaria, se trata de usar una herramienta que automatice las pruebas y genere de forma aleatoria los casos de prueba.

Las pruebas de caja negra son independientes a cualquier lenguaje o paradigma de programación, por lo que son válidas en cualquier tipo de proyecto.

***Pruebas de caja blanca***

Las pruebas de caja blanca son las encargadas de analizar de qué manera un componente software resuelve un problema determinado, basándose en los detalles internos de implementación. Estas se fijan en cómo se ha implementado el código en un módulo / aplicación para determinar qué juego de pruebas se van a usar; estos deben prever todos los casos posibles realizando un pequeño examen para comprobar los distintos caminos lógicos del software. El método de cobertura de flujo de control establece estos juegos de pruebas, consiste en usar la estructura de control del software para obtener los casos de prueba que pasarán una vez por cada camino del software. Con esto se consigue un mínimo de garantías de eficacia ya que diseña una prueba para cada elemento del código como un if o bucles.

***Revisiones***

Es un tipo de prueba unitaria que consiste en inspeccionar directamente el código al completo o por módulos siguiendo una serie de normas. Estas las realiza el propio programador pero también se usan las técnicas en grupo como las *revisiones técnicas;* se pueden encontrar en reuniones, walkthroughs y en inspecciones.

Las *reuniones* son situaciones con poca preparación en las que se encuentran los desarrolladores para revisar el código y calidad, además de detectar errores antes de la fase de pruebas.

Los *walkthroughs* son un procedimiento más formal aplicado en la revisión del código en el que el desarrollador se reúne con expertos revisores para preguntar dudas y encontrar soluciones.

Las *inspecciones* son el procedimiento más formal, requieren una preparación previa y la preparación de un moderador que reparte papeles y trabajos entre los participantes.

Esta fase es muy importante como prueba unitaria pero también debe usarse en tareas relacionadas al control de calidad y otras validaciones y verificaciones del sistema.

Algunas ventajas de las pruebas unitarias son: Documentación del código sobre pruebas, simplificación de la integración de módulos y separación de la interfaz gráfica y la implementación del código.

***Pruebas de integración***

Las pruebas unitarias no identifican todos los errores que hay o habrá en el código por sí mismas. Hay errores que pueden aparecer al integrar más módulos o cargar datos, por ello es necesario tener en cuenta otro tipo de pruebas.

Las pruebas de integración toman como base las pruebas unitarias y consisten en una progresión ordenada de test para usarse en cada módulo hasta que se integra en el sistema al completo. Estas verifican que al enlazarse los módulos funcionen correctamente. Suelen realizarse de manera gradual; se prueban dos componentes y si funcionan se incorpora un tercero, así sucesivamente.

Uno de los objetivos principales de las pruebas de integración es localizar errores en las interfaces entre los distintos módulos. En el desarrollo del software existe una organización jerárquica de los módulos, organizados por niveles. Este tipo de organización facilita la ejecución y localización de errores al contar con una estructura ordenada de los módulos y permite saber dónde se ha encontrado y su motivo o descripción. A partir de estas informaciones se pueden desarrollar distintos tipos de estrategias de pruebas de integración como:

1. Pruebas de integración ascendentes: Estas pruebas comienzan combinando en grupos los módulos de bajo nivel para probarlos. Para realizarlas es necesario diseñar módulos como componentes de pruebas (impulsores / controladores de pruebas) que permiten simular el comportamiento de los módulos superiores.

Los programas controladores tienen la misma interfaz del módulo que sustituyen, pero no realizan su función por lo que son fáciles de construir. Con estos controladores se realizan las pruebas de las interfaces y de los módulos; cuando se ha probado un grupo de módulos se continúa con el siguiente grupo del mismo nivel. Una vez se ha probado un nivel se sustituyen los controladores por módulos superiores hasta que se integren todos los niveles.

1. Pruebas de integración descendentes: Primero se prueban los módulos de nivel superior y después se van integrando los componentes de la siguiente capa. Una vez que se prueban, se pasa al nivel inferior; para esto se necesita crear componentes stub / módulos simuladores de pruebas que emulan su comportamiento y tienen la misma interfaz. Estos módulos stub simulan las funciones que deberían realizar los módulos reales que sustituyen, por ello son más difíciles de construir que los programas controladores.
2. Pruebas de integración combinadas: Estas pruebas combinan las integraciones ascendentes y descendentes; se realizan en paralelo, por lo que se ahorra tiempo. Los módulos individuales se prueban con controladores y stubs, para probar las capas superiores e inferiores sustituyéndolos por módulos probados. En los niveles / capas superiores se usa una estrategia descendente, y en las inferiores ascendentes hasta que se encuentran en una capa intermedia.

1. Pruebas de integración de big bang / de gran explosión: Primero prueba todos los métodos individualmente y después todos los componentes se integran a la vez y se realizan pruebas. Su inconveniente es que al detectar un fallo es muy difícil localizarlo para corregirlo.

En todas estas estrategias deberá decidirse el orden en que se integrarán los módulos. Un buen criterio sería comenzar con los módulos críticos del sistema y los que sean más susceptibles de contener errores por ser más complejos o necesitar más requisitos.

***Pruebas de carga y aceptación***

Al realizar las pruebas unitarias y de integración, el próximo paso será llevar a cabo las pruebas de carga y posteriormente las pruebas de aceptación.

Las pruebas de carga son pruebas cuyo objetivo es comprobar el rendimiento y la integridad de la aplicación una vez terminada, con datos y situaciones reales, sobrecarga y descontrol de datos.

Tras estas pruebas se realizan las pruebas de aceptación; estas son realizadas por el cliente y tienen como objetivo obtener la aprobación del cliente sobre la calidad del funcionamiento. Estas son pruebas funcionales sobre el sistema completo, buscando ayuda en la especificación de requisitos y en el manual de usuario.

El cliente siempre tiene la razón, por lo que muchos desarrolladores usan dos técnicas:

1. Pruebas alfa: Consisten en invitar al cliente al lugar de desarrollo a probar el sistema. Se trabaja de manera controlada y supervisada con un experto para ayudarle a usar el sistema y analizar los resultados.
2. Pruebas beta: Se realizan después de las pruebas alfa, estas se desarrollan en el entorno del cliente por lo que no tendrá una ayuda directa. Se busca que el cliente informe posteriormente a los desarrolladores de los errores que haya encontrado, si se diese el caso.

Estas pruebas son muy eficaces y son habituales en productos que se venderán a muchos clientes y tendrá un gran volumen de uso.

***Pruebas de sistema y seguridad***

Las pruebas de sistemas se llevan a cabo después de haber realizado las pruebas unitarias, de integración, de carga y de aceptación. Después de que el usuario haya testeado la aplicación estas pruebas sirven para validar la aplicación. Algunos tipos de pruebas a desarrollar durante las pruebas de sistema son:

1. Pruebas de rendimiento: Se valoran los tiempos de respuesta, el espacio que ocupa en memoria, etc…
2. Pruebas de resistencia: Se valora la resistencia de la aplicación en determinadas situaciones.
3. Pruebas de robustez: Se valora la capacidad para soportar varias entradas incorrectas.
4. Pruebas de seguridad: Ayudan a determinar los niveles de permisos de los usuarios, las operaciones que se pueden realizar y las de acceso al sistema.
5. Pruebas de usabilidad: Determinan la calidad de la experiencia de un usuario en función de cómo interactúe con el sistema.
6. Pruebas de instalación: Indican operaciones de arranque y actualización de las aplicaciones.

Estas pruebas comprueban varias cosas: Si la aplicación realiza sus funciones y se comporta como debería; los tiempos de respuesta para las pruebas de rendimiento, volumen y sobrecarga; la disponibilidad de los datos en el momento de recuperación de un error, la usabilidad, la instalación, el entorno y la seguridad.

Las pruebas de validación comprueban si se cumplen los requisitos propuestos a nuestro sistema.

***Pruebas de regresión y pruebas de humo***

Las pruebas de regresión son otro tipo de pruebas de software que buscan detectar nuevos errores o problemas que puedan producirse al realizar cambios, mejoras o actualizaciones en la aplicación. Estos cambios pueden solucionar un problema y provocar otros sin haberlo previsto, por ello es necesario llevar a cabo estas pruebas al finalizar el resto de pruebas.

Se pueden automatizar la detección de estos errores con ayuda de herramientas específicas. La automatización es complementaria al resto de pruebas pero facilita su repetibilidad.

Las pruebas de humo surgen de la fabricación de hardware, se dice que si al reparar un componente no echa humo es que funciona correctamente. Estas pruebas se usan para describir la validación de los cambios de código en el software antes de que los cambios se registren en la documentación.

***Validación formal de la corrección***

Para llevar a cabo la validación y pruebas de una aplicación es necesario establecer con precisión su semántica. Estas especificaciones se dan en lenguaje natural. Los fundamentos de la técnica de precondición y postcondición son establecer condiciones para que el programa funcione correctamente (precondiciones). Las postcondiciones describen las condiciones que se cumplen una vez finalizado el programa. Con esta técnica se puede describir el programa y validarlo antes de iniciar su implementación.

***Derivación formal de programas***

La derivación formal demuestra matemáticamente que un programa cumple con las especificaciones. El diseño formal de algoritmos usa métodos para abordar sus tres fases:

1. Especificación: Técnica que especifica la semántica internacional de un algoritmo de forma breve y concisa.
2. Derivación: A partir de una especificación se pueden derivar las instrucciones que debe tener el algoritmo y una vez escrito, verificar su corrección.
3. Verificación: Tiene una gran importancia ya que verifica la corrección de un algoritmo antes de ser escrito en un lenguaje concreto mediante la verificación a priori (se prueban los algoritmos con herramientas de depuración).

***Pruebas de software: Metodologías, IEEE, TMM***

Las metodologías de gestión de proyectos de desarrollo de software tratan otro tipos de temas relacionados con las pruebas software.

1. Metodología de sistemas de información: Hace referencia a la fase de pruebas e indican actividades y tareas a realizar junto con las técnicas y herramientas necesarias para ello. Métrica 3 pertenece al CSAE (Consejo Superior de Administración Electrónica), este ofrece unas fases, actividades y tareas que ayudan al desarrollo de un proyecto software. Es similar al modelo en cascada o el modelo en V, en él se encuentran fases como: Estudio de viabilidad, análisis del sistema, diseño del sistema, construcción del sistema, implantación y aceptación del sistema y mantenimiento del sistema. En la fase de implantación y aceptación del sistema se pueden encontrar actividades como: IAS 5 y 6 que realizan pruebas de implantación y aceptación del sistema. Además ofrece explicaciones técnicas para el desarrollo de: Pruebas unitarias, de integración, de sistema, de implantación, de aceptación, de regresión, revisiones formales y revisiones técnicas.
2. IEEE Estándar 829-1983: Institute of Electrical and Electronic Engineers es una asociación internacional dedicada a la estandarización. Esta ha desarrollado normas y estándares para los procesos de pruebas en desarrollos como el Standard for Software Test Documentation. Este estándar presenta un conjunto de normas y especificaciones sobre su ámbito, enfoque, recursos y calendario. Sus especificaciones se refieren al diseño de pruebas, elementos de los casos de pruebas, características de las pruebas, actividades de las pruebas, función de cada prueba, informes del desarrollo de las pruebas, registro de pruebas y riesgos.
3. Test para los modelos de madurez: Existen herramientas, técnicas y asociaciones que han creado modelos como CMM / Modelo de Madurez de Capacidades. Este presenta un modelo de evaluación de procedimientos de una organización, es decir es un modelo estándar de referencia de la industria del desarrollo software. A partir de este se creó el TMM / Test para Modelos de Madurez. Actualmente hay un sucesor para cada uno de estos dos modelos: CMM > CMMi / Integración de Modelos de Madurez de Capacidades y TMM > TMMi / Integración de Test de Modelos de Madurez.

Este último es un modelo detallado para la mejora del proceso de pruebas, este consta de cinco niveles:

1. Inicial: Verifica que el sistema funciona sin problemas.
2. Definición: Verifica que los requisitos iniciales se cumplen.
3. Integración: Verifica que no existan errores ni problemas en el funcionamiento del software una vez integrado.
4. Gestión y medición: Verifica y evalúa durante todo el ciclo de vida el desarrollo del proyecto.
5. Optimización: Consiste en desarrollar nuevos procesos y mejorar el proyecto para la prevención de posibles errores en el futuro.

***Herramientas para desarrollar pruebas***

Algunas herramientas para facilitar el desarrollo de las pruebas están integradas en los entornos de desarrollo, pero la mayoría de ellas son independientes y será necesario seleccionarlas en función del entorno de trabajo y el lenguaje de programación que se esté utilizando.

Algunas de ellas son:

1. JUnit: Es una herramienta de pruebas para proyectos desarrollados en el lenguaje de programación Java. Esta sirve para realizar pruebas unitarias ejecutando clases Java de forma controlada para evaluar el comportamiento de cada método. Es una herramienta de código abierto. También se usa para automatizar las pruebas de regresión; esta puede usarse de forma independiente o integrada en otras herramientas de desarrollo.
2. NUnit: Es una evolución de JUnit que acepta muchos más lenguajes de programación además de Java. Es un entorno de trabajo para Microsoft .NET que ofrece un nivel más bajo de integración con otras herramientas.
3. XUnit: Es un conjunto de entorno de trabajo de pruebas, extendidas de la herramienta JUnit y NUnit. Este también es de código abierto y se encuentra en constante evolución. Contiene extensiones para: Datos, carga masiva de datos, interfaces web e interfaces Swing.
4. Clover: Es una herramienta que permite analizar la cobertura de código Java, ofrece métricas para la evolución del impacto de sus pruebas. Llevará a cabo las pruebas unitarias por su cuenta, aunque también se puede integrar en otras herramientas de desarrollo de software.
5. OpenSTA: Open Systems Testing Architecture se diseñó en torno al lenguaje de programación CORBA. Es un conjunto de herramientas que tienen capacidad para ejecutar secuencias de sentencias HTTP y HTTPS para realizar pruebas con carga masiva de datos, pruebas de estrés y pruebas de rendimiento. Permite realizar estadísticas a partir de pruebas de rendimiento y registros virtuales con una carga masiva controlada de datos.
6. Bugzilla: Es un software que asegura la calidad y detecta errores en el software. Está incluida en el proyecto Mozilla, bajo licencia pública con código abierto. Este permite realizar un seguimiento de cualquier código web y puede realizar ejecuciones paso a paso hasta detectar la cantidad de datos que se transfieren en cada comunicación o el tiempo de carga.

Algunos de los aspectos que se pueden mejorar con el uso de Bugzilla son: Seguimiento de defectos, mejora de la comunicación, aumento de la calidad del producto, mejora la satisfacción del cliente, aumenta la productividad y se adapta a múltiples situaciones.

***Conclusión***

Las pruebas son una fase muy importante del desarrollo de un software en el que prima la constancia de las mismas desde el principio del desarrollo hasta que se implementa la aplicación, incluso después de lanzarse al mercado en casos como modificaciones. Debe tenerse en cuenta y realizarse más a menudo para que no se haga una tarea pesada y laboriosa.