UNIDAD 3. DISEÑO Y REALIZACIÓN DE PRUEBAS

Módulo Profesional: Entornos de Desarrollo

Contenido

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos relacionados con las pruebas de software, la automatización de las mismas, la relación existente con la calidad de los productos software y el software JUnit. En cuanto a las pruebas a nivel de aplicación, es importante destacar la existencia de varias de ellas, las cuales van a ser aplicadas según la funcionalidad a probar. En este sentido, las pruebas más comunes que se realizan sobre el software son las de unidad, integración, validación, sistema y aceptación. Hay otros tipos de pruebas como las pruebas de rendimiento, que engloban una serie de pruebas como las de estrés, estabilidad y escalabilidad, en función del número de usuarios concurrentes que ejecutarán una acción a la vez. Además, las pruebas de regresión van a jugar un papel fundamental en el desarrollo de software, ya que toda aplicación debe ser nuevamente probada toda vez que se le añada nueva funcionalidad, para poder comprobar si las nuevas líneas de código han afectado o no a funcionalidad existente y probada con anterioridad. Todo este conjunto de pruebas es analizado en esta unidad.

INTRODUCCIÓN

Al hilo de las introducciones propuestas en las unidades anteriores, es importante recordar que la ocupación principal de las factorías de software es trabajar de forma constante desarrollando aplicaciones a demanda de sus clientes. Es crucial asegurar la entrega del producto software totalmente probado. Para ello es necesario realizar una serie de pruebas de forma que garantice que la aplicación cumple los requisitos y objetivos del cliente y, por supuesto libre de errores. También es sumamente importante destacar que una prueba software es efectiva si ha servido para detectar algún error o anomalía en el software.

Los productos software bien probados y testeados, libres de errores, garantizan un nivel de calidad importante frente a aquellos que presentan fallos o cuya estrategia de pruebas no ha sido ejecutada decentemente.

CASO INTRODUCTORIO

El equipo de desarrollo de Juan ha generado la primera versión de la aplicación. A priori, toda la funcionalidad ha sido probada por los propios programadores a medida que han ido implementando los requisitos. Sin embargo, se hace necesario que el equipo de pruebas establezca un plan de pruebas, que consiste en analizar la aplicación y crear tantos casos de prueba como sean necesarios. Por ejemplo, para loguearse en el aplicativo, es necesario probar con un usuario y una contraseña válida y comprobar si

el resultado esperado es el mismo que el resultado obtenido. En caso afirmativo, la prueba verifica el funcionamiento del login. Pero también se hace necesario probar con usuarios y contraseñas no válidas, jugando con las diversas k-suisticas (usuario correcto, contraseña no válida; usuario incorrecto, contraseña no válida, etc.).

Al finalizar la unidad el alumnado:

- Será capaz de realizar las pruebas necesarias para comprobar que la aplicación cumple con los requisitos del cliente.
- Será capaz de realizar el plan de pruebas del software.
- Conocerá las herramientas de depuración existentes en el mercado.
- Aprenderá las normas de calidad del software vigentes y la relación existente entre las pruebas y la calidad del software.

1. PLAN DE PRUEBAS

Se denomina **plan de pruebas** a un conjunto de acciones y pruebas que cumplen con las siguientes características:

- Debe especificar de una forma correcta el **objetivo** de la prueba, por ejemplo, comprobar propiedades del software como corrección, robustez, fiabilidad, amigabilidad, etc.
- Especificar la **medida** en la que se mostrará el resultado.
- Además del objetivo, se debe dejar claro en qué va a **consistir** la prueba hasta el último detalle, desde las entradas que pueda tener la prueba, su ejecución, las salidas, etc.
- En el plan de pruebas y más concretamente en cada caso de prueba, se debe definir siempre cuál es el **resultado esperado**.

PARA SABER MÁS

En este enlace se puede aprender más sobre el plan de pruebas y cada uno de los tipos de pruebas:

http://pruebasdelsoftware.wordpress.com/

Diferentes pruebas que se deben incluir en el plan de pruebas:

- Se deben realizar pruebas donde los valores de entrada y salida correspondan con **valores límite**, para analizar condiciones límite que se puedan producir.
- Pruebas con **valores normales**, diferentes de los límites, pero utilizando un amplio rango de valores.
- Realizar **pruebas de error**, es decir, pruebas basadas en datos con los que se presupone que se producirán errores.
- Normalmente se deben añadir **pruebas especiales** que dependerán del objetivo que se ha definido en el plan de pruebas, se deberán por tanto diferenciar pruebas para comprobar la robustez, la velocidad a la que se ejecuta la aplicación, etc.

1.1 Estrategias de aplicación de pruebas

Las pruebas comienzan a nivel de **módulo**. Una vez terminadas, progresan hacia la integración del sistema completo y su instalación. Culminan cuando el cliente acepta el producto y se continúa a su explotación inmediata.

Objetivo del documento del plan de pruebas: Señalar el enfoque, los recursos y el esquema de actividad desde prueba, así como los elementos a probar, las características, las actividades de prueba, el personal responsable y los riesgos asociados.

Esquema del documento del plan de pruebas:

1. Identificador único del documento.
2. Introducción y resumen de elementos y características a probar.
3. Elementos software a probar.
4. Características a probar.
5. Características que no se probarán (si fuera necesario).
6. Enfoque general de la prueba.
7. Criterios de paso/fallo para cada elemento.
8. Criterios de suspensión y requisitos de reanudación.
9. Documentos a entregar.
10. Actividades de preparación y ejecución de pruebas.
11. Necesidades de entorno.
12. Responsabilidades en la organización y realización de las pruebas.
13. Necesidades de personal y formación.
14. Esquema de tiempos.
15. Riesgos asumidos por el plan y planes de contingencias.
16. Aprobaciones y firmas con nombre y puesto desempeñado.

COMPRUEBA LO QUE SABES

"GOOD TESTING INVOLVES MUCH MORE THAN JUST RUNNING THE PROGRAM A FEW TIMES TO SEE WHETHER IT WORKS". ¿Qué se puede extraer de esta reflexión?

Objetivo del documento de especificación del diseño de pruebas: Especificar los procesos necesarios e identificar las características que se deben probar con el diseño de pruebas.

Objetivo del documento de especificación de caso de prueba: Definir al menos uno de los casos de prueba identificando una especificación del diseño de las pruebas.

Objetivo del documento de especificación de procedimientos de prueba: Especificar los distintos pasos para la ejecución de, al menos, un conjunto de casos de prueba o los pasos utilizados para analizar un

elemento software con el propósito de evaluar un conjunto de características del mismo.

1.2 Pruebas de unidad

Pruebas formales que permiten declarar que un módulo está listo y terminado. Se refiere una unidad de prueba a uno o más módulos que cumplen que:

- Todos son del mismo programa.
- Al menos uno de ellos no ha sido probado.
- El conjunto de módulos es el objeto de un proceso de prueba.
- Este tipo de pruebas se automatizan a través de herramientas como JUnit.

1.3 Pruebas de integración

Los módulos individualmente probados se integran para comprobar sus interfaces (comunicación con otros módulos) en el trabajo conjunto. Se han de tener en cuenta las interfaces entre componentes de la arquitectura del software.

Implican una progresión ordenada de pruebas que van desde los componentes o módulos y que culminan en el sistema completo.

PARA SABER MÁS

Pruebas reales:

http://jacace.wordpress.com/2010/04/22/ejemplo-de-pruebas-de-software-parte-1/

Tipos fundamentales de integración:

- **Integración incremental**: Se combina el siguiente módulo que se debe probar con el conjunto de módulos que ya han sido probados.
- **Ascendente**: Se comienza por los módulos hoja y se va subiendo.
- **Descendente**: Se comienza por el módulo raíz y se va bajando.
- **Integración no incremental**: Se prueba cada módulo por separado y seguidamente se integran todos de una vez, probando todo el programa.

Habitualmente, las pruebas de unidad y de integración se realizan en el mismo tiempo. Al añadir un nuevo módulo, el software cambia y se

establecen nuevos caminos de flujo de datos, nueva E/S y nueva lógica de control. Puede haber problemas con acciones que anteriormente funcionaban bien, por lo que se deberán ejecutar de nuevo el conjunto de pruebas que se han realizado anteriormente para asegurarse que los cambios no han dado lugar a cambios colaterales, estas pruebas son las llamadas **pruebas de regresión**.

1.4 Pruebas de validación

Se llevan a cabo cuando se han terminado las pruebas de integración. El software terminado se prueba como un conjunto para comprobar si cumple o no tanto los requisitos funcionales como los requisitos de rendimiento, seguridad, etc. La validación se consigue cuando el software funciona según las expectativas del cliente en función de los requisitos definidos.

ARTÍCULO DE INTERÉS

Se recomienda leer la información suministrada en el seminario de Verificación, Validación y Pruebas Unitarias:

https://silo.tips/download/seminario-de-verificacion-validacion-y-pruebas-unitarias

1.5 Pruebas de sistema

El software ya cumple con todos los requisitos, por tanto, ahora se debe integrar con el resto del sistema, instalando la aplicación en el mismo sistema operativo en el que el cliente lo va a utilizar y con las mismas características hardware, probando su funcionamiento. Se debe comprobar, además, la seguridad en el acceso a los datos, el rendimiento del software con grandes cargas de trabajo, la tolerancia a fallos en la red, etc. Se debe probar la recuperación del sistema en caso de errores en la aplicación que puedan hacer que finalice de forma no esperada, recoger los logs de la aplicación, etc.

Estas pruebas se deben ejecutar y observar su rendimiento en condiciones límite y de carga máxima (pruebas de rendimiento). Estas pruebas sirven para verificar que se han integrado adecuadamente todos los elementos del sistema y que se realizan las funciones apropiadas, realizándose sobre lo que se denomina entorno de pre-producción.

1.6 Pruebas de aceptación

Se instala el software en el propio entorno de explotación (entorno de producción) y el cliente comprueba que el software cumple con los requisitos establecidos.

2. AUTOMATIZACIÓN DE PRUEBAS

Para la automatización de pruebas se van a tener en cuenta 3 condicionantes: casos de prueba, procedimientos de prueba y componentes de prueba:

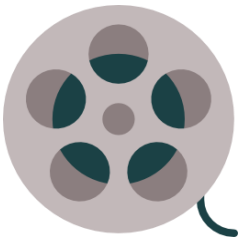
- **Casos de prueba**: Cada caso de prueba debe definir el resultado de salida esperado que se comparará con el realmente obtenido.
- **Procedimientos de prueba**: Especificar cómo y cuándo se realizarán las pruebas.
- **Componentes de prueba**: Automatiza uno o varios procedimientos de prueba o parte de ellos.

¿Cómo describir los componentes de la prueba?

- Por cada caso de prueba, implementar el código correspondiente en el componente.
- Se escribe el código del componente de tal manera que recorra en una Base de Datos los casos de prueba y los ejecute.
- Cada vez que se añada un caso de prueba, simplemente se añade en la Base de Datos, pero el código del componente no cambiaría.
- Se pueden usar entornos de trabajo disponibles para pruebas (JUnit).

ARTÍCULO DE INTERÉS

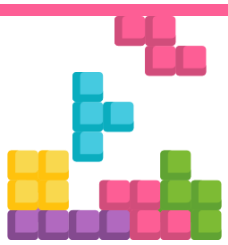
Aunque al final de la unidad se encuentra una de las herramientas utilizadas para automatizar las pruebas (JUnit), se facilita un enlace a otras herramientas del mercado: http://robotium.com/ y http://www.seleniumhq.org/projects/ide/



VIDEO DE INTERÉS

Se recuerda visitar el siguiente vídeo para ampliar información sobre las ventajas de la automatización:

https://www.youtube.com/watch?v=wyzDnE3UeIc



EJEMPLO PRÁCTICO

A nuestro equipo de trabajo llega una aplicación software que debe ser probada con cierta urgencia, ya que el cliente espera que le sea entregada y, por consecuencia, probada, en los próximos días. ¿Cómo se procedería a realizar las diferentes pruebas?

Solución:

Es imprescindible hacer uso del plan de pruebas implementado para dicha aplicación. En él se encontrarán los casos de prueba que se hayan definido. Cada caso de prueba tendrá un enunciado de la prueba, datos de la ejecución de la prueba (nombre del tester, fecha, hora, lugar, equipo en el que ha sido privado, etc.) un caso esperado y una zona para indicar el caso obtenido. Siempre que el caso esperado sea idéntico al caso obtenido se podrá decir que la prueba no ha conseguido encontrar ningún tipo de defecto. En muchas ocasiones, esta tarea se puede realizar de forma automática, mediante el uso y la configuración de herramientas como ALM de HP, entre otras.

3. HERRAMIENTAS DE DEPURACIÓN

Se conoce **depuración** al proceso de identificar y corregir defectos que pueda tener el software creado. Son muchos y muy frecuentes los errores humanos que se cometen mientras se escribe un código, aumentando bastante la complejidad de un software. Es por ello que la depuración se convierte es un aliado para los programadores bastante eficiente a la hora de corregir este tipo de problemas. Los compiladores tienen la posibilidad de ejecutar un programa paso por paso, permitiendo al desarrollador comprobar los valores intermedios de las variables, las posibles entradas, salidas, etc. Los depuradores se deben usar para realizar un seguimiento sobre el comportamiento dinámico de los programas.

COMPRUEBA LO QUE SABES

¿A través de qué icono se representa (y se ejecuta) la acción de depuración de código tanto en el entorno de desarrollo Eclipse como en NetBeans?

¿SABÍAS QUE...?

Utilizar las diversas herramientas de depuración como ayuda para depurar aplicaciones en un entorno de desarrollo autónomo o colaborativo.

En la práctica, no obstante, su uso es tedioso y sólo serán eficaces si se persigue un objetivo claro y previamente definido o al menos previsible. Por norma general, su utilización es consecuencia de la detección de un error. Si el programa se comporta mal en un cierto punto, se debe averiguar la causa para poder repararlo. Esta causa a veces es inmediata, por ejemplo, una asignación de variables errónea o un operador equivocado.

Sin embargo, el error también puede depender del valor concreto de los datos en un cierto punto y hay que buscar el motivo. Antes de acudir al depurador hay que delimitar lo máximo posible la zona donde se puede encontrar el fallo, identificar el dominio del mismo e investigar las propiedades de los datos que lo provocan.

3.1 Consejos para la depuración

- Analizar la información e intentar resolver los errores antes de depurar.
- Usar herramientas de depuración sólo como último recurso secundario.
- Se deben depurar los errores individualmente.
- Fijar la atención en los datos.
- Al corregir un error, volver a realizar todas las pruebas y volver a depurar.
- Al corregir un error puede que aparezcan otros.

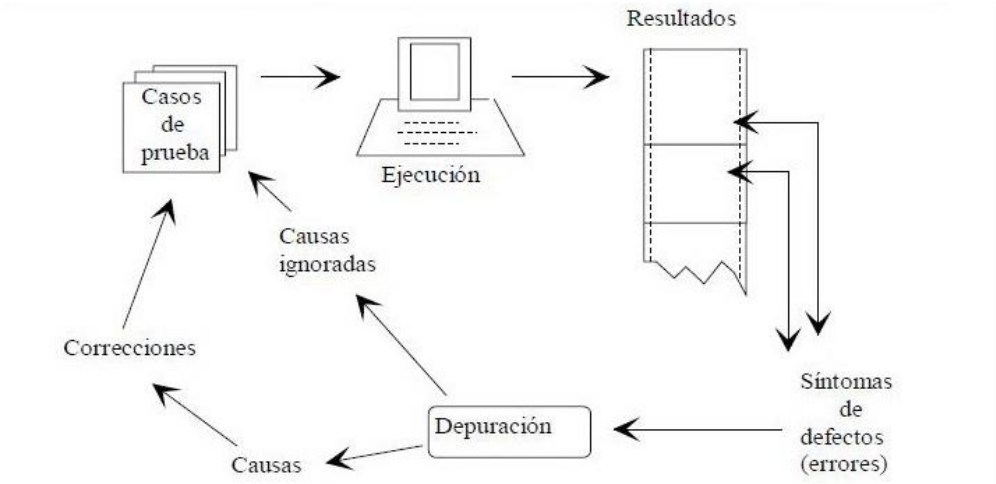


Imagen: Relación entre pruebas y depuración

VIDEO DE INTERÉS

Acción de depuración en Eclipse:

https://www.youtube.com/watch?v=lFLFxVUUD74

3.2 Análisis causal

El **análisis casual** se toma información sobre los errores que se han depurado. Por tanto, se debe recoger información y para poder prevenir los próximos errores, prediciendo posibles fallos: ¿Qué se hizo mal?, ¿cómo se podría haber prevenido?, ¿cómo se podría haber detectado antes?, ¿cómo se encontró el error?, ¿por qué no se detectó antes?, ¿cuándo se cometió?, ¿Quién lo hizo?

4. CALIDAD EN EL SOFTWARE

El software desempeña un papel crítico a día de hoy en la gran mayoría de las empresas. Si además una empresa que realiza su software desea ofrecérsela a terceros, necesita de una garantía de que se está haciendo bien. En el mundo del desarrollo software como en el de otros servicios informáticos, se necesita de una supervisión constante que consta de una normativa que regulan los criterios de calidad. En este sentido, los estándares ISO15505 o CMMI proporcionan unos planteamientos estructurados para desarrollar aplicaciones software fiable.

Los distintos modelos de calidad hoy en día podrían clasificarse en función de la calidad en el producto software. En los procesos en donde se definen la creación del software o en los sistemas de gestión, las normativas se aplican en las distintas etapas del ciclo de vida de los proyectos informáticos y contribuyen a mejorar la calidad del software.

PARA SABER MÁS

En el siguiente enlace podrás visitar la página donde se detallan las normas a seguir para las pruebas del software:

http://in2test.lsi.uniovi.es/gt26/?lang=es

En la actualidad existen diversas opciones, tales como las siguientes:

ISO 9001: No se basa de una forma exclusiva en la etapa del desarrollo, esta norma identifica el alcance que tendrá el software en los procesos productivos que se generan, por ejemplo, en la identificación de requisitos, en la entrega y mantenimiento.

ISO/IEC 9003: Norma no certificable, es una guía de buenas prácticas para definir con detalle conceptos sobre los procesos de la organización.

ISO/IEC 9126: Norma desarrollada entre 1991 y 2001, en donde se definen unas características y subcaracterísticas de calidad del producto, así como la calidad en su uso que van a permitir ayudar a medir la calidad del software en base a ciertos atributos de calidad.

ISO 25000, de la familia de normas 25000, tiene 5 partes publicadas y establecen un modelo de calidad tanto para el software como para la evaluación.

ISO/IEC 12207, Information Technology / Software Life Cycle Processes, es el estándar para los procesos de ciclo de vida del software de la organización. Es la base para ISO 15504-SPICE (Software Process Improvement And Assurance Standards Capability Determination). Es una norma que define un modelo para todo tipo de empresas, está en continuo desarrollo. La implantación y evaluación externa para la certificación se puede realizar por etapas, de tal manera que en años posteriores irá aumentando su alcance. Esta norma evalúa la calidad software por niveles de madurez y la mejora de procesos.

CMMI (Capability Maturity Model Integration) es a nivel mundial un requisito para acceder a la oferta de servicios software. Ofrece una guía para implementar además del software una estrategia de calidad y mejorar

los procesos de una organización sobre el desarrollo y mantenimiento de software. Es certificable a partir de organismos privados. Es una norma dirigida a grandes empresas con requisitos de calidad muy altos cuyo desarrollo se realiza en países donde no se encuentran sus oficinas centrales y que se apoyan en pequeñas empresas que ofertan outsourcing. Su certificación verifica y puntúa en distintos niveles la organización. En 2010 hay más de 200 empresas certificadas.

¿SABÍAS QUE...?

Para alcanzar un cierto nivel de madurez en CCMI, se hace necesario bastante tiempo y esfuerzo por parte de las empresas. El valor medio para alcanzar estos niveles es: Nivel 1 al 2, 5 meses; Nivel 2 al 3, 19 meses; Nivel 3 al 4, 21 meses; Nivel 3 al 5, 25.5 meses. A partir del Nivel 3 se aumenta la complejidad en alcanzar los niveles posteriores.

5. JUNIT

Una de las fases más importantes del ciclo de vida del software son las pruebas, ya que en ella se verifica si la aplicación cumple con los requisitos que fueron especificados por el cliente. Como se ha comprobado, existen diferentes tipos de pruebas: pruebas unitarias, que prueban los distintos módulos que forman parte del aplicativo y que indican el punto de partida para el resto de pruebas como las de integración, validación, sistema, regresión, etc.

JUnit es un framework o conjunto de herramientas open source para realizar las pruebas unitarias sobre software creado en lenguaje de programación Java. Con JUnit se organizan las pruebas y se reduce el tiempo que necesita el desarrollador o tester y así podrá centrarse en la verificación de los resultados.

PARA SABER MÁS

En el siguiente enlace encontrará ejemplos de pruebas de unidad con NetBeans y JUnit:

http://alcasoft.blogspot.com.es/2013/05/java-pruebas-unitarias-con-junit-y.html

Las pruebas con JUnit se hacen de manera organizada y controlada sobre los distintos métodos que conforman las clases en Java, haciendo una comparación entre el valor esperado y el valor que devuelven los métodos. JUnit se encarga de crear los casos y las clases de pruebas, por lo que las pruebas se realizan de manera automática. JUnit crea, por cada clase de prueba, una clase con el mismo nombre que la clase original con el sufijo Test. Es en los **asserts** o condiciones donde se comprueba el correcto funcionamiento.

Los asserts son las afirmaciones de una proposición (línea de código) en un programa o, lo que es lo mismo, una condición que debe de cumplirse, el desarrollador debe considerar que el assert siempre es verdadero. Al tener condiciones también se pueden definir precondiciones y postcondiciones de la ejecución en determinadas líneas de código, lo que se conocen como pruebas unitarias.

JUnit proporciona métodos para realizar distintas pruebas y, dependiendo de las pruebas que se desee realizar, los métodos comprueban en distintos contextos el nivel de aceptación de una prueba:

AsserEquals	Proporciona sobrecargas que permiten comprobar si un valor real coincide con el esperado
AssertFalse	Cuando se sabe que la función siempre devuelve falso
AssertNotNull	Se utiliza cuando existe un caso donde no se devuelva null
AssertNotSame	Útil cuando se debe devolver un elemento de una lista que se puede usar, con el fin de determinar si ese elemento pertenece a la lista
AssertNull	Si un método retorna null se usa este assert para comprobar su veracidad o falsedad
Fail	Condicionales
FailNotEquals	Hace lo mismo que assertEquals pero esperando que la prueba no sea igual
FailNotSame	Esencialmente lo mismo que assertNotSame, salgo en lugar de causar un error que causa un fracaso

Imagen: Relación entre pruebas y depuración

5.1 Crear proyecto en Eclipse

Se procede a crear un proyecto en lenguaje de programación Java, creando dos clases y dentro de estas se implementarán 4 métodos que serán sobre los que se realizarán las pruebas unitarias, a través del entorno de desarrollo Eclipse.

Abrir el IDE Eclipse y crear nuevo Proyecto: File → New → Other → Java → JUnit → JUnit Test Case. Pulsar Next.

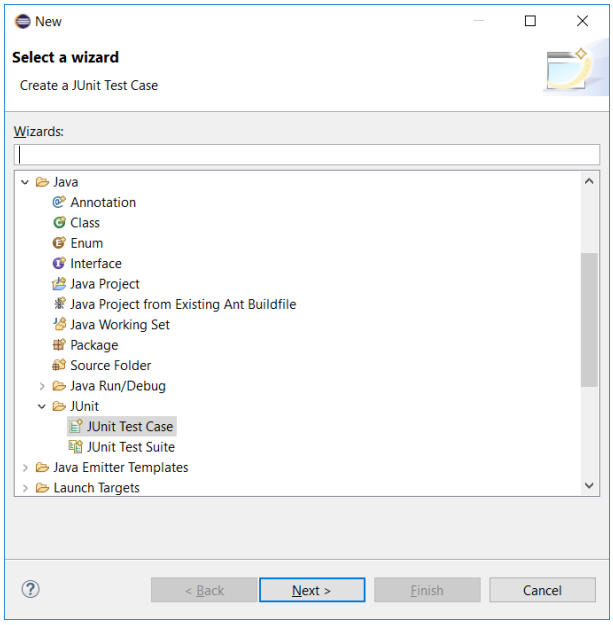


Imagen: Crear proyecto JUnit en Eclipse (1)

En la siguiente interfaz, justo en el nombre (Name) escribir SumaTest, para realizar unas pruebas unitarias sencillas sobre una suma. Pulsar Finish:

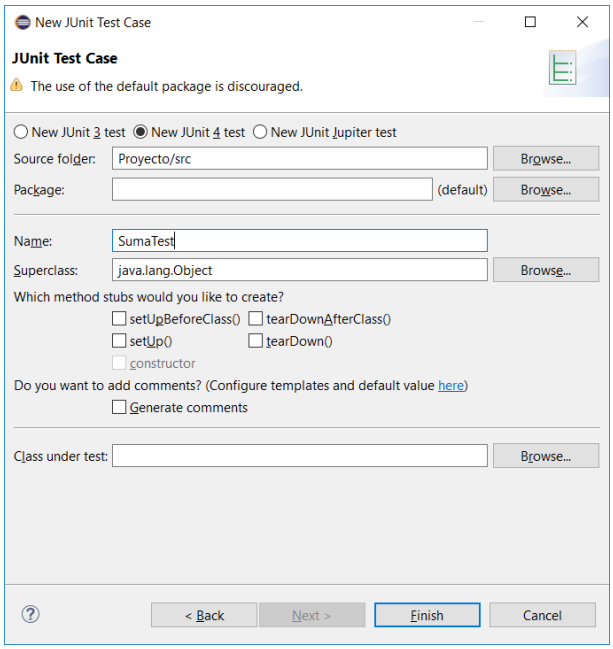


Imagen: Crear proyecto JUnit en Eclipse (2)

En este preciso momento se ha creado, de forma automática, una clase llamada SumaTest.java que contiene las siguientes líneas de código:

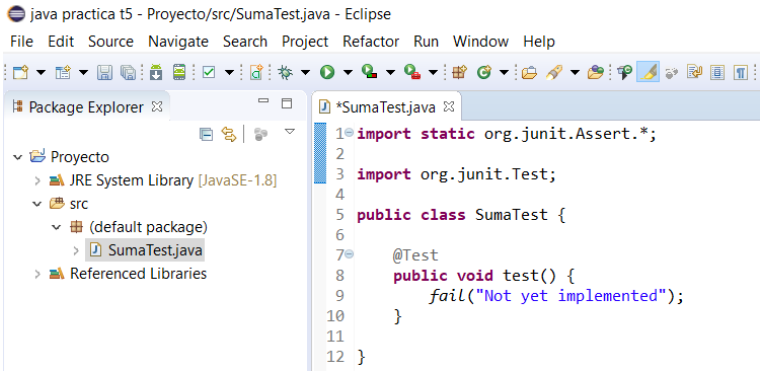
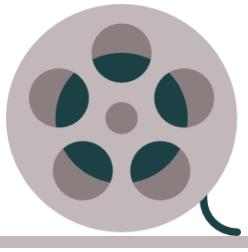


Imagen: Crear proyecto JUnit en Eclipse (3)



VIDEO DE INTERÉS

Tutorial de JUnit en Eclipse:

https://www.youtube.com/watch?v=ql0BX4hq6D4

5.2 Crear clase Suma

La siguiente acción será crear una nueva clase que va a ser la encargada de realizar la suma. Aunque en estos momentos no se tenga mucho conocimiento sobre los conceptos de clase y objeto (programación orientada a objetos), no es necesario saber más allá de lo propuesto en esta guía de ejemplo. Para crear la clase Suma, el procedimiento es File → New → Class → en Name escribir Suma:

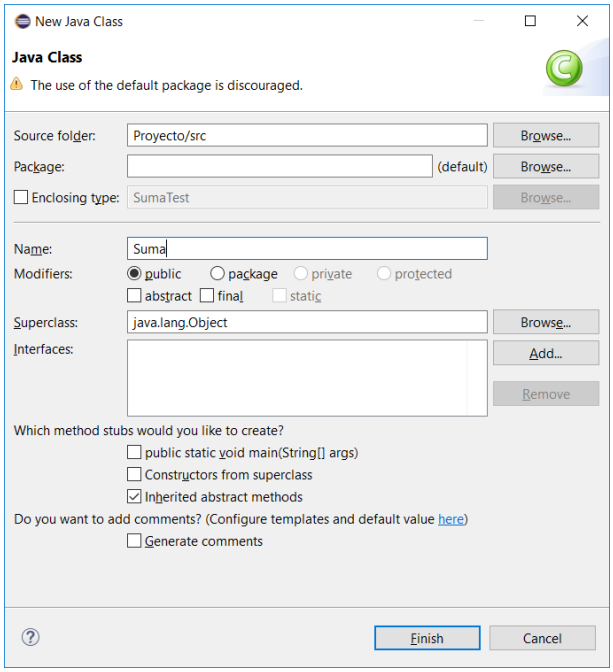


Imagen: Crear clase para pruebas JUnit en Eclipse (1)

De forma automática se crea una clase (vacía) llama Suma.java dentro de la cual habrá que añadirle el siguiente código:

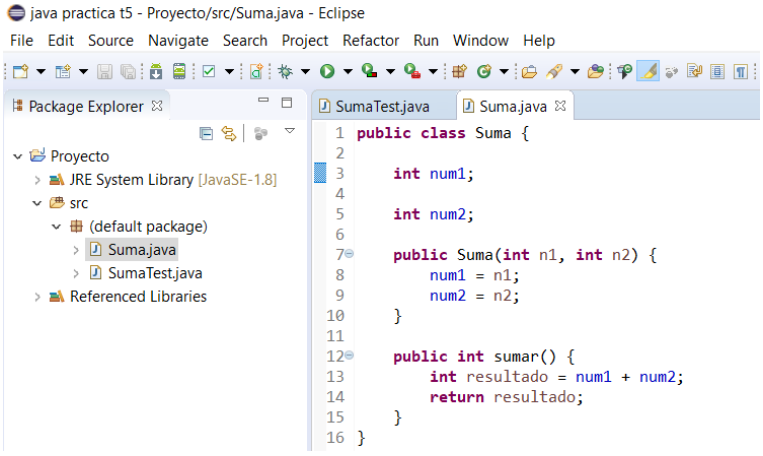


Imagen: Crear clase para pruebas JUnit en Eclipse (2)

Es importante que el código respete todos y cada uno de los caracteres, ya que cualquier error no permitirá su ejecución. Grosso modo, esta clase Suma.java presenta 2 variables (num1 y num2), por tanto, permitirá sumar 2 números de tipo entero (int). Posee 2 métodos: **Suma**, para la creación de objetos y **sumar**, que retorna la suma de num1+num2. Se insiste en la idea de que no es preciso conocer Java en estos momentos para realizar pruebas unitarias, simplemente seguir esta guía para realizar las pruebas unitarias con JUnit.

Una vez implementada la clase Suma.java, ya es posible realizar los casos de prueba en la clase SumaTest.java. Para ello, se procede a visualizar esta clase y programar en ella las siguientes líneas de código:

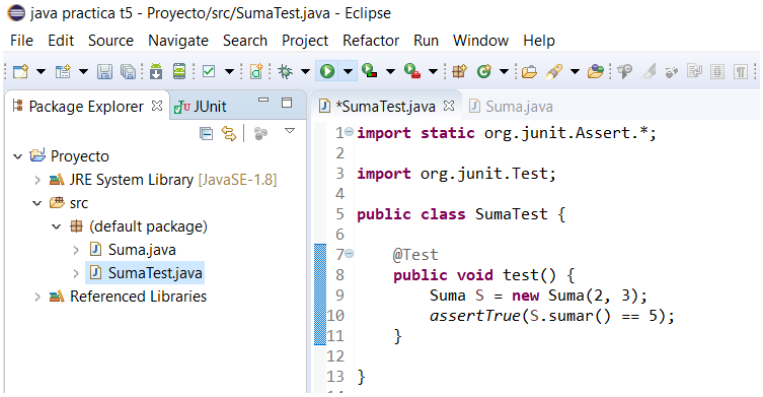


Imagen: Crear clase para pruebas JUnit en Eclipse (3)

Como se puede comprobar, únicamente se han añadido 2 líneas:

```
Suma S = new Suma(2, 3);
assertTrue(S.sumar() == 5);
```

La primera de ellas va a permitir crear un objeto de la clase Suma, otorgando un valor "2" a **num1** y un valor "3" a **num2**. La segunda línea va a realizar una llamada al método sumar, con esos valores, estimando que la suma va a dar como resultado el valor "5". Con estos valores de prueba introducidos, ¿qué resultado se obtiene de forma automática con JUnit? Bastará con ejecutar la clase SumaTest.java (botón play) y observar resultado esperado válido (verde):

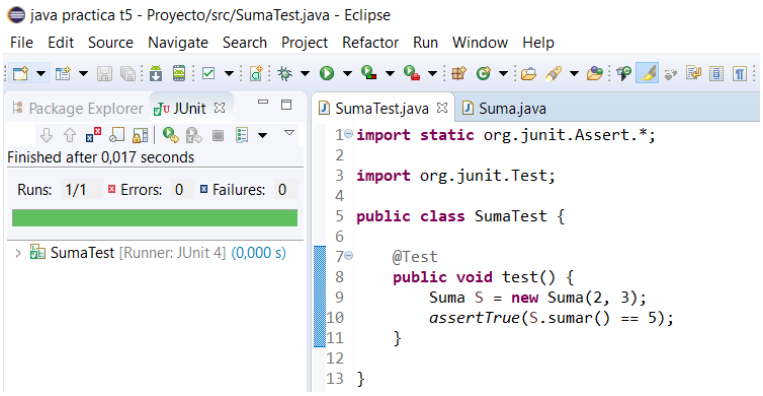


Imagen: Crear clase para pruebas JUnit en Eclipse (4)

Cambiando cualquier valor no esperado, por ejemplo, si el caso de prueba consiste en sumar 4 y 7, manteniendo el resultado anterior, la validación JUnit dictará que el resultado obtenido no es el esperado, ya que 5 ≠ 11 (marrón).

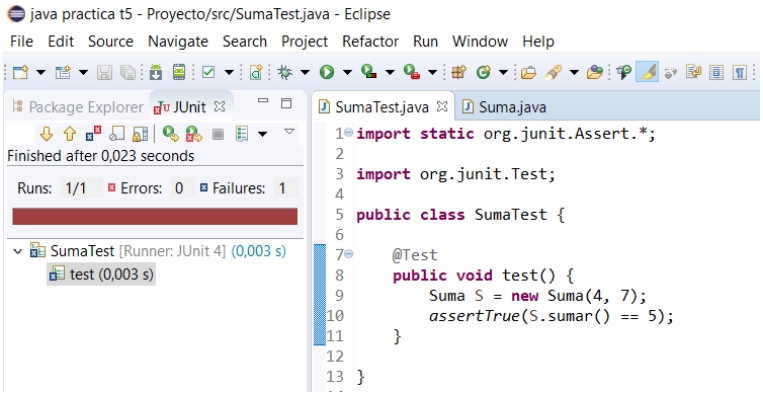


Imagen: Crear clase para pruebas JUnit en Eclipse (5)

RESUMEN FINAL

En esta unidad se ha destacado la importancia de las pruebas en el desarrollo del software. En la fase de pruebas, es importante establecer un plan de pruebas y una estrategia para ser llevado a cabo. El equipo de pruebas será el encargado de realizar las diferentes pruebas a realizar, tales como las pruebas de unidad, de integración, de sistema y de validación. Será en el entorno del cliente donde se realizarán las de aceptación, momento en el cual el cliente aceptará o no el software desarrollado, en función del cumplimiento de los requisitos establecidos al comienzo del mismo.

Como proceso interesante, en esta unidad se destaca la gran labor que tiene la automatización de las pruebas a través de las herramientas que la soportan. Gracias a ella, los casos de prueba, procedimientos de prueba y componentes de prueba se van a poder ejecutar y validar de forma automática. Existen una serie de pruebas que podrían encuadrarse dentro de las pruebas de sistema, llamadas pruebas de rendimiento, que van a permitir llevar las aplicaciones al límite de su carga y medir el comportamiento en base a unos parámetros de rendimiento, como por ejemplo el consumo de memoria, el procesamiento y los tiempos de ejecución, entre otros. Estas pruebas de rendimiento no serían posibles sin la automatización, ya que se llevan a cabo a través de software específico capaz de ejecutar acciones estresantes para el aplicativo y el servidor que lo contiene. Por ejemplo, mil usuarios concurrentes realizando un login o un logout durante una hora, o doscientos usuarios concurrentes realizando transferencias bancarias sin descanso en un periodo constante de más de 5 horas.

Por último, destacar en la unidad se destaca la relación existente entre la calidad del software y las pruebas, tomando como ejemplo una serie de normativas de calidad que proporcionan características y subcaracterísticas de calidad medibles (ISO 9126, ISO 25000, etc.). En este sentido, cuanto mayor probado esté un software, mayor número de defectos se habrán encontrado, con lo que se habrá trabajado en solventarlos y el producto estará (casi) libre de errores. Este hecho le otorgará un nivel de calidad bastante aceptable.