U5. Diseño Y Realización De Pruebas.

Todo software debe ser probado una vez ha sido terminado. Esta fase es fundamental antes de entregar el producto al cliente para su explotación, aunque, dependiendo de la metodología elegida para el trabajo, las pruebas son una constante en todo proceso de desarrollo.

5.1 Pruebas en el desarrollo de software.

Necesidad de planificación: Para probar un software es necesario tener un plan de pruebas donde se realizará la programación a los que se someterá a la aplicación y la estimación de recursos que se tendrán en cuenta antes de perpetrar el proceso.

La planificación es parte de la documentación de todo proyecto de software, donde se establecerán ya desde un inicio el tipo y la forma en que se llevarán a cabo. Asimismo, se debe contar con los recursos personales, y de hardware y con el tiempo para su realización.

Plan de Pruebas.

Documentación de todo proyecto de software, donde se establecerán ya desde un inicio el tipo y la forma en que se llevarán a cabo. Partes más importantes:

- Descripción de cada una de las fases del proceso de pruebas.
- Requisitos para asegurar que se prueban todos los requisitos.
- Elementos y cronograma a ser probado.

Para que el plan de pruebas sea efectivo es fundamental las siguientes partes:

- Registro de la prueba bien documentada.
- Requisitos de hardware y software.
- Plan de contingencia.
- Definir los casos de pruebas que deben aplicarse al sistema.

Nº	Tester	Fecha	Objetivo de Prueba	Resultado esperado	Resultado obtenido	Conclusión
1	JCV	11/04/19	Usuario y *password* de usuario válido. Pulsar *Aceptar*.	El usuario entra en el aplicativo.	El usuario entra en el aplicativo.	Prueba ☑
2	JCV	11/04/19	Usuario y *password* de usuario no válido. Pulsar *Aceptar*.	El usuario no entra en el aplicativo. Se muestra mensaje de usuario no válido.	El usuario no entra en el aplicativo. Se muestra mensaje de usuario no válido.	Prueba ☑
3	JCV	11/04/19	Usuario válido y *password* incorrecta. Pulsar *Aceptar*.	El usuario no entra en el aplicativo. Se muestra mensaje de *password* incorrecta.	El usuario no entra en el aplicativo. Se muestra mensaje de *password* incorrecta.	Prueba ☑
4	JCV	11/04/19	Usuario añadido y campo *password* vacío. Pulsar *Aceptar*.	El usuario no entra en el aplicativo. Se muestra mensaje de *password* vacío.	El usuario no entra en el aplicativo. Se muestra mensaje de *password* incorrecta.	Prueba ☒

PROCESO DE PRUEBAS: VALIDACIÓN vs VERIFICACIÓN.

Validación: Es el proceso que comprueba que el sistema se esté construyendo de la forma adecuada.

Verificación: Se refiere a si el sistema está bien diseñado y sin errores de funcionamiento.

5.2. Tipos de pruebas.

5.2.1. PRUEBA ESTÁTICA.

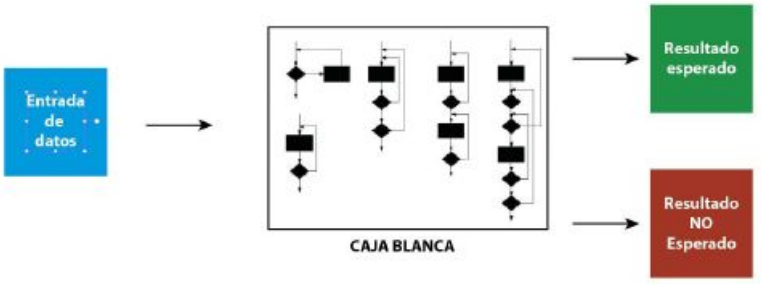
Pruebas de inspección de requisitos, prototipos o errores en el código fuente. Buscando errores.

5.2.2. PRUEBA DINÁMICA.

Se comprueba el comportamiento del código generado en tiempo de ejecución.

5.2.3. PRUEBAS ESTRUCTURADAS.

Las pruebas estructuradas tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado.



Las pruebas de software se dividen, de forma general, en dos grupos:

Pruebas de caja negra. Son aquellas en las que únicamente se tiene en cuenta la entrada y la salida de información. Son pruebas funcionales y no validan la implementación software. Por naturaleza de su funcionalidad, son de alto nivel (no es necesario conocer su código).

Pruebas de caja blanca. A diferencia de las anteriores, toman la información de entrada, observan cómo se encuentra desarrollado el código y si cumple con los requisitos establecidos y se observa la información de salida. En Java existe una herramienta que ayuda a su implementación, llamada **JUnit**. Por la esencia de su funcionalidad, son de bajo nivel (es necesario observar su implementación).

5.4. PRUEBAS DE INTEGRACIÓN.

Es muy común, que el desarrollo de un proyecto software se lleve a cabo mediante la implementación de diferentes módulos. Esto se realiza de esta forma paralela ya que los miembros de un equipo de desarrollo suelen repartirse las tareas para trabajar de forma simultánea y tener un mayor aprovechamiento del tiempo empleado.

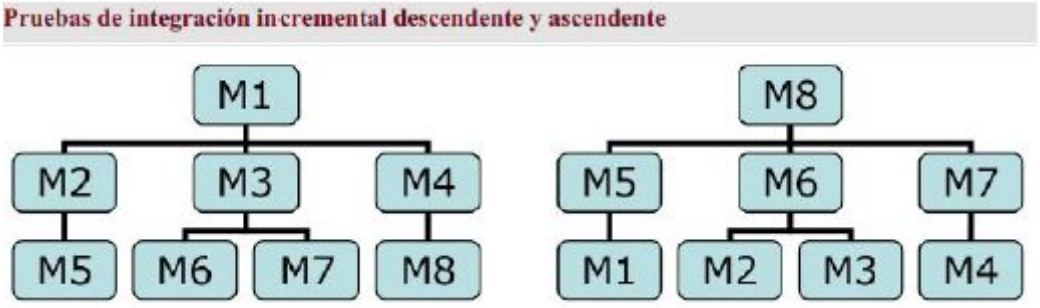
Todos estos módulos deben ser probados de forma individual a través de pruebas de caja negra (funcionales) y de caja blanca (unitarias - Junit).

TIPOS DE PRUEBAS. Para integrar los módulos ha de existir una comunicación e interacción entre ellos. Existen fundamentalmente **dos tipos de pruebas de integración**:

1. Integración incremental. Al añadir un nuevo módulo, se deben probar los que ya existen y han sido probados anteriormente, con el objetivo de comprobar si esta nueva integración ha provocado algún tipo de cambio en el comportamiento de la aplicación. La integración incremental puede realizarse, a su vez, de dos formas:

1. **Descendente**, si se parte de un módulo o módulos padre y se avanza hasta probar los módulos hijos.
2. **Ascendente**, si se parte de un módulo o módulos hijos hasta probar el módulo o módulos padre.

2. Integración no incremental. A diferencia del anterior, se integran todos y cada uno de los módulos que forman parte del aplicativo y, una vez acoplados, se comprueba la funcionalidad completa del programa a través de todos los módulos.



5.5. PRUEBAS DE RENDIMIENTO: Tiempos de respuesta, uso de recursos

Las pruebas de volumen y de estrés se encuadran dentro de las conocidas pruebas de rendimiento del software.

Objetivo: Encontrar el punto de rendimiento máximo del equipo y del sistema donde se ejecuta el software a través de ejecuciones concurrentes y crecientes de la aplicación.

1. PRUEBAS DE VOLUMEN. Dada una aplicación se incrementa progresivamente el número de usuario/peticiones. Este proceso de repetiré durante un tiempo hasta el sistema de corrompa/finalice.
2. PRUEBAS DE ESTRÉS. A diferencia de la prueba de volumen, consiste en ejecutar el script con un número máximo de usuarios/peticiones y la prueba continuará con ese número tope de usuarios hasta finalizar el tiempo estipulado.

5.2.6. PRUEBAS DE USABILIDAD Y ACCESIBILIDAD.

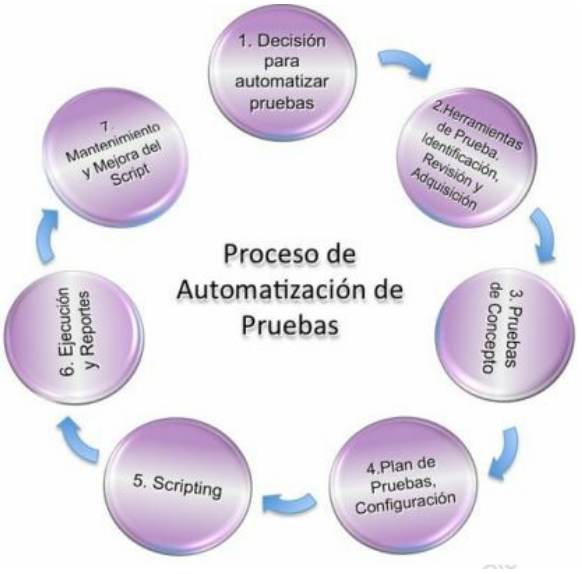
5.2.7. PRUEBAS DE SEGURIDAD.

Las pruebas de seguridad tienen como objetivo, por un lado, asegurar el acceso a la aplicación a aquellas personas autorizadas y denegar el acceso a las no autorizadas y, por otro lado, permitir que cada usuario autorizado dentro del aplicativo pueda realizar únicamente las acciones que les concedan los permisos que tenga asignados.

5.3. PRUEBAS DE CÓDIGO.

Herramientas de testeo.

La fase de pruebas en todo proyecto de software es un proceso laborioso. Es obvio que contar con herramientas que nos ayuden a automatizar las tareas donde configurando parámetros podamos construir una serie de pasos que nuestros códigos deban superar.



Para automatizar las pruebas tenemos muchas herramientas, cada una con funcionalidades diferentes. Vamos a clasificarlas según su objetivo**:**

1. **Gestor de pruebas**

Este tipo de aplicaciones es una de las más usadas por los desarrolladores. Normalmente son

frameworks de los propios lenguajes o herramientas integradas en los IDE, como el caso de JUnit.

2. **Analizador dinámico**
3. **Oráculo**
4. **Comparador de ficheros**
5. **Generador de informes**
6. **Generador de datos de pruebas**

5.4. Automatización de pruebas

Herramientas podemos agilizar el trabajo de la realización de las pruebas, que pueden ser de dos tipos:

1. Pruebas codificadas: Se automatizan las pruebas unitarias utilizando casos de uso.
2. Pruebas de usuario: Se graban acciones que realizarán los usuarios sobre la interfaz de la aplicación a evaluar. Con esto se consigue que las acciones repetitivas puedan ser ejecutadas las veces que haga falta.

Existen entornos de pruebas dependiendo del lenguaje de programación: JUnit, TestNG, JTiger (JAVA)... PHPUnit, CPPUnit.. (Otros lenguajes).

Pruebas Unitarias con JUNIT.

Las pruebas unitarias (pruebas de caja blanca) comprueban si una determinada funcionalidad indivisible cumple con los requisitos establecidos. Estas pruebas pueden realizarse:

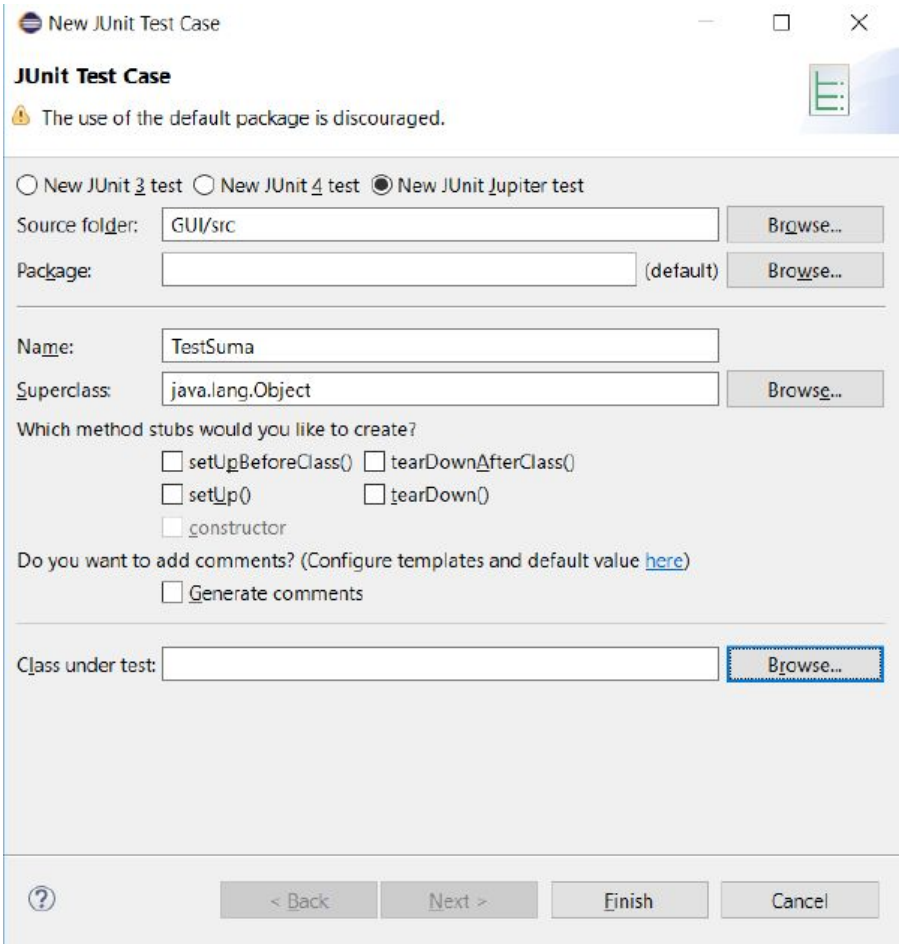
- **De forma manual** (introduciendo los valores correspondientes para cada caso).
- **De forma automática**; para esto, Java presenta una herramienta llamada **JUnit**.

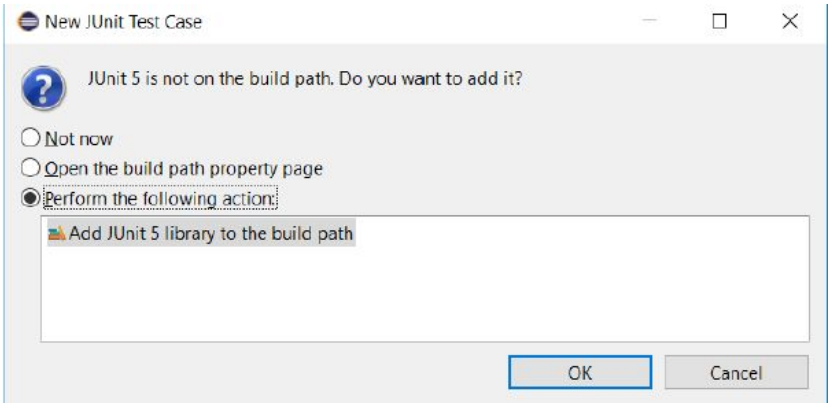
En un formulario tipo, se pueden realizar algunas pruebas unitarias, por ejemplo:

- Pulsar sobre el botón Cancelar, esperando que el resultado sea salir de esta interfaz.
- Comprobar la funcionalidad del elemento radioButton, es decir, si al seleccionar una u otra acción
- Hacer clic en el botón Limpiar campo opinión y comprobar si realmente deja el área de texto nuevamente sin texto.
- La comprobación de contraseñas y muestra un mensaje aclaratorio en consecuencia al pulsar Aceptar.

5.4. PRUEBAS UNITARIAS con JUNIT.

File > New > JUnit test case.





La herramienta JUnit genera una clase de forma automática llamada TestSuma, en la que se debe implementar la prueba unitaria. Para este caso, se va a realizar una

prueba para comprobar si realiza bien o no una suma de dos números decimales. El código implementado es el siguiente:

```
public class Suma {

        public double getSuma(double d, double e) {
                // TODO Auto-generated method stub
                return d+e;
        }
}
```

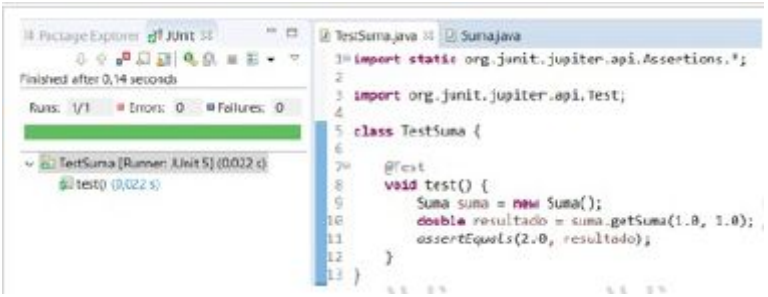
```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class TestSuma {

        @Test
        void test() {
                Suma suma = new Suma();
                double resultado = suma.getSuma(1.0, 1.0);
                assertEquals(2.0, resultado);
        }
}
```

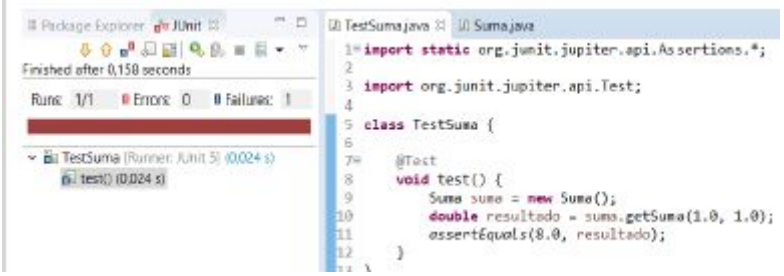
La esencia de JUnit aparece en la llamada al método AssertEquals, el cual toma como parámetros el resultado que se espera y el resultado obtenido.

Para el caso del ejemplo actual, ambos coinciden, por tanto, JUnit dará la prueba como correcta y lo reflejará en color verde, como se muestra en la figura:



Para el caso en el que el resultado esperado no coincida con el resultado obtenido, el IDE Eclipse a través de JUnit lo mostraría en color marrón (erróneo).

Ej) assertEquals(8.0,resultado);



Ampliando el número de pruebas, sobre la misma clase Suma.

```
@Test
        public void test_ok_1 () {
                Suma suma = new Suma();
                double resultado = suma.getSuma(1.0, 1.0);
                assertEquals(2.0, resultado);
        }

// Test unitario cuyo resultado saldrá satisfactorio

        @Test
        public void test_ok_2 () {
                Suma suma = new Suma();
                double resultado = suma.getSuma(1000.0, 1000.0);
                assertEquals(2000.0, resultado);
        }

// Test unitario cuyo resultado saldrá erróneo

        @Test
        public void test_fail () {

                Suma suma = new Suma();
                assertNotNull(suma);
                double resultado = suma.getSuma(1.0, 1.0);
                assertNotEquals(8.0, resultado);
        }
}
```

RESULTADO: