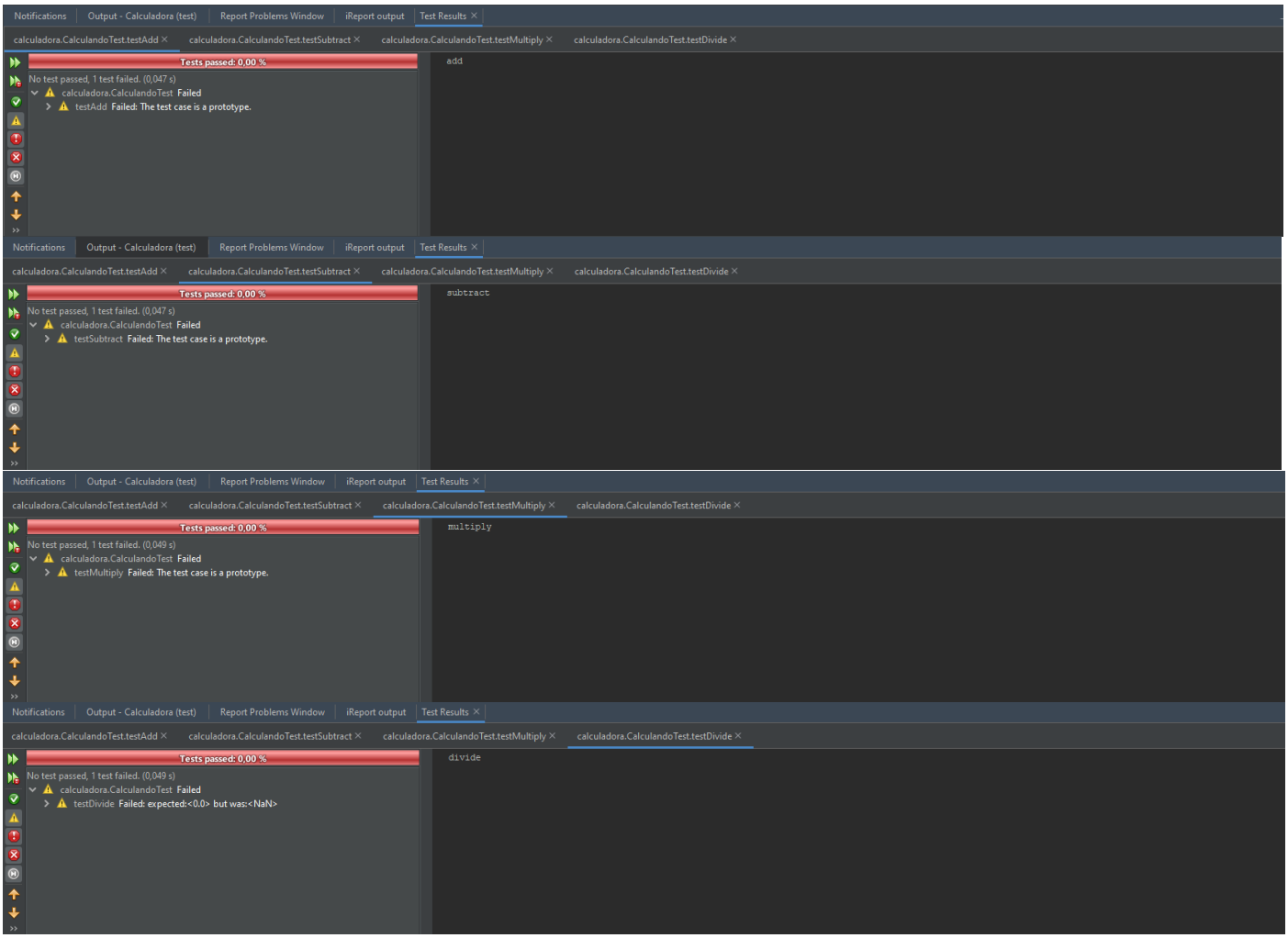
TAREA 08 DESARROLLO DE INTERFACES

1. Más abajo puedes descargar el Proyecto Java, ábrelo con NetBeans. Observa los métodos definidos en la clase Calculando.java. Vamos a probar cada método de la clase con JUnit. Para ello, deberás de seleccionar la clase y en el menú Herramientas deberás de seleccionar la opción Create /update Tests. Nos aparecerá una ventana donde consta la clase a la que se le van a realizar las pruebas y la ubicación de las mismas. Seleccionaremos como Framework Junit y veremos que el código de la aplicación importa automáticamente el framework Junit. Como solución a este apartado deberás de aportar el código de la clase generado. (Calificación 1 punto)

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/UnitTests/JUnit3TestClass.java to edit this template
 */
package calculadora;

import junit.framework.TestCase;

/**
 *
 * @author hfkra
 */
public class CalculandoTest extends TestCase {

    public CalculandoTest(String testName) {
        super(testName);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /**
     * Test of add method, of class Calculando.
     */
    public void testAdd() {
        System.out.println("add");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.add(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    /**
     * Test of subtract method, of class Calculando.
     */
    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    /**
     * Test of multiply method, of class Calculando.
     */
    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.multiply(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

    /**
     * Test of divide method, of class Calculando.
     */
    public void testDivide() {
        System.out.println("divide");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.divide(number1, number2);
        assertEquals(expResult, result, 0.0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

}
```

2. Selecciona la nueva clase de pruebas que has generado. Ejecútala. Realiza una captura de la ventana Test results como solución a este aparado. (Calificación 1 punto)

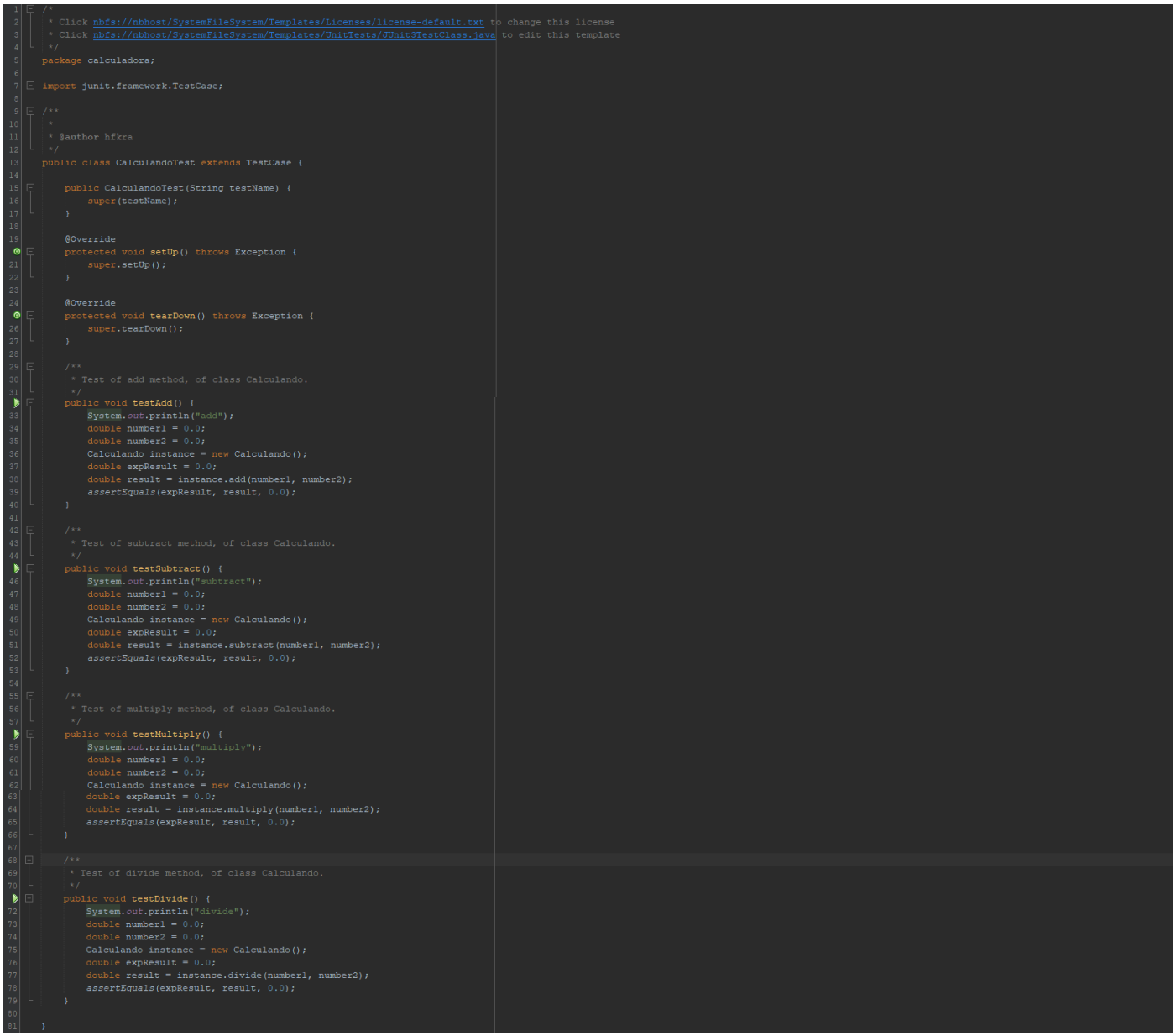


3. Accede al código de la clase de pruebas y elimina las líneas:

// TODO review the generated test code and remove the default call to fail.

fail ("The test case is a prototype.");

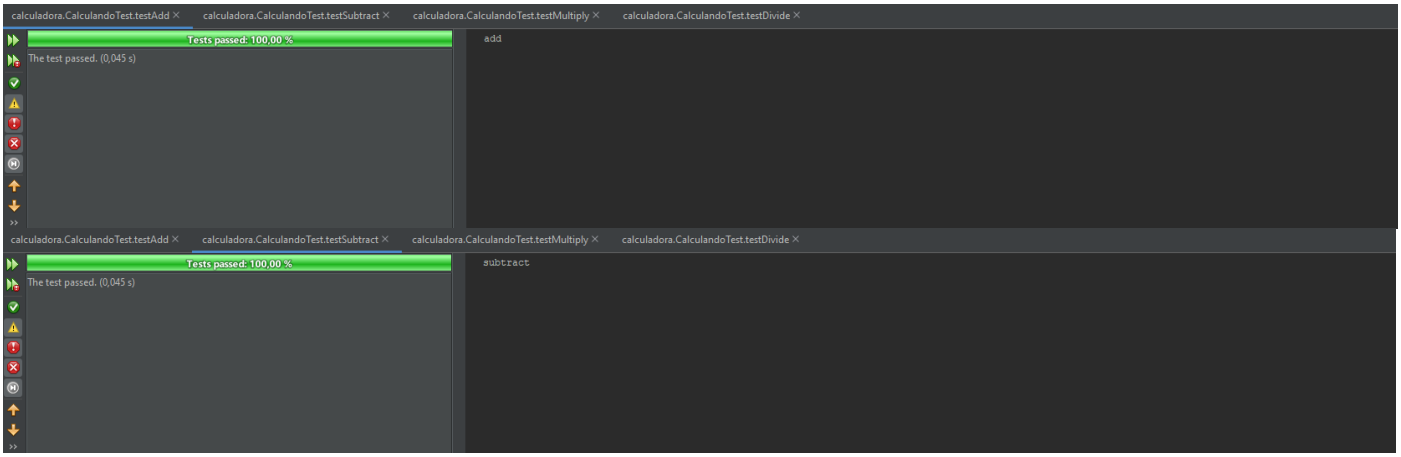
que aparece al final de cada método. Como solución a este apartado deberás de entregar el código de la clase generado. (Calificación 1 punto)

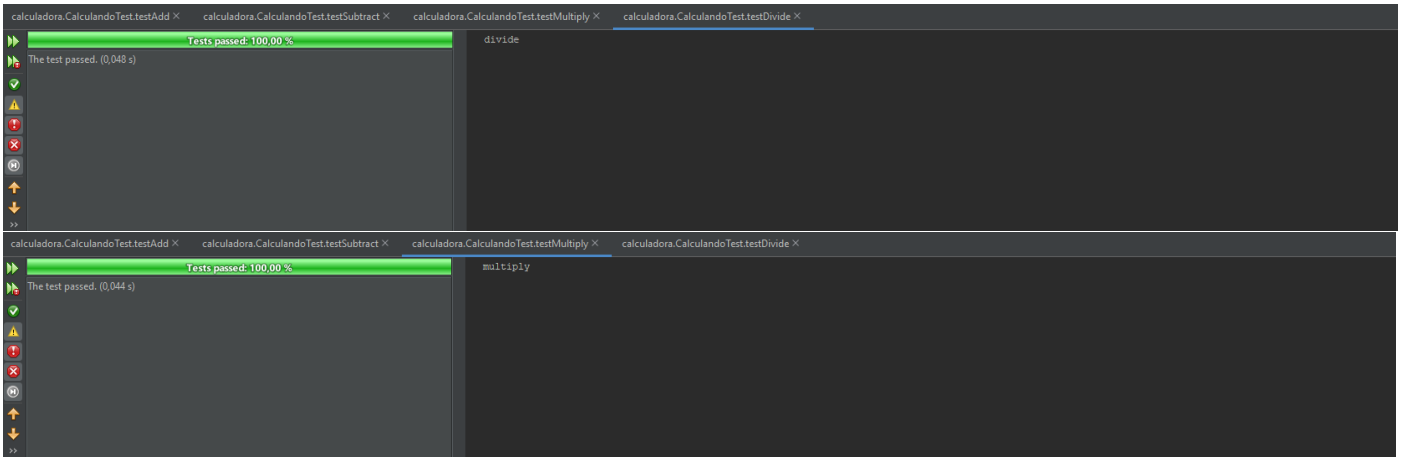


```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/UnitTests/JUnit3TestClass.java to edit this template
 */
package calculadora;

import junit.framework.TestCase;

/**
 *
 * @author hfkra
 */
public class CalculandoTest extends TestCase {

    public CalculandoTest(String testName) {
        super(testName);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /**
     * Test of add method, of class Calculando.
     */
    public void testAdd() {
        System.out.println("add");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.add(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of subtract method, of class Calculando.
     */
    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of multiply method, of class Calculando.
     */
    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.multiply(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of divide method, of class Calculando.
     */
    public void testDivide() {
        System.out.println("divide");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculando instance = new Calculando();
        double expResult = 0.0;
        double result = instance.divide(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

}
```

4-. Selecciona la clase de prueba y ejecútala de nuevo. Debes de corregir todos los errores asignándole valores a las variables. Al final, debes de conseguir que la ejecución de la prueba sea satisfactoria. Como solución a este apartado deberás de aportar el código de la clase de prueba una vez que ha sido modificado para conseguir que las pruebas fueran satisfactorias. (Calificación 1 punto)

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/UnitTests/JUnit3TestClass.java to edit this template
 */
package calculadora;

import junit.framework.TestCase;

/**
 *
 * @author hfkra
 */
public class CalculandoTest extends TestCase {

    public CalculandoTest(String testName) {
        super(testName);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
    }

    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /**
     * Test of add method, of class Calculando.
     */
    public void testAdd() {
        System.out.println("add");
        double number1 = 4.0;
        double number2 = 8.0;
        Calculando instance = new Calculando();
        double expResult = 12.0;
        double result = instance.add(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of subtract method, of class Calculando.
     */
    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 7.0;
        double number2 = 10.0;
        Calculando instance = new Calculando();
        double expResult = -3.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of multiply method, of class Calculando.
     */
    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 10.0;
        double number2 = 3.0;
        Calculando instance = new Calculando();
        double expResult = 30.0;
        double result = instance.multiply(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of divide method, of class Calculando.
     */
    public void testDivide() {
        System.out.println("divide");
        double number1 = 35.0;
        double number2 = 7.0;
        Calculando instance = new Calculando();
        double expResult = 5.0;
        double result = instance.divide(number1, number2);
        assertEquals(expResult, result, 0.0);
    }

}
```





5-. Implementa la planificación de las pruebas de integración, sistema y regresión. (Calificación 2 puntos)

Pruebas de Integración: verifica la interacción entre los componentes del sistema. En este caso en concreto el proyecto sólo consta de un módulo con 4 métodos, las pruebas de integración habría que realizarlas con más de 1 modulo, para comprobar que al unir esos módulos que en los test de las pruebas unitarias nos has salido favorables no producen una respuesta indeseada por el hecho de unirlos. Después de leer el tema y sin tener más información, yo me decantaría por diseñar una planificacion de abajo a arriba, probando a unir los módulos de uno en uno y una vez que esos no dan errores de integración, añadiría otro módulo, y así, sucesivamente hasta integrar la totalidad de los módulos. Es cierto que tienes mayor incertidumbre, pero para empezar me parece la metodología más cómoda de aplicar.

Pruebas de Sistema: comprueba los requisitos no funcionales de la aplicación: seguridad, velocidad, exactitud, fiabilidad. Actualmente estas pruebas se simulan en equipos con máquinas virtuales como si fueran hechas en varios equipos físicos distintos. El objetivo es hacer que la aplicación falle, para poder identificar los defectos y arreglarlos para cumplir con los requerimientos. Tambien habrá que hacer unas pruebas de recuperación, forzando el fallo del software y verificando que la recuperacion se lleva a cabo de forma satisfactoria. Para economizar el tiempo y reducir costes, las haremos de forma automática, evaluando la corrección de la inicialización, los mecanismos de recuperación del estado del sistema, los datos y el proceso de arranque. Para finalizar, podemos hacer un analisi manual comprobando que los tiempos medios de reparacion están dentro de unos limites aceptables.

Pruebas de Regresión: consiste en volver a ejecutar un subconjunto de pruebas que ya se ha llevado a cabo anteriormente para asegurarnos de que los cambios efectuados en nuestro software no han producido algun efecto no deseado. En este caso, las pruebas se podrían hacer manualmente, simplemente repitiendo las pruebas que ya nos han salido positivas al ejecutar la union de varios modulos tambien al añadir otro que cambie el total de la aplicación integrada. En la actualidad, se utilizan herramientas que permiten detectar este tipo de errores de manera automatizada.

6-. Planifica las restantes pruebas, estableciendo qué parámetros se van a analizar. (Calificación 2 puntos)

Pruebas Funcionales: validan si el comportamiento observado del software es conforme con sus requisitos funcionales. Básicamente se tratan de pruebas en las que se comprueba si la aplicación responde al motivo por ql que ha sido creada. Yo planificaría estas pruebas haciendo un analisis de los valores limite siempre que sea posible más una ronda de pruebas aleatorias si es posible con miembros distintos. Por ejemplo, el caso del método **add**, al devolver un double, pobraría una suma superior a su rango, una suma positiva, una suma valor 0, una suma negativa y una suma inferior a su rango. Controlaría las posibles excepciones y probaria números aleatorios para terminar de corroborar que todo funciona como se espera.

Pruebas de Capacidad: determinar hasta donde es capaz de llegar el sistema en condiciones extremas. Se podría aumentar la frecuencia de las entradas, abrir muchas veces el programa…

Pruebas de Rendimiento: determinar el rendimiento del programa en tiempo de ejecucion dentro de un contexto de un sistema integrado. Son pruebas que se han de realizar durante todos los pasos del proceso de la prueba. Requieren de instrumentación para la monitorización donde podemos comprobar los tiempos de respuesta de las tareas, el espacio que ocupan los módulos en el disco, lo que ocupan en la memoria, el flujo de datos que se genera…etc.

Pruebas de Eficiencia: comprueban que se hace un uso eficiente de los recursos de los que se dispone. El objetivo es buscar una buena adaptación del programa a la arquitectura de destino. El software tendrá que ser probado e implementado en distintos sistemas concretos para comprobar que no tiene problemas de optimización en ninguna estructura concreta, y si los hubiera, tratar de solucionarlos para lograr su adaptación.

Prueba de Usuario: sirve para asegurar que la interfaz de la aplicación sea amigable, intuitiva y que funcione correctamente. Las pruebas se basan en hacer que una ronda de usuarios que no han formado parte del desarrollo del software, realicen una serie de tareas y comprobemos si tienen facilidad de uso y si el cliente sale satisfecho de su interacción. Para comprobar la calidad del programa podemos medir el tiempo que tarda en realizar las tareas, el número de clicks o los errores que comete durante el proceso. Para ayudar a completar esta prueba, podemos recurrir a las entrevistas o a encuestas para extraer toda la información que nos pueda ser util para mejorar nuestra aplicación.

Pruebas de Aceptación: una vez que hemos superado satisfactoriamentes todas las pruebas anteriores y tenemos una versión lo bastante pulida para mostrársela al cliente final, hacemos una revisión de la aplicación con el cliente para que pueda comprobar si cumple con los requerimientos iniciales y que funciona con lo que se esperaba de ella. Hay 2 tecnicas en este tipo de prueba:

- **Alfa:** se trae al cliente a nuestro lugar de trabajo, donde hemos desarrollado el software. El cliente hará uso de él como si se estuviera en su lugar de trabajo, pero con el desarrollador delante, para guiarle y ayudarle en el proceso.
- **Beta:** el cliente, ya en su lugar de trabajo, hace uso de la aplicación sin tener al desarrollador presente. Este registrará todos los problemas que vaya encontrando según la vaya usando, para informar al desarrollador de estos para que realice las correcciones oportunas. El desarrollador preparará una nueva versión de software con todos los problemas encontrados durante esta fase de pruebas.

7-. Supuestas exitosas las pruebas, documenta el resultado (Calificación 2 puntos).

La documentacion se utiliza para el desarrollo del producto y, sobre todo, para su mantenimiento futuro. Se documenta para comunicar estructura y comportamiento del sistema, visualizar y controlar la arquitectura del sistema, comprenderlo mejor y controlar el riego… Cuanto más complejo es el sistema, más importante es la documentación.

Todas las fases de un desarrollo deben documentarse: requerimientos, analisis, diseño, programación, pruebas… Un aspecto a tener en cuenta cuando se documenta es el nivel de detalle. Al documentar el software debemos cuidar el nivel de detalle.

Nos interesa especialmente la documentación que se usa para documentar la programación, y en particular la que hemos ido desarrollando durante las fases de prueba para que quede constancia de los procesos y cambios realizados durante cada una de las fases.