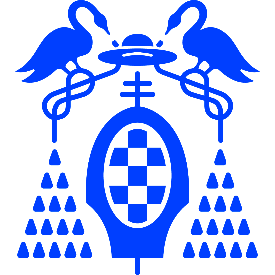
Trabajo en grupo 3: automated testing frameworks for php

Grupo T8

Universidad de Alcalá

Desarrollo con Tecnologías Emergentes



Daniel Manzano Estébanez

Inés López Baldominos

José María Sanz Górriz

Adrián de la Hoz Casanova

Cristian Abellán Madrigal

Contenido

[1. Autores del trabajo, planificación y entrega 2](#_Toc8075633)

[1.1 Autores 2](#_Toc8075634)

[1.2 Planificación 2](#_Toc8075635)

[1.3 Entrega 2](#_Toc8075636)

[2. Requisitos del prototipo a implementar 3](#_Toc8075637)

[2.1 Requisitos funcionales 3](#_Toc8075638)

[2.2 Otros requisitos 3](#_Toc8075639)

[3. Criterios de comparación en la implementación 4](#_Toc8075640)

[3.1 Criterio 1: Dificultad en obtener e instalar el software necesario 4](#_Toc8075641)

[3.2 Criterio 2: Recursos necesarios para el funcionamiento de las herramientas de testeo automático 4](#_Toc8075642)

[3.3 Criterio 3: Tiempo de respuesta funcionamiento del sistema 4](#_Toc8075643)

[3.4 Criterio 4: Información para utilizar la herramienta 4](#_Toc8075644)

[3.5 Criterio 5: Implementación intuitiva 4](#_Toc8075645)

[4. Proyecto de implementación de un prototipo del sistema utilizando PHPUnit 5](#_Toc8075646)

[4.1 Documentación de diseño 5](#_Toc8075647)

[4.2 Documentación de construcción 5](#_Toc8075648)

[4.3 Documentación de pruebas 7](#_Toc8075649)

[4.4 Documentación de instalación 9](#_Toc8075650)

[4.5 Manual de usuario 10](#_Toc8075651)

[5. Proyecto de implementación de un prototipo del sistema utilizando Codeception 11](#_Toc8075652)

[5.1 Documentación de diseño 11](#_Toc8075653)

[5.2 Documentación de construcción 11](#_Toc8075654)

[5.3 Documentación de pruebas 13](#_Toc8075655)

[5.4 Documentación de instalación 14](#_Toc8075656)

[5.5 Manual de usuario 16](#_Toc8075657)

[6. Comparación de las dos implementaciones 17](#_Toc8075658)

[6.1 Evaluación de los criterios en la implementación usando PHPUnit 17](#_Toc8075659)

[6.2 Evaluación de los criterios en la implementación usando Codeception 17](#_Toc8075660)

[7. Comparación de la implementación de las tecnologías 18](#_Toc8075661)

[8. Conclusiones 19](#_Toc8075662)

# 1. Autores del trabajo, planificación y entrega

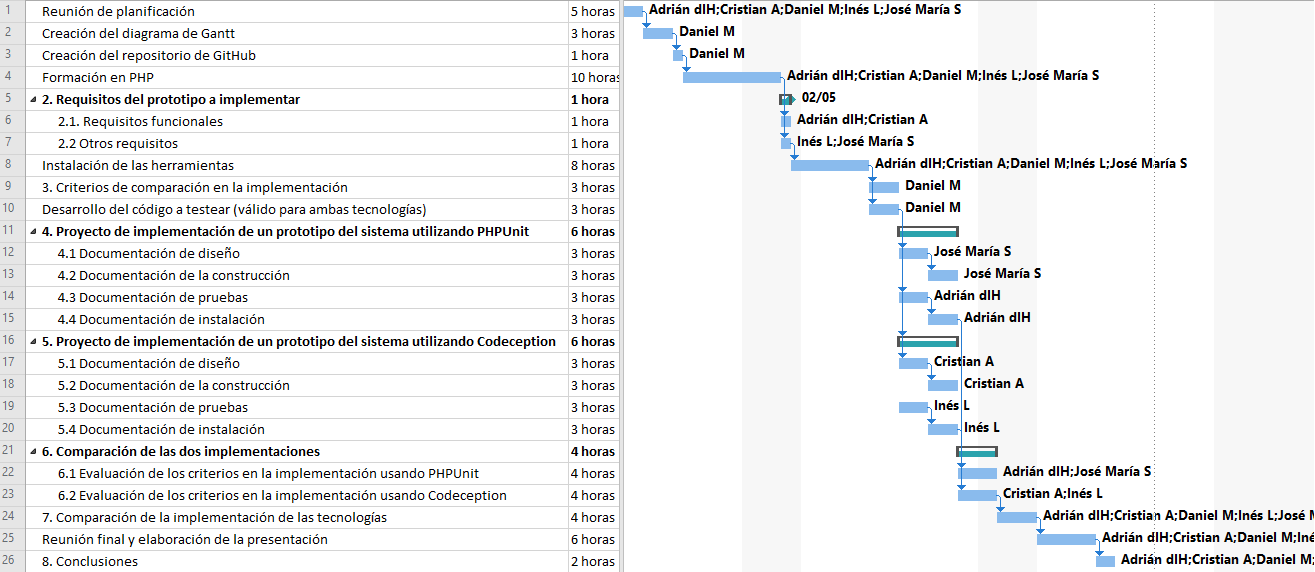
## 1.1 Autores

Grupo T8

* Daniel Manzano Estébanez (Coordinador)
* Cristián Abellán Madrigal
* Inés López Baldominos
* José María Sanz Górriz
* Adrián de la Hoz Casanova

## 1.2 Planificación

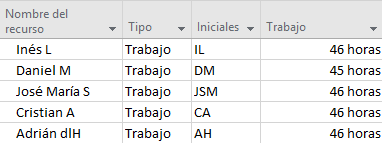
Con la ayuda de Microsoft Project, se ha elaborado un diagrama de Gantt en el que se han registrado las diferentes tareas de este trabajo y su duración, que han sido asignadas a los diferentes miembros del grupo. Este archivo se encuentra en el repositorio de GitHub que se indica en el apartado 1.3. Esta es la planificación realizada.

**

*\* Las tareas 1, 4, 8, 24, 25 y 26 han sido llevadas a cabo por todos los miembros del equipo.*

Las tareas siguen la estructura de este trabajo y, además, se han añadido tareas correspondientes a la preparación de este, formación en PHP, instalación de las herramientas y la reunión de finalización.

Con el reparto de tareas, se ha logrado cumplir con las 45 horas exigidas para este trabajo, que se corresponde con el 30% de la asignatura. A continuación se puede ver a los miembros del grupo en Microsoft Project con el total de horas de cada uno.

**

## 1.3 Entrega

El trabajo realizado se encuentra disponible en la siguiente dirección de GitHub: <https://github.com/daniel-manzanoe/TG3>.

# 2. Requisitos del prototipo a implementar

Como objetivo de esta práctica, se desea comparar la implementación de tests automatizados utilizando las dos tecnologías que llevamos estudiando, documentando y analizando durante prácticas anteriores, que son Codeception y PHPUnit.

Como punto de partida, se ha decidido crear dos equipos de trabajo, por lo que uno de ellos se encargará de la parte de testing con la herramienta Coception y el otro equipo de la de la otra herramienta PHPUnit. Ambos equipos, cada uno en su herramienta, realizarán las mismas pruebas con el mismo programa que en este caso será NetBeans, sobre el mismo código PHP (una calculadora desarrollada también por el equipo) y a la vez teniendo en cuenta los mismos requisitos funcionales que se especificarán a continuación en el punto siguiente. En estos requisitos ambos equipos deben ceñirse a lo indicado para mantener una estructura similar para ambas herramientas.

En cambio, en cuanto a la parte de diseño, la libertad para ambos equipos es mayor que en la de requisitos funcionales, por lo que se les permitirá no deber continuar una estructura fijada, ni unas condiciones preestablecidas previamente.

## 2.1 Requisitos funcionales

Los requisitos funcionales deben ser los mismos para las dos implementaciones.

En la siguiente tabla se indicará el catálogo de requisitos funcionales del sistema:

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RF01 | Efectuar pruebas para la operación suma |
| RF02 | Efectuar pruebas para la operación resta | |
| RF03 | Efectuar pruebas para la operación multiplicación | |
| RF04 | Efectuar pruebas para dividir | |
| RF05 | Controlar las posibles excepciones |

## 2.2 Otros requisitos

Además de los requisitos anteriores, se tienen en cuenta otra serie de requisitos que se indican en la siguiente tabla:

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RI01 | La interfaz sobre la que se realizarán las pruebas será NetBeans |
| RD01 | Se probarán las herramientas sobre el código php disponible en el repositorio del grupo (calculadora) |
| RR01 | Los tests se realizarán sin tener ningún otro servicio en el PC, ya que podría afectar en los tiempos |

Glosario de términos:

RF: requisito funcional

RI: requisito de interfaz

RD: requisito de datos

RR: requisito de rendimiento

# 3. Criterios de comparación en la implementación

## 3.1 Criterio 1: Dificultad en obtener e instalar el software necesario

Nombre del criterio: Dificultad en obtener e instalar el software necesario

Descripción: Complejidad encontrada en el proceso de conseguir los programas necesarios e instalarlos para realizar los test automáticos en PHP.

Tipo de valor: Numérico (horas).

## 3.2 Criterio 2: Recursos necesarios para el funcionamiento de las herramientas de testeo automático

Nombre del criterio: Recursos necesarios para el funcionamiento de las herramientas de testeo automático.

Descripción: Cantidad de herramientas software que hemos tenido que instalar para poder realizar los test automáticos en PHP.

Tipo de valor: Numérico (cantidad de herramientas).

## 3.3 Criterio 3: Tiempo de respuesta funcionamiento del sistema

Nombre del criterio: Tiempo de respuesta del sistema

Descripción: Tiempo que tarda en realizar el testeo automático. Este criterio va a evaluarse en función del tiempo que ambas herramientas tardan en realizar el testeo de un mismo código.

Tipo de valor: Numérico (segundos).

## 3.4 Criterio 4: Información para utilizar la herramienta

Nombre del criterio: información para utilizar la herramienta

Descripción: Puntuación en función de la cantidad y calidad de la información encontrada para poner en marcha los test.

Tipo de valor: Numérico (1-5).

## 3.5 Criterio 5: Implementación intuitiva

Nombre del criterio: implementación intuitiva de los tests

Descripción: facilidad para utilizar las funciones de las herramientas, resolver los fallos producidos e interpretar los resultados.

Tipo de valor: Numérico (1-5).

# 4. Proyecto de implementación de un prototipo del sistema utilizando PHPUnit

A continuación, se describe toda la documentación necesaria para entender el desarrollo de los test realizados con PHPunit, las pruebas realizadas durante el proceso de implementación y la configuración requerida para poder probar el proyecto en cualquier otro equipo.

## 4.1 Documentación de diseño

Hemos creado en primer lugar un código en PHP de una calculadora con las funciones principales (suma, resta, multiplicación y división), la cual nos sirve para realizar el cometido de esta práctica, testing automático para PHP, y en base a ese código hemos realizado las pruebas con PHPUnit.

Hemos diseñado los siguientes casos de prueba para comprobar que cada una de las funciones realiza lo que se espera de ella. Comprobando con estos test que se realiza correctamente la suma, la resta, la multiplicación y la división, controlando en este último caso la división entre cero como una excepción.

* TestSumar
* TestRestar
* TestMultiplicar
* TestDividir

## 4.2 Documentación de construcción

* TestSumar

Para realizar la función de suma el prototipo que se ha seguido ha sido el siguiente uso de código PHP que suma los números a y b:

function sumar($a, $b) {

return ($a + $b);

}

Para realizar su testing, hemos cinco funciones que realizan una suma de dos números dados y comprueba que el dato introducido sea correcto al dado como se puede ver en el caso del siguiente ejemplo que suma 0+0 y como resultado tiene que proporcionar el valor de 0:

public function testSumar() {

$this->assertEquals(

0, $this->object->sumar(0, 0)

);

}

* TestRestar

Para realizar la función de restar dos números se ha utilizado el siguiente fragmento de código que resta el número a menos el número b:

function restar($a, $b) {

return ($a - $b);

}

En cuanto al testing de esta operación, se han elaborado cuatro casos en los que dados dos números se comprueba que al realizar la resta el resultado obtenido es el resultado deseado, como por ejemplo en el fragmento de código siguiente que resta 1-2 dando como resultado –1.

public function testRestar4() {

$this->assertEquals(

-1, $this->object->restar(1, 2)

);

}

* TestMultiplicar

La función de multiplicar se ha realizado usando el siguiente código que multiplica el primer valor a por el segundo valor b:

function multiplicar($a, $b) {

return ($a \* $b);

}

Para realizar el testing de la función de multiplicar, se han elaborado cuatro casos que realizan la propia multiplicación de dos números comprobando que su resultado es correcto, como puede observarse el en código siguiente que multiplicando 1x1 obtenemos el resultado 1.

public function testMultiplicar3() {

$this->assertEquals(

1, $this->object->multiplicar(1, 1)

);

}

* TestDividir

La última función de la calculadora es la división de dos números, teniendo en cuenta que el denominador de dicha división no puede ser 0, ya que en tal caso no sería posible realizar la operación y nos mostraría un error diciendo que la división por cero no es posible.

function dividir($a, $b) {

if ($b == 0) {

throw new \InvalidArgumentException("Division by zero is not possible");

}

return ($a / $b);

}

En cuanto al testing de esta última operación se han realizado un caso, incluyendo el posible error de dividir entre 0:

public function testDividir2() {

$this->object->dividir(1, 0);

}

Y realizando otros tres casos, observando que el resultado obtenido al dividir dos números introducidos es correcto:

public function testDividir3() {

$this->assertEquals(

1, $this->object->dividir(1, 1)

);

}

## 4.3 Documentación de pruebas

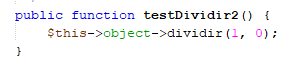
Se han establecido los siguientes casos de prueba para las funciones sumar, restar, multiplicar y dividir:

|  |  |
| --- | --- |
| **Primer número** | **Segundo número** |
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |
| 1 | 2 |

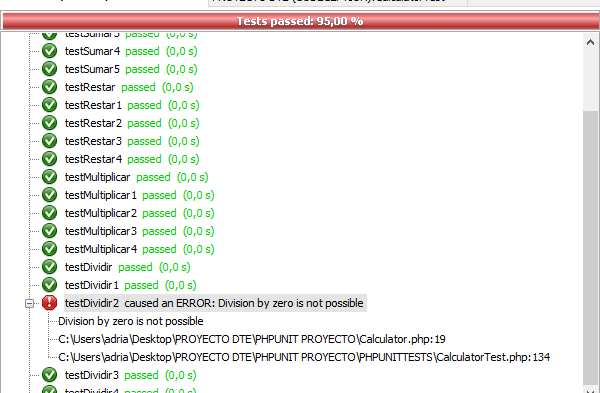
De esta manera, nos aseguramos de controlar resultados negativos, positivos y operaciones con el cero.

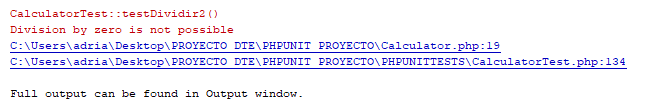
Para superar correctamente las pruebas, hay que indicar el resultado esperado de una operación. Esto ha sido correcto hasta que nos hemos encontrado con problemas en el momento que se da cuando se divide por cero:

En este ejemplo, se muestra el caso en el que se intenta dividir 1 entre 0:

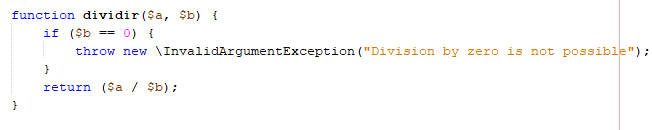


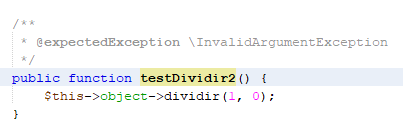
Por tanto, no se puede indicar el resultado esperado y se obtiene un error:





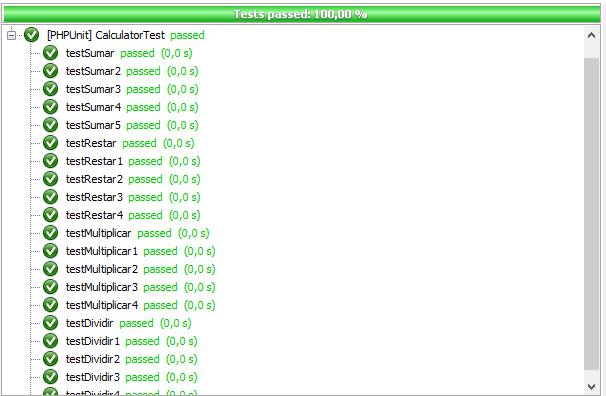
Para resolver esta situación, hay que controlar dicha excepción de la siguiente manera:





De esta forma, le estamos indicando que esperamos obtener una excepción al realizar esa operación. En concreto, la división por cero.

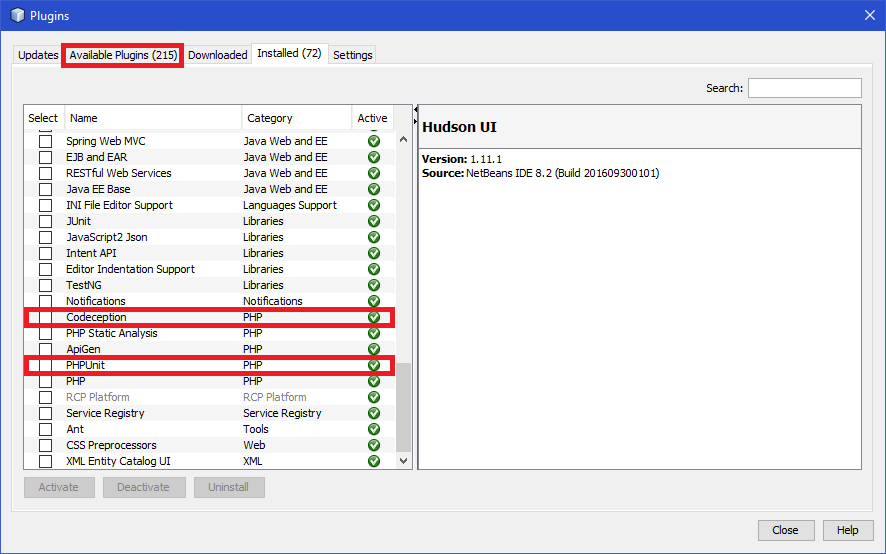
Finalmente, el resultado de las pruebas efectuadas es satisfactorio:

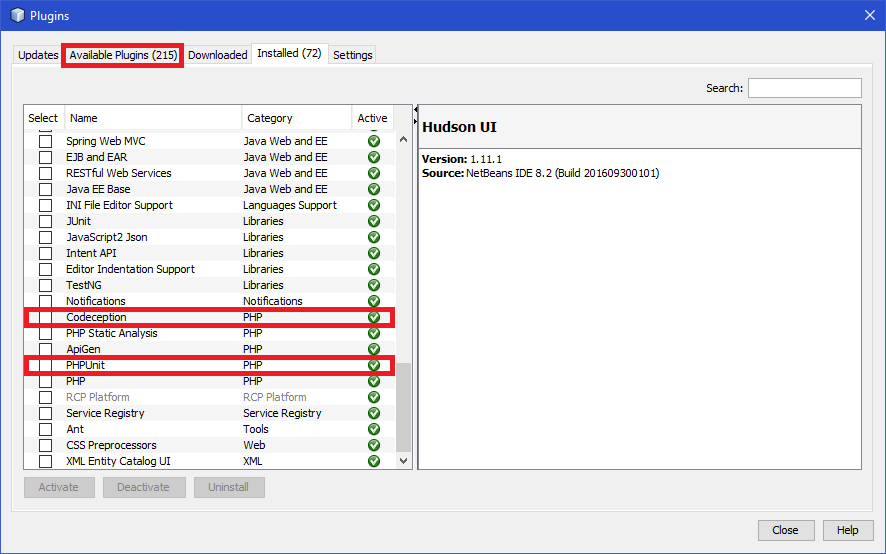


## 4.4 Documentación de instalación

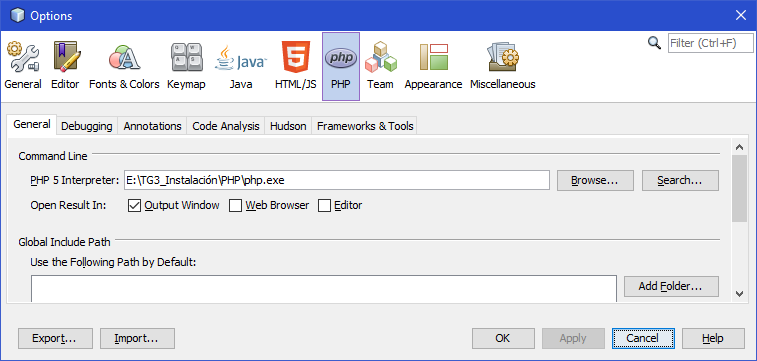
**Instalación de NetBeans**: en nuestro caso hemos utilizado NetBeans como plataforma para desarrollar los test con ambas plataformas. Se puede descargar la última versión desde el siguiente enlace, eligiendo la configuración completa (all) correspondiente a la última columna: <https://netbeans.org/downloads/8.2/>

**Plugin de PHP y PHPunit:** desde NetBeans hay que comprobar que estén instalados los plugins de PHP y PHPunit. Para ello hay que ir a la pestaña Tools y seleccionar Plugins. Es posible que si se ha elegido la configuración completa no sea necesario.

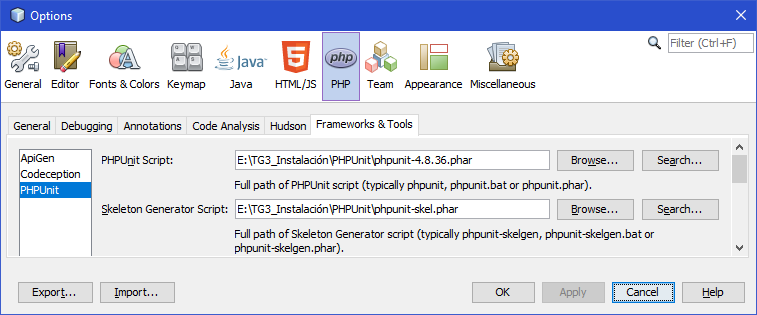




**Configurar PHP:** se accede a los ajustes de NetBeans en la pestaña Tools > Options. Hecho esto se accede al menú de PHP y en la pestaña General hay que seleccionar la ruta del archivo php.exe que se encuentra disponible en “TG3\_Instalación > PHP”, en el repositorio de GitHub (no está comprimido porque el tamaño supera el máximo permitido). Es necesario descargar la carpeta PHP completa ya que se han modificado los archivos .ini necesarios para que el proyecto funcione adecuadamente



**Configurar PHPunit con el archivo.phar y el Skeleton Generator:** se accede a los ajustes de NetBeans en la pestaña Tools > Options. Hecho esto se accede al menú de PHP y, a su vez, a la pestaña Frameworks & Tools, y se elige PHPUnit en el menú izquierdo.



En la primera caja (PHPUnit Script) hay que indicar la ruta al archivo PHAR de PHPUnit, y en el segundo el Skeleton Generator Script. Ambos se encuentran en la carpeta “TG3\_Instalación > PHPUnit” del repositorio de GitHub.

Por último, hay que descargar la carpeta del proyecto y abrirlo desde NetBeans.

## 4.5 Manual de usuario

Una vez llevada a cabo la instalación descrita en el apartado anterior. Hay que descargar la carpeta del proyecto para PHPunit disponible en el repositorio de GitHub del grupo T8. Una vez descargado, hay que abrirlo desde NetBeans para poder ejecutar la suite de pruebas.

# 5. Proyecto de implementación de un prototipo del sistema utilizando Codeception

A continuación, se describe toda la documentación necesaria para entender el desarrollo de los test realizados con Codeception, las pruebas realizadas durante el proceso de implementación y la configuración requerida para poder probar el proyecto en cualquier otro equipo.

## 5.1 Documentación de diseño

Para esta herramienta hemos utilizado el mismo código que se ha utilizado para la herramienta anterior, que consistía en una calculadora con las funciones principales (suma, resta, multiplicación y división), la cual nos sirve para realizar el cometido de esta práctica, testing automático para PHP, y en base a ese código hemos realizado las pruebas con Codeception.

Al igual con la otra herramienta, hemos diseñado los siguientes casos de prueba para comprobar que cada una de las funciones realiza lo que se espera de ella. Comprobando con estos test que se realiza correctamente la suma, la resta, la multiplicación y la división, controlando en este último caso la división entre cero como una excepción.

* TestSuma
* TestResta
* TestMultiplicacion
* TestDivision

## 5.2 Documentación de construcción

* TestSumar

Para realizar la función de suma el prototipo que se ha seguido ha sido el siguiente uso de código PHP que suma los números a y b:

function sumar($a, $b) {

return ($a + $b);

}

Para realizar su testing, hemos cinco funciones que realizan una suma de dos números dados y comprueba que el dato introducido sea correcto al dado como se puede ver en el caso del siguiente ejemplo que suma 0+0 y como resultado tiene que proporcionar el valor de 0:

public function testSumar() {

$this->assertEquals(

0, $this->object->sumar(0, 0)

);

}

* TestRestar

Para realizar la función de restar dos números se ha utilizado el siguiente fragmento de código que resta el número a menos el número b:

function restar($a, $b) {

return ($a - $b);

}

En cuanto al testing de esta operación, se han elaborado cuatro casos en los que dados dos números se comprueba que al realizar la resta el resultado obtenido es el resultado deseado, como por ejemplo en el fragmento de código siguiente que resta 1-2 dando como resultado –1.

public function testRestar4() {

$this->assertEquals(

-1, $this->object->restar(1, 2)

);

}

* TestMultiplicar

La función de multiplicar se ha realizado usando el siguiente código que multiplica el primer valor a por el segundo valor b:

function multiplicar($a, $b) {

return ($a \* $b);

}

Para realizar el testing de la función de multiplicar, se han elaborado cuatro casos que realizan la propia multiplicación de dos números comprobando que su resultado es correcto, como puede observarse el en código siguiente que multiplicando 1x1 obtenemos el resultado 1.

public function testMultiplicar3() {

$this->assertEquals(

1, $this->object->multiplicar(1, 1)

);

}

* TestDividir

La última función de la calculadora es la división de dos números, teniendo en cuenta que el denominador de dicha división no puede ser 0, ya que en tal caso no sería posible realizar la operación y nos mostraría un error diciendo que la división por cero no es posible.

function dividir($a, $b) {

if ($b == 0) {

throw new \InvalidArgumentException("Division by zero is not possible");

}

return ($a / $b);

}

En cuanto al testing de esta última operación se han realizado un caso, incluyendo el posible error de dividir entre 0:

public function testDividir2() {

$this->object->dividir(1, 0);

}

Y realizando otros tres casos, observando que el resultado obtenido al dividir dos números introducidos es correcto:

public function testDividir3() {

$this->assertEquals(

1, $this->object->dividir(1, 1)

);

}

## 5.3 Documentación de pruebas

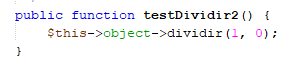
Se han establecido los siguientes casos de prueba para las funciones sumar, restar, multiplicar y dividir:

|  |  |
| --- | --- |
| **Primer número** | **Segundo número** |
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |
| 1 | 2 |

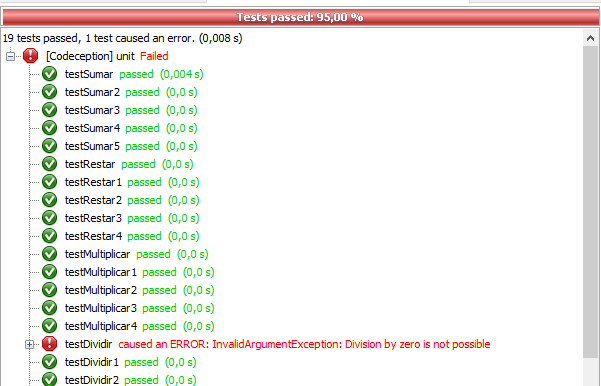
De esta manera, nos aseguramos de controlar resultados negativos, positivos y operaciones con el cero.

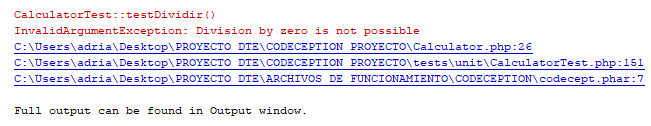
Para superar correctamente las pruebas, hay que indicar el resultado esperado de una operación. Esto ha sido correcto hasta que nos hemos encontrado con problemas en el momento que se da cuando se divide por cero:

En este ejemplo, se muestra el caso en el que se intenta dividir 1 entre 0:

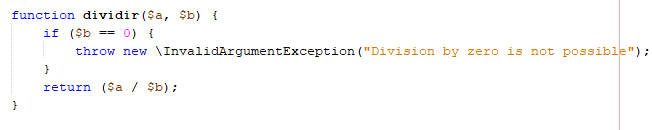


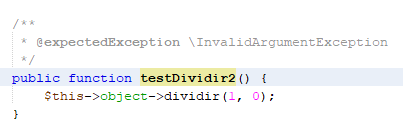
Por tanto, no se puede indicar el resultado esperado y se obtiene un error:





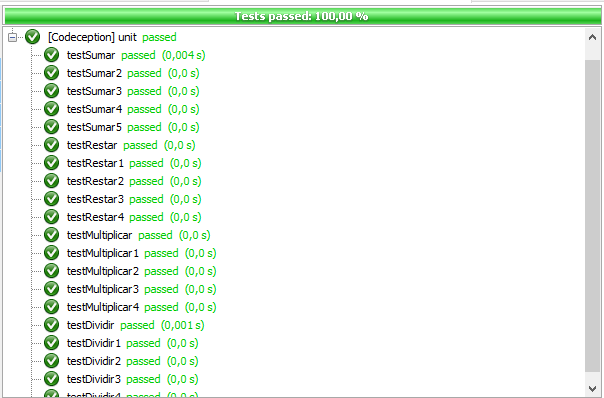
Para resolver esta situación, hay que controlar dicha excepción de la siguiente manera:





De esta forma, le estamos indicando que esperamos obtener una excepción al realizar esa operación. En concreto, la división por cero.

Finalmente, el resultado de las pruebas efectuadas es satisfactorio:

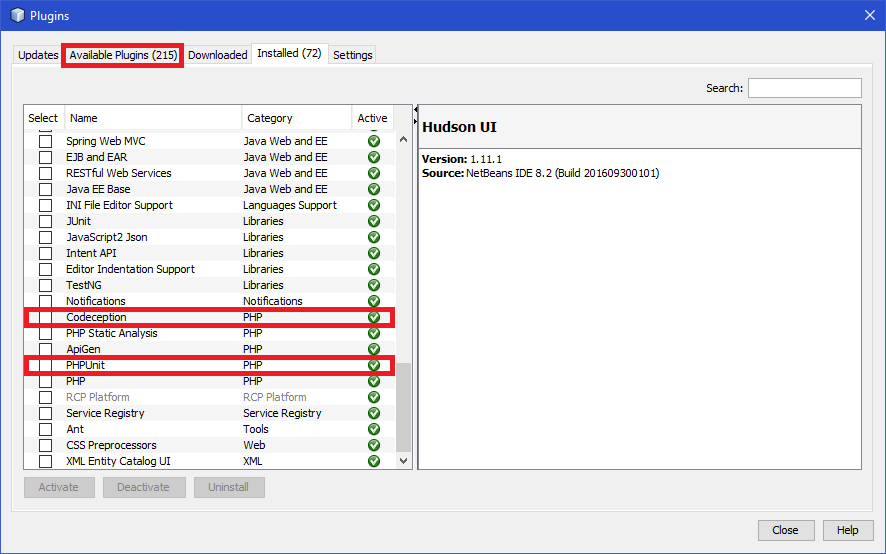


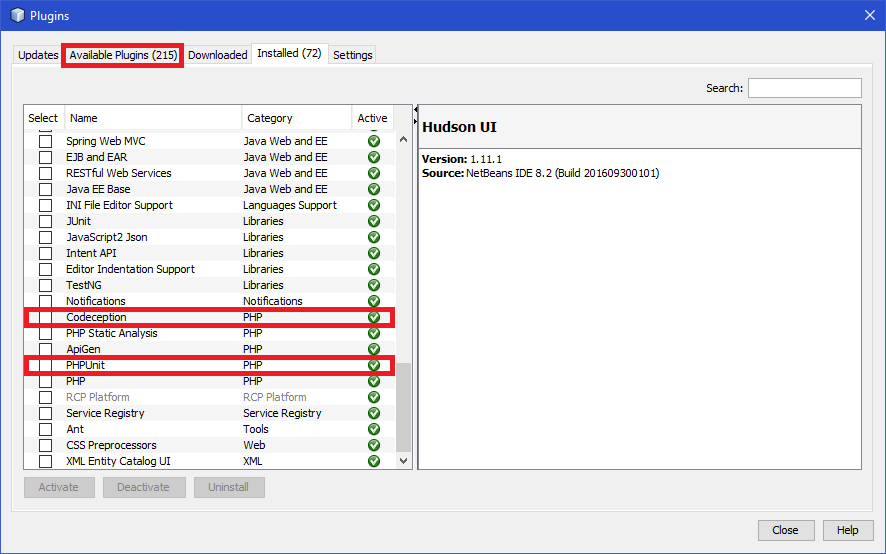
## 5.4 Documentación de instalación

**Instalación de NetBeans**: en nuestro caso hemos utilizado NetBeans como plataforma para desarrollar los test con ambas plataformas. Se puede descargar la última versión desde el siguiente enlace, eligiendo la configuración completa (all) correspondiente a la última columna: <https://netbeans.org/downloads/8.2/>

Si ya se ha llevado a cabo para la utilización de PHPunit no hay que repetir la instalación.

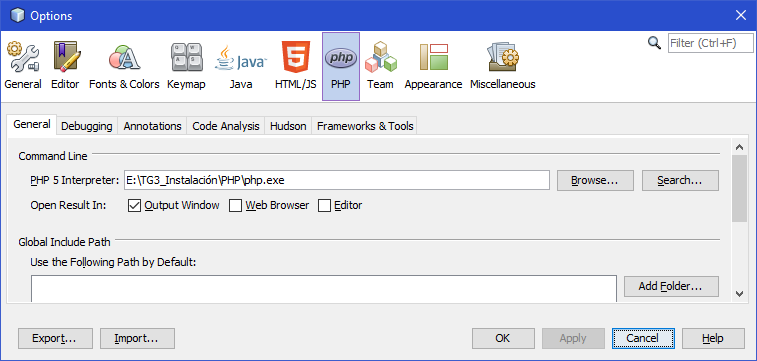
**Plugin de PHP y Codeception:** desde NetBeans hay que comprobar que estén instalados los plugins de PHP y Codeception. Para ello hay que ir a la pestaña Tools y seleccionar Plugins. Al igual que en el caso anterior, es posible que si se ha elegido la configuración completa no sea necesario.



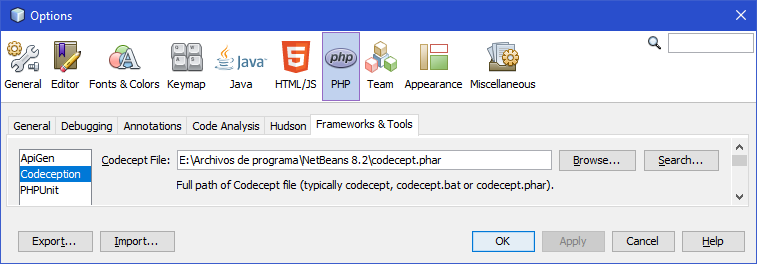


**Configurar PHP:** En caso de que ya se haya configurado PHP en el prototipo de PHPunit no será necesario volver a configurarlo. En caso contrario las instrucciones son las siguientes:

Se accede a los ajustes de Netbeans en la pestaña Tools > Options. Hecho esto se accede al menú de PHP y en la pestaña General hay que seleccionar la ruta del archivo php.exe que se encuentra disponible en la carpeta del proyecto.



**Configurar Codeception con el archivo .phar:** se accede a los ajustes de NetBeans en la pestaña Tools > Options. Hecho esto se accede al menú de PHP y, a su vez, a la pestaña Frameworks & Tools, y se elige Codeception en el menú izquierdo.



## 5.5 Manual de usuario

Una vez llevada a cabo la instalación descrita en el apartado anterior. Hay que descargar la carpeta del proyecto para Codeception disponible en el repositorio de GitHub del grupo T8. Una vez descargado, hay que abrirlo desde NetBeans para poder ejecutar la suite de pruebas.

# 6. Comparación de las dos implementaciones

Teniendo en cuenta los criterios de comparación definidos en el apartado 3 sobre la implementación de cada uno de los prototipos, se ha fijado la evaluación de los criterios en la implementación usando ambas herramientas.

## 6.1 Evaluación de los criterios en la implementación usando PHPUnit

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| C1 Dificultad instalación SW | 3 horas |
| C2 Recursos necesarios | 3 herramientas |
| C3 Tiempo de respuesta | 0,000 segundos |
| C4 Información disponible | 2 |
| C5 Implementación intuitiva | 2 |

## 6.2 Evaluación de los criterios en la implementación usando Codeception

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| C1 Dificultad instalación SW | 4 horas |
| C2 Recursos necesarios | 3 herramientas |
| C3 Tiempo de respuesta | 0,007 segundos |
| C4 Información disponible | 1 |
| C5 Implementación intuitiva | 1 |

# 7. Comparación de la implementación de las tecnologías

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIOS | PHPUnit | Codeception | COMENTARIOS |
| C1 Dificultad instalación SW | 3 horas | 4 horas | Se ha tardado una hora más en Codeception, mayor complejidad. |
| C2 Recursos necesarios | 3 herramientas | 3 herramientas | Necesitamos la misma cantidad de herramientas- |
| C3 Tiempo de respuesta | 0,000 segundos | 0,007 segundos | En el código PHP creado se tarda más en realizarse las pruebas en Codeception, al ser un código tan pequeño los tiempos son mínimos, tanto es así, que en PHPUnit nos da 0,000 segundos. |
| C4 Información disponible | 2 | 1 | Se ha encontrado algo más de información en PHPUnit, pero aun así la cantidad de información encontrada ha sido muy reducida y difícil de encontrar. |
| C5 Implementación intuitiva | 2 | 1 | Como era más difícil encontrar la información en Codeception, ante un problema nos ha costado más resolverlo, y encima hemos tenido más problemas con Codeception para hacer funcionar sus tests. |

# 8. Conclusiones

Tal y como se explicó en el trabajo anterior PHPUnit y Codeception presentan funcionalidades similares.

Codeception es más aconsejable para aplicaciones grandes ya que la velocidad que ofrece para pruebas complejas es mayor. También ofrece una mayor usabilidad a la hora de entender los test de prueba. Sin embargo, el esfuerzo requerido para la puesta en marcha y la configuración es mucho mayor. Esto es debido a que tiene mayor complejidad, pero también a que hay mucha menos documentación disponible.

En este caso en concreto, la aplicación que se ha desarrollado es pequeña y los tests requeridos no implican una dificultad excesiva por lo que la mejor opción es PHPUnit.