

**title:** Clase 11 Proyecto 2014  
**Author:** Einar Lanfranco, Claudia Banchoff  
**description:** Notaciones para descripción de datos  
**keywords:** test de unidad  
**css:** proyecto.css

---

## Proyecto de Software

### Cursada 2014

---

#### Temario

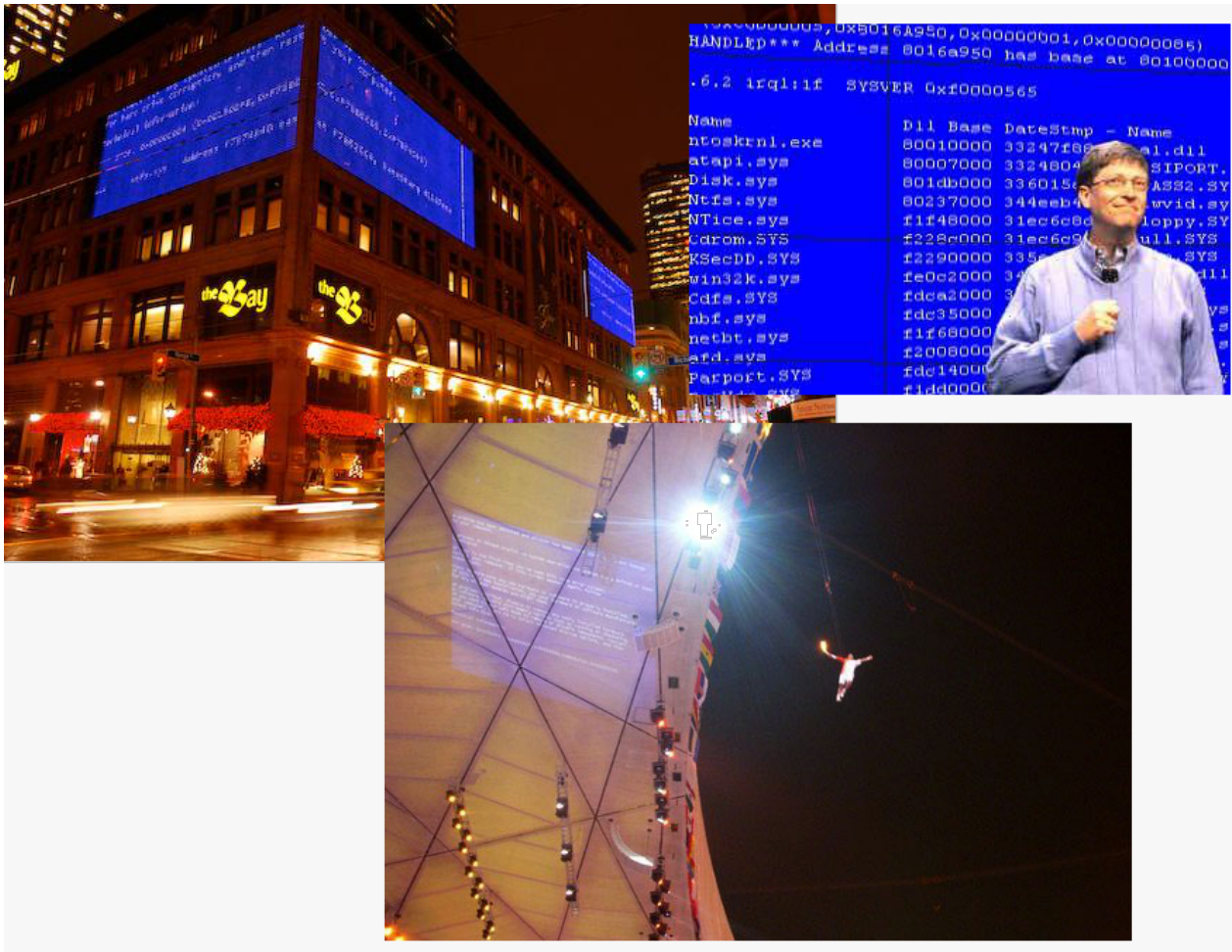
- Testeando código
  - PHP Unit
- 

#### ¿Por qué testear?

- Existe una gran cantidad de parches y versiones surgidas luego del lanzamiento de una versión final de un software, destinadas a cubrir "agujeros de seguridad" o malos funcionamientos.
  - Por lo general, la etapa de pruebas es la menos sistematizada y tenida en cuenta.
- 

#### ¿Por qué testear?

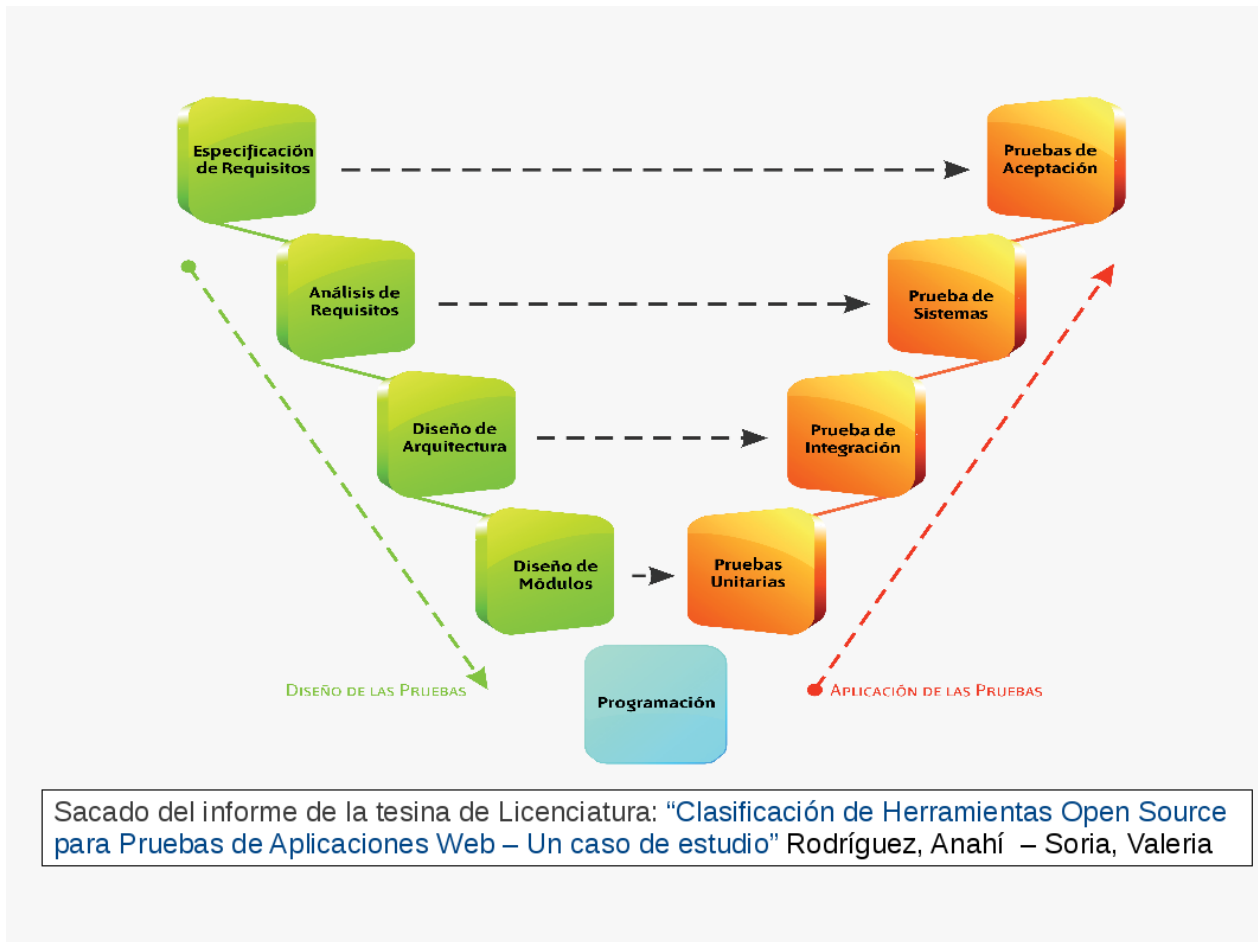
"Una imagen vale más que mil palabras..."



```
0x00000000,0x016A950,0x00000001,0x00000005)  
HANDLE*** Address: 0016a950 has base at 00100000  
6.2 Irql:1f SYSVER 0x10000565  
Name Dll Base DateStmp - Name  
ntoskrnl.exe 80010000 33247f89 - al.dll  
atapi.sys 80007000 3324804 - SIPOPT.  
Disk.sys 801db000 336015 - ASS2.SY  
Ntfs.sys 80237000 344eeb4 - wvid.sy  
NTfs.sys f1f48000 31ec6e8d - lippy.SY  
LdrRom.SYS f228e000 31ec6e9 - ull.SYS  
KSecDD.SYS f2290000 335 - .SYS  
win32k.sys fe0e2000 3 - .dll  
Cdfe.SYS fdca2000 3 - .ys  
nbf.sys fdca35000 - .s  
netbt.sys f1f68000 - .s  
ard.sys f2008000 - .s  
Parport.SYS fdca14000 - .s  
f1dd0000
```



# Testeando código



class: destacado

## Tipos de pruebas

- **Pruebas de unidad:** Permite examinar cada módulo de manera individual para asegurar el correcto funcionamiento.
- **Pruebas de integración:** Permiten ver si existen fallas en la interacción de un módulo con otros módulos.
- **Pruebas de sistema:** Permiten analizar al sistema como un todo, verificando que cumple con los requisitos especificados, la interacción del mismo con el hardware, lo cual lleva a realizar pruebas de rendimiento, seguridad, resistencia, de recuperación, etc.
- **Pruebas de aceptación:** Permiten encontrar errores que sólo el usuario final puede descubrir.

Para muchas de estas pruebas existen herramientas que ayudan.

## Pruebas de Unidad

- Permiten examinar cada parte del programa (módulo, clase o función) de manera individual para asegurar su correcto funcionamiento.
  - Por lo general, se realizan una vez que se ha desarrollado, revisado y verificado el código fuente.
- 

## Pruebas de Unidad: Ventajas

- Permite probar que el programa funciona correctamente.
  - Se puede identificar variables que no existen o tipos esperados en las funciones.
  - Permite identificar, en caso de que se haga una modificación, que siga funcionando correctamente todas las parte del programa.
  - Permiten documentar el código (no en forma directa, pero como los tests indican cómo es que se tiene que comportar el programa, viendo los tests se puede ver el resultado esperado para ciertas entradas del programa). Esto no excluye que se tenga que escribir la documentación del código.
- 

**class:** destacado

## Pruebas de Unidad: Desventajas

- Toman bastante tiempo de escribir. Algunas clases son fáciles de testear pero otras no tanto.
- Cuando se hace un gran cambio en el código hay que actualizar los tests.
- Algo muy importante a tener en cuenta:

Aprobar los tests no significa que el sistema funcione a la perfección.

---

## ¿Cómo tienen que ser los tests?

- Tiene que poder correr sin interacción humana: sin que el usuario ingrese valores en ningún caso.
  - Tienen que poder verificar el resultado de la ejecución sin interacción humana: se debe poder analizar los resultados en forma automática.
  - Un test tiene que ser independiente del otro. Es decir, el resultado de un test no debería depender del resultado anterior.
- 

## ¿Qué condiciones deben cumplir un test?

- Que funcione correctamente cuando los valores de entrada son válidos.
  - Que falle cuando los valores de entrada son inválidos, o que levante una excepción.
-

# ¿Test de unidad en PHP?

## PHPUnit

---

## PHPUnit

- Instalamos de varias formas (ver en documentación)
- Usamos: `phpunit nombreTest`

Donde **nombreTest** poder ser:

- un único test
  - un directorio con los test
- 

## Algunos conceptos importantes

- **test case** (o caso de prueba): chequea por una respuesta específica a un conjunto de entradas (PHPUnit provee una clase denominada `PHPUnit_Framework_TestCase` que permite crear estos casos)
  - **test suite**: es el conjunto de los casos de prueba.
  - **test runner**: permite la ejecución de las pruebas
- 

## Primero... ¿test?

- Si vamos a testear la clase "Alimento", entonces lo haremos definiendo la clase "AlimentoTest".
  - AlimentoTest extenderá la clase **`PHPUnit_Framework_TestCase`**.
  - El caso de prueba (AlimentoTest) chequea por una respuesta específica a un conjunto de entradas.
  - Cada test es un método público que debe comenzar con **"test"**, de manera tal que puedan ser reconocidos por el test runner
  - **Cada test invoca métodos tales como:**
    - `assertEqual()`: para chequear por un resultado esperado.
    - `assertTrue()`: para verificar una condición.
    - `assertNull()`: para chequear si un resultado es null.
    - Ver todos en: <https://phpunit.de/manual/current/en/appendixes.assertions.html>
- 

## Veamos un ejemplo sencillo

Este ejemplo, testea las operaciones pop y push.

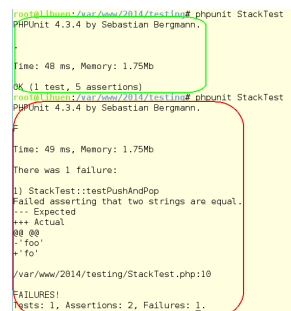
```
<?php
class StackTest extends PHPUnit_Framework_TestCase{
    public function testPushAndPop()
    {
        $stack = array();
        $this->assertEquals(0, count($stack));

        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertEquals(1, count($stack));

        $this->assertEquals('foo', array_pop($stack));
        $this->assertEquals(0, count($stack));
    }
}
?>
```

## Ejecutando el test

- El comando **phpunit**



```
root@libman:/var/www/2014/testing# phpunit StackTest
PHPUnit 4.3.4 by Sebastian Bergmann.

Time: 48 ms, Memory: 1.75Mb
OK (1 test, 5 assertions)

root@libman:/var/www/2014/testing# phpunit StackTest
PHPUnit 4.3.4 by Sebastian Bergmann.

Time: 49 ms, Memory: 1.75Mb
There was 1 failure:
1) StackTest::testPushAndPop
Failed asserting that two strings are equal.
... Expected
... Actual
00 00
"foo"
"fo"

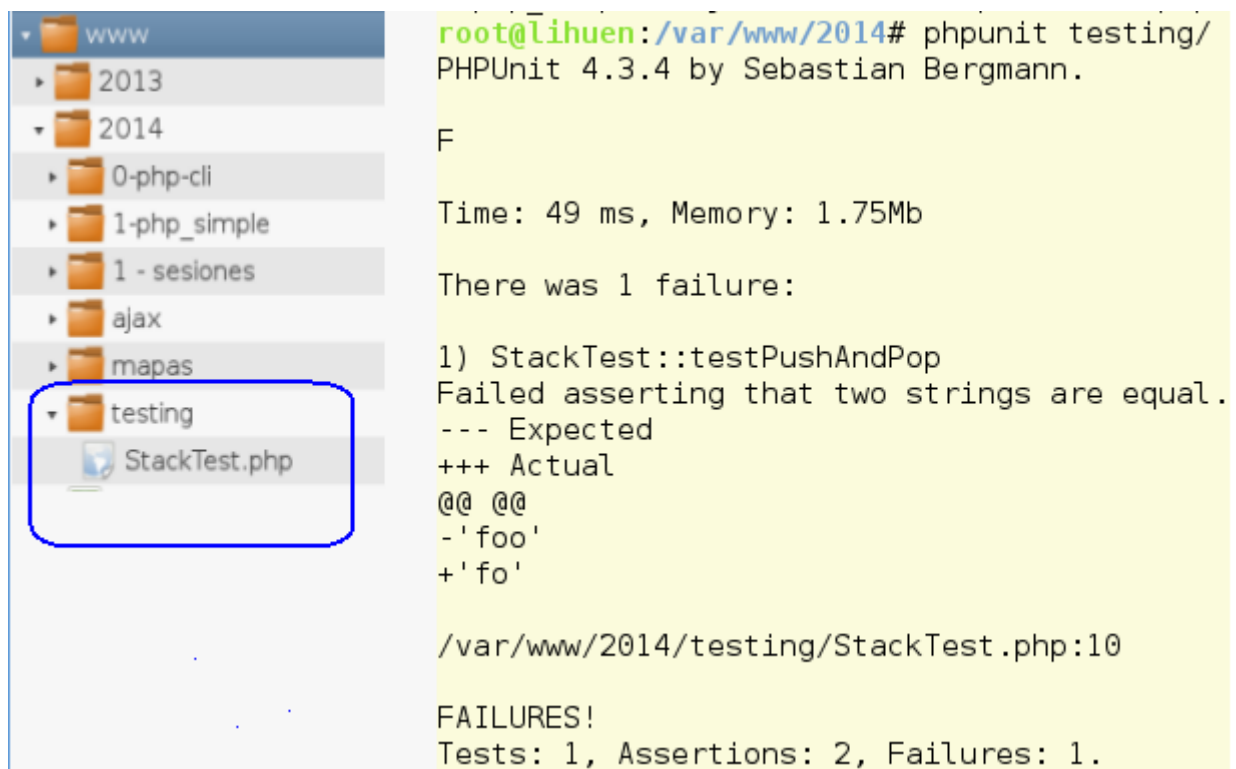
/var/www/2014/testing/StackTest.php:10

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

- En nuestro caso: **phpunit StackTest**

## Organizando los test (test suite)

- Ubicando los test en un directorio
- Ejemplo: **phpunit testing**



The image shows a file explorer on the left and a terminal window on the right. The file explorer displays a directory structure under 'www', with 'testing/StackTest.php' highlighted by a blue box. The terminal window shows the execution of 'phpunit testing/' in the directory '/var/www/2014'. The output indicates a failure in the 'StackTest::testPushAndPop' test, where the expected string 'foo' did not match the actual string 'fo'.

```
root@lihuen:/var/www/2014# phpunit testing/
PHPUnit 4.3.4 by Sebastian Bergmann.

F

Time: 49 ms, Memory: 1.75Mb

There was 1 failure:

1) StackTest::testPushAndPop
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'foo'
+'fo'

/var/www/2014/testing/StackTest.php:10

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

---

## Organizando los test (cont.)

- Describiendo el **test suite** usando XML
- Ejemplo: `phpunit -c archivo.xml`

```
<phpunit>
  <testsuites>
    <testsuite name="Mis pruebas">
      <file>testing/StackTest.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

- Más info en <https://phpunit.de/manual/current/en/organizing-tests.html>
- El archivo de configuración

---

## Referencias

- <https://phpunit.de/>
- <http://librosweb.es/tutorial/accelera-la-ejecucion-de-tus-tests-con-phpunit/>