Hello! My name is Sebastian.
Hello! My name is Sebastian …

and this is what I do:

- sebastian@thePHP.cc
  Co-Founder of The PHP Consulting Company

- sebastian@phpunit.de
  Creator and Main Developer of PHPUnit

- sebastian@php.net
  Contributor to PHP

- sebastian@apache.org
  Contributor to Zeta Components

- sbergmann@{acm.org | ieee.org}
  Practitioner of academic computing
Get the help you need from the experts you trust.

Consulting | Code Review | Training
Some of these practices may be obvious to you.

Especially the first one :-)
Do not write tests that do not test anything

```php
<?php

class FooTest extends PHPUnit_Framework_TestCase {

    public function testSomething() {
        $foo = new Foo;
        $foo->doSomething(new Bar);
    }

} 
```
Best Practices for Writing Tests

Do not write tests that do not test anything

```
sb@ubuntu ~ % phpunit FooTest
PHPUnit 3.5.0 by Sebastian Bergmann.
.
Time: 0 seconds, Memory: 3.75Mb
OK (1 test, 0 assertions)
```
Best Practices for Writing Tests

Do not write tests that do not test anything

sb@ubuntu ~ % phpunit FooTest
PHPUnit 3.5.0 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 3.75Mb

OK (1 test, 0 assertions)
Best Practices for Writing Tests

Do not write tests that do not test anything

```
sb@ubuntu ~ % phpunit --strict FooTest
PHPUnit 3.5.0 by Sebastian Bergmann.

I

Time: 0 seconds, Memory: 3.75Mb

OK, but incomplete or skipped tests!
Tests: 1, Assertions: 0, Incomplete: 1.
```
Do not write tests that do not test anything

```
 sb@ubuntu ~ % phpunit --strict --verbose FooTest
 PHPUnit 3.5.0 by Sebastian Bergmann.

 FooTest
 I

 Time: 0 seconds, Memory: 3.75Mb

 There was 1 incomplete test:

 1) FooTest::testSomething
 This test did not perform any assertions

 OK, but incomplete or skipped tests!
 Tests: 1, Assertions: 0, Incomplete: 1.
```
Best Practices for Writing Tests

Do not write tests that do not test anything

```php
<?php
class FooTest extends PHPUnit_Framework_TestCase {
    public function testSomething() {
        $foo = new Foo;
        $this->assertEquals('something', $foo->doSomething(new Bar));
    }
}
```
Best Practices for Writing Tests

Do not write tests that do not test anything

```
sb@ubuntu ~ % phpunit FooTest
PHPUnit 3.5.0 by Sebastian Bergmann.
.
Time: 0 seconds, Memory: 3.75Mb
OK (1 test, 1 assertion)
```
Do not write tests that test too much

```php
<?php
class StackTest extends PHPUnit_Framework_TestCase {
    public function testPushAndPopWorks()
    {
        $stack = array();
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
```
Best Practices for Writing Tests

Do not write tests that test too much

```php
<?php
class StackTest extends PHPUnit_Framework_TestCase
{
    public function testPushAndPopWorks()
    {
        $stack = array();
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        $this->assertNotEmpty($stack);
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
```
Best Practices for Writing Tests

Do not write tests that test too much

sb@ubuntu ~ % phpunit --testdox StackTest
PHPUnit 3.5.0 by Sebastian Bergmann.

Stack
[x] Push and pop works
Best Practices for Writing Tests

Do not write tests that test too much

```
sb@ubuntu ~ % phpunit --testdox StackTest
PHPUnit 3.5.0 by Sebastian Bergmann.

Stack
  [x] Push and pop works
```
Best Practices for Writing Tests

Exploit dependencies between tests

```php
<?php

class StackTest extends PHPUnit_Framework_TestCase {

    public function testStackIsInitiallyEmpty() {
        $stack = array();
        $this->assertEmpty($stack);
        return $stack;
    }

    /**
     * @depends testStackIsInitiallyEmpty
     */
    public function testPushingAnElementOntoTheStackWorks(array $stack) {
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        return $stack;
    }

    /**
     * @depends testPushingAnElementOntoTheStackWorks
     */
    public function testPoppingAnElementOffTheStackWorks(array $stack) {
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
```
Best Practices for Writing Tests

Exploit dependencies between tests

sb@ubuntu ~ % phpunit --testdox StackTest
PHPUnit 3.5.0 by Sebastian Bergmann.

Stack
[x] Stack is initially empty
[x] Pushing an element onto the stack works
[x] Popping an element off the stack works
Best Practices for Writing Tests

Exploit dependencies between tests

```php
<?php
class StackTest extends PHPUnit_Framework_TestCase
{

    public function testStackIsInitiallyEmpty()
    {
        $stack = array('foo');
        $this->assertEmpty($stack);
        return $stack;
    }

    /**
     * @depends testStackIsInitiallyEmpty
     */
    public function testPushingAnElementOntoTheStackWorks(array $stack)
    {
        array_push($stack, 'foo');
        $this->assertEquals('foo', $stack[count($stack)-1]);
        return $stack;
    }

    /**
     * @depends testPushingAnElementOntoTheStackWorks
     */
    public function testPoppingAnElementOffTheStackWorks(array $stack)
    {
        $this->assertEquals('foo', array_pop($stack));
        $this->assertEmpty($stack);
    }
}
```
Exploit dependencies between tests

sb@ubuntu ~ % phpunit StackTest
PHPUnit 3.5.0 by Sebastian Bergmann.

FSS

Time: 0 seconds, Memory: 4.00Mb

There was 1 failure:

1) StackTest::testEmpty
Failed asserting that an array is empty.

/home/sb/StackTest.php:7

FAILURES!
Best Practices for Writing Tests

Exploit dependencies between tests

```
sb@ubuntu ~ % phpunit --verbose StackTest
PHPUnit 3.5.0 by Sebastian Bergmann.
StackTest
FSS
Time: 0 seconds, Memory: 3.75Mb

There was 1 failure:
1) StackTest::testEmpty
   Failed asserting that an array is empty.
   /home/sb/StackTest.php:7

There were 2 skipped tests:
1) StackTest::testPush
   This test depends on "StackTest::testEmpty" to pass.
2) StackTest::testPop
   This test depends on "StackTest::testPush" to pass.

FAILURES!
```
Best Practices for Writing Tests

Use the most specific assertion available to express what you want to test

```php
$this->assertEmpty($stack);
```

vs.

```php
$this->assertTrue(empty($stack));
```
Best Practices for Writing Tests

Use the most specific assertion available to express what you want to test

```php
$this->assertEmpty($stack);

vs.

$this->assertTrue(empty($stack));
```

```php
$this->assertInstanceOf('Foo', $foo);

vs.

$this->assertTrue($foo instanceof Foo);
```
Best Practices for Writing Tests

Use the most specific assertion available to express what you want to test

```php
$this->assertEmpty($stack);
vs.
$this->assertTrue(empty($stack));
```

```php
$this->assertInstanceOf('Foo', $foo);
vs.
$this->assertTrue($foo instanceof Foo);
```

```php
$this->assertInternalType('string', 'foo');
vs.
$this->assertTrue(is_string('foo'));
```
Decouple test code from test data

```php
class DataTest extends PHPUnit_Framework_TestCase {
    /**
     * @dataProvider providerMethod
     */
    public function testAdd($a, $b, $c) {
        $this->assertEquals($c, $a + $b);
    }

    public function providerMethod() {
        return array(
            array(0, 0, 0),
            array(0, 1, 1),
            array(1, 1, 3),
            array(1, 0, 1)
        );
    }
}
```
Best Practices for Writing Tests

Decouple test code from test data

```
sb@ubuntu ~ % phpunit DataTest
PHPUnit 3.5.0 by Sebastian Bergmann.

..F.

Time: 0 seconds, Memory: 4.00Mb

There was 1 failure:

1) DataTest::testAdd with data set #2 (1, 1, 3)
Failed asserting that <integer:2> matches expected <integer:3>.

/home/sb/DataTest.php:9

FAILURES!
```
Best Practices for Organizing Tests

Composing a Test Suite Using the Filesystem

Object
|-- Freezer
 |   |-- HashGenerator
 |       `-- NonRecursiveSHA1.php
 |   |-- HashGenerator.php
 |   |-- IdGenerator
 |       `-- UUID.php
 |   |-- IdGenerator.php
 |   |-- LazyProxy.php
 |   |-- Storage
 |       `-- CouchDB.php
 |   `-- Storage.php
 |     `-- Util.php
 `-- Freezer.php

Tests
|-- Freezer
 |   |-- HashGenerator
 |       `-- NonRecursiveSHA1Test.php
 |   |-- IdGenerator
 |       `-- UUIDTest.php
 |   `-- Storage
 |       `-- CouchDB
 |           `-- WithLazyLoadTest.php
 |           `-- WithoutLazyLoadTest.php
 |   `-- StorageTest.php
 |     `-- UtilTest.php
 `-- FreezerTest.php
Running all tests in a directory

sb@ubuntu ~ % phpunit Tests
PHPUnit 3.5.0 by Sebastian Bergmann.

............................................................ 60 / 75

...............

Time: 0 seconds, Memory: 11.00Mb

OK (75 tests, 164 assertions)
Best Practices for Organizing Tests

Running all tests of a test case class

```
sb@ubuntu ~ % phpunit Tests/FreezerTest
PHPUnit 3.5.0 by Sebastian Bergmann.

............................

Time: 0 seconds, Memory: 8.25Mb

OK (28 tests, 60 assertions)
```
Best Practices for Organizing Tests

Filter tests based on name

```
sb@ubuntu ~ % phpunit --filter testFreezingAnObjectWorks Tests
PHPUnit 3.5.0 by Sebastian Bergmann.
.
Time: 0 seconds, Memory: 10.25Mb
OK (1 test, 2 assertions)
```
Best Practices for Running Tests

Use an XML Configuration File

```xml
<?xml version="1.0" encoding="UTF-8"?>

<phpunit>
  <testsuites>
    <testsuite name="My Test Suite">
      <directory>path/to/dir</directory>
    </testsuite>
  </testsuites>
</phpunit>
```
Best Practices for Running Tests

Use a bootstrap script

<?xml version="1.0" encoding="UTF-8"?>

<phpunit bootstrap="/path/to/bootstrap.php">
  <testsuites>
    <testsuite name="My Test Suite">
      <directory>path/to/dir</directory>
    </testsuite>
  </testsuites>
</phpunit>
Best Practices for Running Tests

bootstrap.php

```php
<?php
function __autoload($class)
{
    require $class . '.php';
}
```
Best Practices for Running Tests

Configure the test suite using the XML configuration file

```xml
<?xml version="1.0" encoding="UTF-8"?>

<phpunit>
  <php>
    <const name="foo" value="bar"/>
    <var name="foo" value="bar"/>
    <ini name="foo" value="bar"/>
  </php>
</phpunit>
```
Best Practices for Running Tests

Disable PHPUnit features (that you should not need anyway)

- **Syntax Check**
  - Enabled by default in PHPUnit 3.4
  - Disabled by default in PHPUnit 3.5
  - Removed in PHPUnit 3.6

- **Backup/Restore of global variables**
  - Enabled by default in PHPUnit 3.5
  - Disabled by default in PHPUnit 3.6

- **Backup/Restore of static attributes**
  - Disabled by default
Best Practices for Running Tests

Disable PHPUnit features (that you should not need anyway)

```xml
<?xml version="1.0" encoding="UTF-8"?>

<phpunit backupGlobals="false"
    backupStaticAttributes="false"
    syntaxCheck="false">
    <testsuites>
        <testsuite name="My Test Suite">
            <directory>path/to/dir</directory>
        </testsuite>
    </testsuites>
</phpunit>
```
Best Practices for Code Coverage

Use Code Coverage Whitelisting

```xml
<?xml version="1.0" encoding="UTF-8"?>

<phpunit backupGlobals="false"
    backupStaticAttributes="false"
    syntaxCheck="false">
    <filter>
        <whitelist addUncoveredFilesFromWhitelist="true">
            <directory suffix=".php">path/to/dir</directory>
        </whitelist>
    </filter>
</phpunit>
```
Best Practices for Code Coverage

Make the Code Coverage information more meaningful

```php
<?php

class FooTest extends PHPUnit_Framework_TestCase
{
    /**
     * @covers Foo::doSomething
     */
    public function testSomething()
    {
        $foo = new Foo;

        $this->assertEquals(
            'something', $foo->doSomething(new Bar)
        );
    }
}
```
<?php
/**
 * @covers Foo
 */

class FooTest extends PHPUnit_Framework_TestCase
{
    public function testSomething()
    {
        $foo = new Foo;

        $this->assertEquals('something', $foo->doSomething(new Bar));
    }
}
Best Practices for Code Coverage

Make the Code Coverage information more meaningful

```xml
<?xml version="1.0" encoding="UTF-8"?>

<phpunit mapTestClassNameToCoveredClassName="true"/>
</phpunit>
```
Best Practices for Code Coverage

Make the Code Coverage information more meaningful

```xml
<?xml version="1.0" encoding="UTF-8"?>

<phpunit forceCoversAnnotation="true"/>
</phpunit>
```
The following practices are not really Best Practices.

Not because they are bad, but because you should not need them.
Singletons
<?php
class Singleton
{
    private static $uniqueInstance = NULL;

    protected function __construct() {}
    private final function __clone() {}

    public static function getInstance()
    {
        if (self::$uniqueInstance === NULL) {
            self::$uniqueInstance = new Singleton;
        }

        return self::$uniqueInstance;
    }
}
<?php

class Client
{
    public function doSomething()
    {
        $singleton = Singleton::getInstance();

        // ...
    }
}

<?php

class Client
{
    public function doSomething(Singleton $singleton = NULL)
    {
        if ($singleton === NULL) {
            $singleton = Singleton::getInstance();
        }

        // ...
    }
}

<?php

class ClientTest extends PHPUnit_Framework_TestCase
{

    public function testSingleton()
    {
        $singleton = $this->getMock(
            'Singleton', /* name of class to mock */
            array(),    /* list of methods to mock */
            array(),    /* constructor arguments */
            '',         /* name for mocked class */
            FALSE       /* do not invoke constructor */
        );

        // ... configure $singleton ...

        $client = new Client;
        $client->doSomething($singleton);

        // ...
    }
}

Replacing the singleton with a test-specific equivalent
<?php

class Singleton
{
    private static $uniqueInstance = NULL;

    protected function __construct() {}
    private final function __clone() {}

    public static function getInstance()
    {
        if (self::$uniqueInstance === NULL) {
            self::$uniqueInstance = new Singleton;
        }

        return self::$uniqueInstance;
    }

    public static function reset()
    {
        self::$uniqueInstance = NULL;
    }
}

Alternative implementation of the Singleton pattern
<?php

class Singleton {

    private static $uniqueInstance = NULL;
    public static $testing = FALSE;

    protected function __construct() {}
    private final function __clone() {}

    public static function getInstance()
    {
        if (self::$uniqueInstance === NULL || self::$testing) {
            self::$uniqueInstance = new Singleton;
        }

        return self::$uniqueInstance;
    }

}
<?php

class Registry {
    private static $uniqueInstance = NULL;
    protected $objects = array();

    protected function __construct() {}
    private final function __clone() {}
    public static function getInstance() { /* … */ }

    public function register($name, $object) {
        $this->objects[$name] = $object;
    }

    public function getObject($name) {
        if (isset($this->objects[$name])) {
            return $this->objects[$name];
        }
    }
}

class ClientTest extends PHPUnit_Framework_TestCase
{
    public function testSingleton()
    {
        $singleton = $this->getMock(
            'Singleton', /* name of class to mock */
            array(),    /* list of methods to mock */
            array(),    /* constructor arguments */
            '',         /* name for mocked class */
            FALSE       /* do not invoke constructor */
        );

        // ... configure $singleton ...

        Registry::getInstance()->register('Singleton', $singleton);

        // ...
    }
}
"Static methods are death to testability."

– Miško Hevery
<?php
class Foo
{
    public static function doSomething()
    {
        return self::helper();
    }

    public static function helper()
    {
        return 'foo';
    }
}
?>
<?php

class Foo
{
    public static function doSomething()
    {
        return self::helper();
    }

    public static function helper()
    {
        return 'foo';
    }
}

class FooMock extends Foo
{
    public static function helper()
    {
        return 'bar';
    }
}

var_dump(FooMock::doSomething());
?>
string(3) "foo"
Static Methods

Late Static Binding (PHP 5.3)

```php
<?php
class Foo
{
    public static function doSomething()
    {
        return static::helper();
    }

    public static function helper()
    {
        return 'foo';
    }
}

class FooMock extends Foo
{
    public static function helper()
    {
        return 'bar';
    }
}

var_dump(FooMock::doSomething());
?>
string(3) "bar"
```
<?php
class FooTest extends PHPUnit_Framework_TestCase
{
    public function testDoSomething()
    {
    }
}
<?php
class FooTest extends PHPUnit_Framework_TestCase {
    public function testDoSomething() {
        $class = $this->getMockClass('Foo', /* name of class to mock */ array('helper') /* list of methods to mock */ );
    }
}
<?php

class FooTest extends PHPUnit_Framework_TestCase
{
    public function testDoSomething()
    {
        $class = $this->getMockClass('Foo', /* name of class to mock */
                                         array('helper') /* list of methods to mock */);

        $class::staticExpects($this->any())->method('helper')->will($this->returnValue('bar'));
    }
}

<?php

class FooTest extends PHPUnit_Framework_TestCase
{
    public function testDoSomething()
    {
        $class = $this->getMockClass('Foo', /* name of class to mock */
                                        array('helper') /* list of methods to mock */);

        $class::staticExpects($this->any())
            ->method('helper')
            ->will($this->returnValue('bar'));

        $this->assertEquals('bar', $class::doSomething());
    }
}

Static Methods

... are death to testability.

```php
<?php
class Foo {
    public static function doSomething()
    {
        return Bar::helper();
    }
}

class Bar {
    public static function helper()
    {
        /* ... */
    }
}

class BarMock extends Bar {
    public static function helper()
    {
        return 'baz';
    }
}
```
Testing What Should Not Be Tested

Non-Public Attributes and Methods
Testing What Should Not Be Tested

A class with private attributes and methods

```php
<?php
class Foo
{
    private $bar = 'baz';

    public function doSomething()
    {
        return $this->bar = $this->doSomethingPrivate();
    }

    private function doSomethingPrivate()
    {
        return 'blah';
    }
}
```
Testing What Should Not Be Tested

Assertions on non-public attributes

```php
<?php
class FooTest extends PHPUnit_Framework_TestCase {
    public function testPrivateAttribute()
    {
        $this->assertEquals('baz',
            $this->readAttribute(new Foo, 'bar'),
        );
    }
}
```
Testing What Should Not Be Tested

Assertions on non-public attributes

```php
<?php
class FooTest extends PHPUnit_Framework_TestCase {
    public function testPrivateAttribute() {
        $this->assertAttributeEquals('baz', /* expected value */, 'bar', /* attribute name */, new Foo /* object */);
    }
}
```
<filename>
```php
<?php

class FooTest extends PHPUnit_Framework_TestCase {
    
    public function testPrivateMethod() {
        
        $method = new ReflectionMethod('Foo', 'doSomethingPrivate');
        
        $method->setAccessible(TRUE);
        
        $this->assertEquals('blah', $method->invoke(new Foo));
        
    }

} 
```

Testing a non-public method (requires PHP 5.3.2)
”The secret in testing is in writing testable code”
- Miško Hevery
Avoid the hell that is global state. This includes *singletons* and *static methods*. 
Use loosely coupled objects …
Use loosely coupled objects …

…and dependency injection to wire them together.
Write short methods.
The End

- Web:  http://thePHP.cc/
  http://sebastian-bergmann.de/
- Mail:  sebastian@thePHP.cc
- Twitter:  @s_bergmann
- Slides:  http://talks.thePHP.cc/
- Buy the book:  http://phpqabook.com/