Die drei Dimensionen des Testens

Sebastian Bergmann | 4. Juli 2015

Sebastian Bergmann

Hilft Teams, erfolgreich die richtige Software zu entwickeln.

sharing experience

"I'll take your brain to another dimension / (hold it!) / pay close attention" – The Prodigy

Dimension Zero

Why test?

9/9

0800 Arctan started { 1.2700 9.037 847 025
1000 " stopped - arctan ✓ 9.037 846 ?95 correct
13"uc (032) MP - MC ~~1.9821647000~~ ~~2.13047764915 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.13047764915
correct 2.13067764915

Relays 6-2 in 033 failed special speed test
in relay " 11.00 test.
Relays changed

Relays 3145
Relay 3370

1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.

1545 

Relay #70 Panel F
(moth) in relay.

First actual case of bug being found.

~~1625~~ 1630 Arctangent started.
1700 closed down.

Global Annual Bugfixing Cost: $312 Billion

"[O]n average, software developers spend 50% of their programming time finding and fixing bugs. When projecting this figure onto the total cost of employing software developers, this inefficiency is estimated to cost the global economy $312 billion per year."

— Cambridge University Study

The Humble Programmer

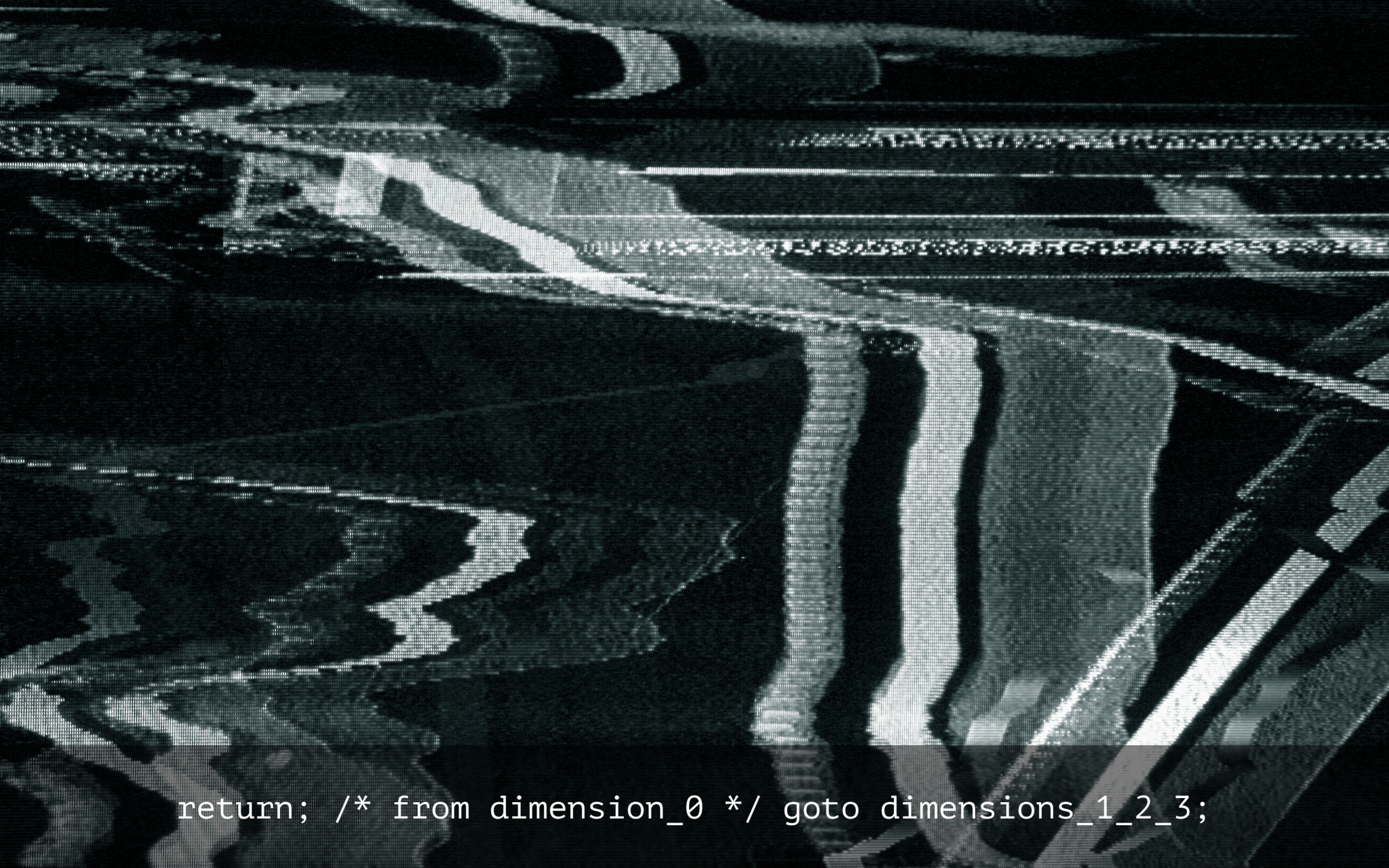
"Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging – they should not introduce bugs to start with."

— Edsger W. Dijkstra


thePHP.cc

Average Technical Debt: $3.61 per LOC

– B. Curtis, J. Sappidi, A. Szynkarski


return; /* from dimension_0 */ goto dimensions_1_2_3;

Types of Software Testing

- Dynamic vs. Static
- Functional vs. Non-Functional
- Black-Box vs. White-Box
- Development vs. Production
- Manual vs. Automated
- ...

The Three Dimensions of Testing

- **Role**

  What do we want achieve with the test?

- **Scope**

  What is the environment in which we perform the test?

- **Implementation**

  How do we denote the test stimulus and our expectations?

Role: Acceptance Test

- **Does our software do the right thing?**
- Helps developers and domain experts to understand and agree on what is built.

Scope: End-to-End Test

- **Does the system as a whole work?**
- Exercises a web application by sending it a real HTTP request and inspecting a real HTTP response returned by it.

```
# Scope:           End-to-End Test
# Implementation: PHPUnit + Webdriver

abstract class WebdriverTestCase extends PHPUnit_Framework_TestCase
{
    protected $webDriver;

    protected function setUp()
    {
        $this->webDriver = RemoteWebDriver::create(
            'http://localhost:4444/wd/hub',
            [
                WebDriverCapabilityType::BROWSER_NAME => 'firefox'
            ]
        );
    }

    protected function tearDown()
    {
        $this->webDriver->close();
        $this->webDriver = null;
    }
}
```

```
# Scope:          End-to-End Test
# Implementation: PHPUnit + Webdriver

class ExampleTest extends WebdriverTestCase
{
    public function testHasCorrectTitle()
    {
        $this->webDriver->get('http://example.org');
        $this->assertEquals('Example Domain', $this->webDriver->getTitle());
    }

    public function testMoreInformationCanBeAccessed()
    {
        $this->webDriver->get('http://example.org');

        $this->webDriver
             ->findElement(WebDriverBy::linkText('More information...'))
             ->click();

        $this->assertContains('IANA', $this->webDriver->getTitle());
    }
}
```

```
# Scope:          End-to-End Test
# Implementation: PHPUnit + Mink (+ Goutte)

use Behat\Mink\Session;
use Behat\Mink\Driver\GoutteDriver;

abstract class MinkTestCase extends PHPUnit_Framework_TestCase
{
    private $session;

    protected function setUp()
    {
        $this->session = new Session(new GoutteDriver);
    }

    protected function visit($url)
    {
        $this->session->visit($url);

        return $this->session->getPage();
    }
}
```

```
# Scope:          End-to-End Test
# Implementation: PHPUnit + Mink (+ Goutte)

class ExampleTest extends MinkTestCase
{
    public function testHasCorrectTitle()
    {
        $page = $this->visit('http://example.com/');

        $this->assertEquals(
            'Example Domain', $page->find('css', 'title')->getHtml()
        );
    }

    public function testMoreInformationCanBeAccessed()
    {
        $page = $this->visit('http://example.com/');
        $page->clickLink('More information...');

        $this->assertContains(
            'IANA', $page->find('css', 'title')->getHtml()
        );
    }
}
```

```
# Scope:          End-to-End Test
# Implementation: Behat + Mink (+ Goutte)

Feature: Example

Scenario: Accessing an example web page works
    Given I am on "/"
    When I follow "More information..."
    Then I should see "IANA"
```

Scope: Edge-to-Edge Test

- **Does the system as a whole work?**
- As end-to-end as possible but without using real HTTP requests and responses, for instance.

```
# Scope:          Edge-to-Edge
# Implementation: Behat

Feature: Homepage

Scenario:
  Given the default configuration
  When a request to the homepage is made
  Then the response should contain "foo"
```

```
# Scope:          Edge-to-Edge
# Implementation: Behat

use Behat\Behat\Context\SnippetAcceptingContext;

class FeatureContext implements SnippetAcceptingContext {
    private $application;
    private $response;

    /** @Given the default configuration */
    public function theDefaultConfiguration() {
        $this->application = new Application;
    }

    /** @When a request to the homepage is made */
    public function aRequestToTheHomepageIsMade() {
        $this->response = $this->application->run(new Request);
    }

    /** @Then the response should contain :string */
    public function theResponseShouldContainFoo($string) {
        PHPUnit_Framework_Assert::assertTrue($this->response->hasData($string));
    }
}
```

```
# Scope:          Edge-to-Edge
# Implementation: PHPUnit

class HomepageTest extends PHPUnit_Framework_TestCase
{
    public function testHasFoo()
    {
        $application = new Application(/* ... */);
        $request     = new Request(/* ... */);

        $response = $application->run($request);

        $this->assertTrue($response->hasData('foo'));

        // ...
    }
}
```

Role: Integration Test

- **Is our software correctly integrated?**
- Do the units of our software interoperate correctly?
- Do our abstractions over third-party code work?
- Does our communication with other systems work?

```
# Scope:          Integration Test
# Implementation: PHPUnit + DbUnit

class DataMapperTest extends PHPUnit_Extensions_Database_TestCase
{
    public function testObjectCanBePersisted()
    {
        // ...

        $this->assertDataSetsEqual(
            $this->createFlatXmlDataSet('expected.xml');,
            $this->getConnection()->createDataSet(array('actual_table'));
        );
    }

    public function getConnection() { /* ... */ }
    public function getDataSet() { /* ... */ }
}
```

```
class SampleWorkflow
{
    private $backend;
    private $service;

    public function __construct(Backend $backend, Service $service)
    {
        $this->backend = $backend;
        $this->service = $service;
    }

    public function execute(Request $request)
    {
        // ...
        $this->service->doWork(
            $this->backend->getObjectById($request->getValue('id'))
        );
        // ...

        return new Result(/* ... */);
    }

    // ...
}
```

```
# Scope:          Integration Test
# Implementation: PHPUnit

class SampleWorkflowTest extends PHPUnit_Framework_TestCase
{
    public function testWorkflowIntegratesCorrectlyWithBackendAndService()
    {
        $workflow = new SampleWorkflow(new Backend, new Service);

        $this->assertEquals(
            new Result('expected result'),
            $workflow->execute(new Request(array('id' => 2204)))
        );
    }
}
```

```
# Scope:          Integration Test
# Implementation: PHPUnit

class SampleWorkflowTest extends PHPUnit_Framework_TestCase
{
    public function testWorkflowIntegratesCorrectlyWithBackendAndService()
    {
        $service = $this->getMockBuilder('Service')
                        ->enableProxyingToOriginalMethods()
                        ->getMock();

        $service->expects($this->once())
                ->method('doWork');

        $workflow = new SampleWorkflow(new Backend, $service);

        $this->assertEquals(
            new Result('expected result'),
            $workflow->execute(new Request(array('id' => 2204)))
        );
    }
}
```

```
# Scope:          Unit Test
# Implementation: PHPUnit

class SampleWorkflowTest extends PHPUnit_Framework_TestCase
{
    public function testWorkflowExecutesCorrectly()
    {
        $backend = $this->getMockBuilder('Backend')
                        ->getMock();

        $service = $this->getMockBuilder('Service')
                        ->getMock();

        // ... configure $backend and $service test doubles ...

        $workflow = new SampleWorkflow($backend, $service);

        $this->assertEquals(
            new Result('expected result'),
            $workflow->execute(new Request(array('id' => 2204)))
        );
    }
}
```

Role: Unit Test

- **Is our software well crafted?**
- Do the units of our software work correctly in isolation from each other?
- Are the units of our software convenient to work with and easy to test?

Clean Code

"Clean code can be read, and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone."

— Dave Thomas

```
# Scope:          Unit Test
# Implementation: PHPUnit

class CalculatorTest extends PHPUnit_Framework_TestCase
{
    public function testTwoNumbersCanBeAdded()
    {
        $calculator = new Calculator;

        $this->assertEquals(2, $calculator->add(1, 1));
    }
}
```

```
# Scope:          Unit Test
# Implementation: PHPSpec

namespace spec;

use PhpSpec\ObjectBehavior;

class CalculatorSpec extends ObjectBehavior
{
    function it_is_initializable()
    {
        $this->shouldHaveType('Calculator');
    }

    function it_can_add_two_numbers()
    {
        $this->add(1, 1)->shouldReturn(2);
    }
}
```

The Three Dimensions of Testing

- **Role**

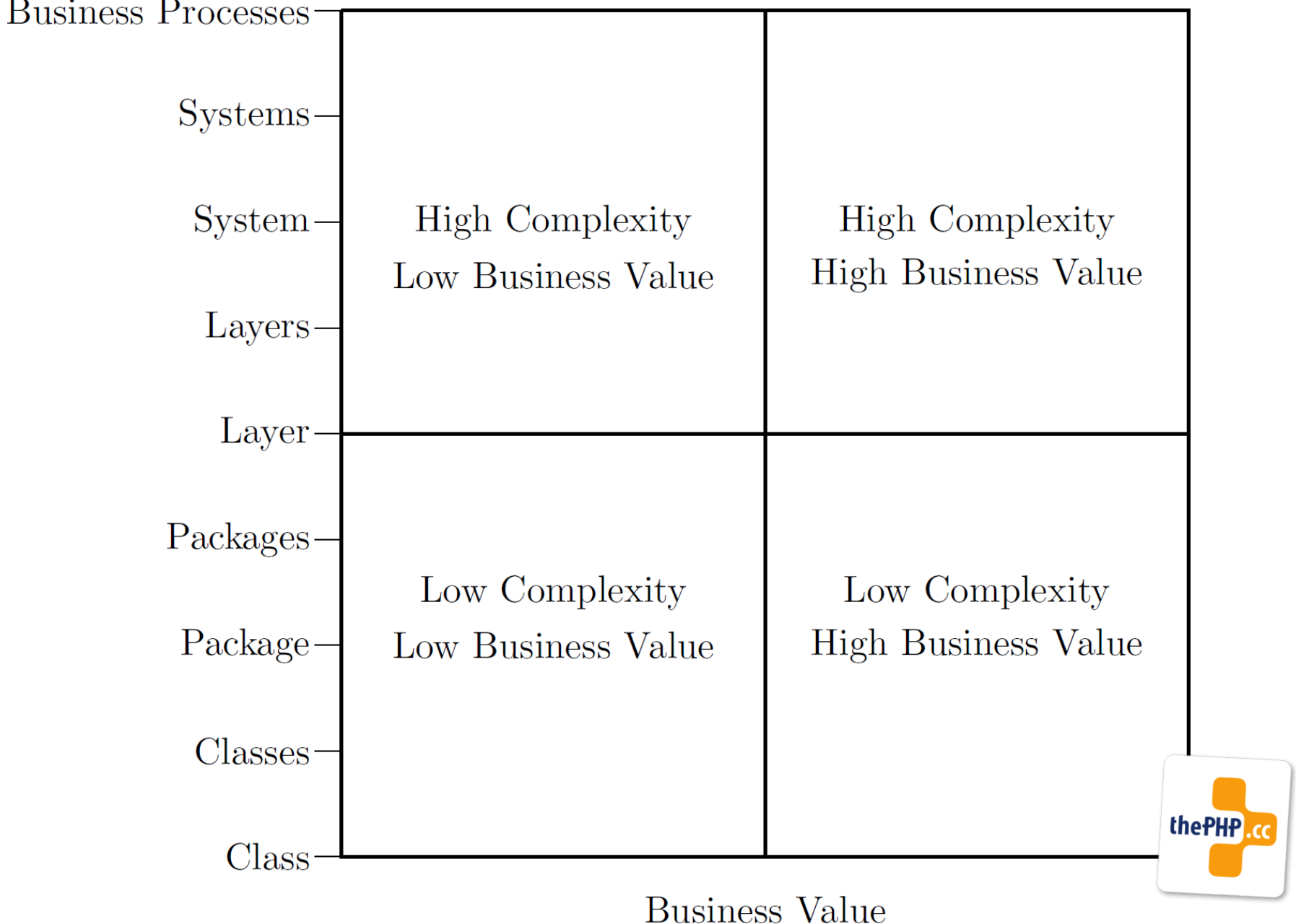
  What do we want to test?

- **Scope**

  What is the minimal environment in which we can test it?

- **Implementation**

  How do we denote the test stimulus and our expectations?


Business Processes
Systems
System
Layers
Layer
Packages
Package
Classes
Class
High Complexity
Low Business Value
High Complexity
High Business Value
Low Complexity
Low Business Value
Low Complexity
High Business Value
Business Value

Effective Testing

- A **high-fidelity** test is one which is very sensitive to defects in the code under test
- A **resilient** test is one that only fails when a breaking change is made to the code under test
- A **high-precision** test tells you exactly where the defect lies

— Rich Martin

External and Internal Quality

"*Running* end-to-end tests tells us about the the external quality of our system, and *writing* them tells us something about how well we [...] understand the domain, but end-to-end tests don't tell us how well we've written the code. *Writing* unit tests gives us a lot of feedback about the quality of our code, and *running* them tell us that we haven't broken any classes [...]"

— Steve Freeman and Nat Pryce

 talks.thePHP.cc

 sebastian@thePHP.cc

 @s_bergmann

sharing experience