Testable Code

PHPBenelux 2013

Benjamin Eberlei (@beberlei)
Tobias Schlitt (@tobySen)

January 24th, 2013

About us

- Benjamin Eberlei
- `benjamin@qafoo.com`
- @beberlei

- Tobias (Toby) Schlitt
- `toby@qafoo.com`
- @tobySen

- Long time PHP professionals
- Open source enthusiast

Working at

Qafoo

passion for software quality

Helping people to create high quality web applications.

`http://qafoo.com`

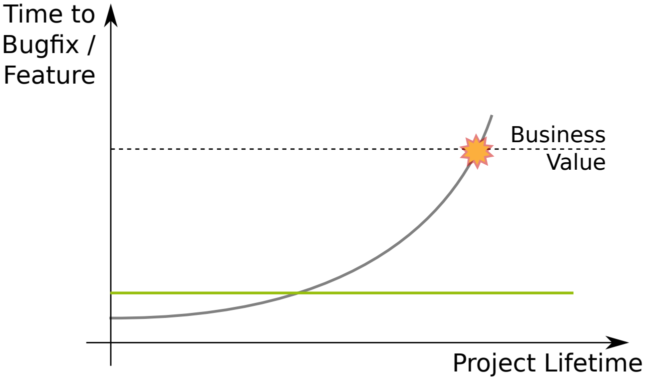
- Expert consulting
- Individual training

From 2013 on incorporating Doctrine 2 & Symfony2 expertise!

Part I

Testing

Why Test?



Outline

Types

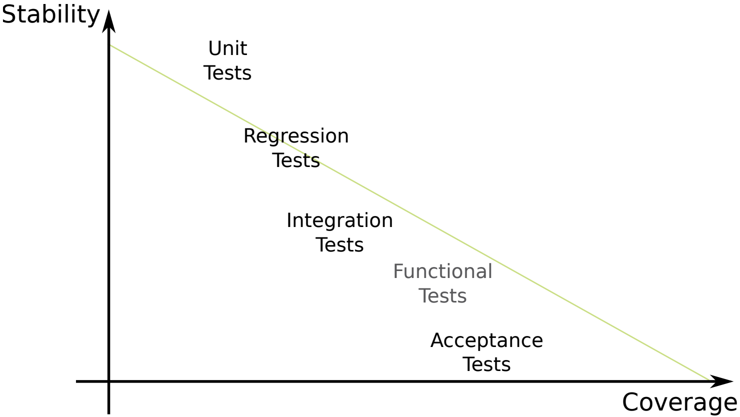
Unit tests

Example

Test methods

- Unit tests
- Integration tests
- Regression tests
- Acceptance tests

Test Stability



Outline

Types

Unit tests

Example

Unit tests

- Purpose
  - Validate functionality
  - Test a single unit of code
  - Avoid regressions
- Applications
  - Verify interfaces (public API)
  - Test bugs dedicatedly
- Benefits
  - Force code modularization
  - Ensures backwards compability
  - Migrate safely

Test Driven Development (TDD)

- Test Driven Development
  - 1) Write (& document) interfaces
  - 2) Write tests
  - 3) Write implementation
- Benefits
  - A lot less defects in code
  - Faster development after a couple of projects
  - More developer satisfaction
  - Less code

Outline

Types

Unit tests

Example

Example

Developing a weather service

Requirements

- Fetch weather for a city
- Relevant data:
  - Condition
  - Temperature
  - Wind
- Be service-agnostic
  - Weather service come and go
  - Data licenses may change
- Log service failures
- Make it possible to add service fallbacks later

What tests do you want?

What types of tests do you desire for the future?

Part II

Testable Code

Outline

Testing issues

Conclusion

The Example

```php
<?php

class WeatherLoader
{
    public function getWeatherForLocation( Location $location )
    {
        $xml = $this->fetchData( $location->city );
        Logger::logDebug( 'Fetched␣XML', $xml );
        return $this->parseData( $xml );
    }
    protected function fetchData( $city )
    {
        $url = sprintf( 'http://...?city=%s', $city );
        return $this->fetchFromUrl( $url );
    }
    protected function parseData( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        $weather->windSpeed = $this->milesToKilometers(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }
    /* ... */
}
```

Issue #1

```php
<?php

class WeatherLoader
{
    public function getWeatherForLocation( Location $location )
    {
        $xml = $this->fetchData( $location->city );
        Logger::logDebug( 'Fetched␣XML', $xml );
        return $this->parseData( $xml );
    }
    protected function fetchData( $city )
    {
        $url = sprintf( 'http://...?city=%s', $city );
        return $this->fetchFromUrl( $url );
    }
    protected function parseData( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        $weather->windSpeed = $this->milesToKilometers(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }
    /* ... */
}
```

Protected to Public

```
<?php

class Weathe
{
    public funct        WeatherForL        ( Location $location )
    {
        $xml = $thi        hData(        tion->city );
        Logger::logD        'Fet        L', $xml );
        return $this->        a        ml );
    }
    public function fetc        city )
    {
        $url = sprintf(        ?city=%s', $city );
        return $this->        r        $url );
    }
    public functio        Data( $
    {
        $weathe        Weather();
        $weath        ditions = $this        eConditions( $xml );
        $weat        ndSpeed = $this->        Kilometers(
                ->parseWindSpeed( $xm
        );
        r        weather;
    }
    /* ... */
}
```

Mocking the Subject

```
<?php

class TestingWeat[illegible]er extend[illegible]herLoader
{
    public function [illegible]ta( [illegible] )
    {
        return parent::[illegible] $city );
    }
    public function pars[illegible]ml )
    {
        return parent[illegible]D[illegible]ml );
    }
    // ...
}
```

Protected to Public

- Exposed functionality will be used
- Creates public API that is hard to change
- Internal dependencies might break

The Real Issue

E_TOO_MANY_RESPONSIBILITIES

The Fix

```php
<?php

class WeatherLoader
{
    public function __construct( WeatherService $service, WeatherParser $parser )
    {
        // ...
    }
    public function getWeatherForLocation( Location $location )
    {
        $data = $this->service->getWeather( $location );
        Logger::logDebug( 'Fetched data', $data );
        return $this->parser->parseData( $data );
    }
}
```

The Fix

- Never test private/protected explicitely
- Test them implicitely . . .
- . . . or change the code

Issue #2

```php
<?php

class WeatherLoader
{
    public function __construct( WeatherService $service, WeatherParser $parser )
    {
        // ...
    }
    public function getWeatherForLocation( Location $location )
    {
        $data = $this->service->getWeather( $location );
        Logger::logDebug( 'Fetched␣data', $data );
        return $this->parser->parseData( $data );
    }
}
```

Test Code in Production

```
<?php

class Logger
{
    public static fun        og          $message, $data )
    {
        // ...
    }
    public static functi              orTesting()
    {
        // ...
    }
}
```

Test Code in Production - continued

```php
<?php

class Logger
{
    public static fun[illegible]get[illegible]ce()
    {
        // ...
    }
    public static functi[illegible]ance( Logger $logger )
    {
        // ...
    }
}
```

The Real Issue

E_STATIC_DEPENDENCY

The Fix

```php
<?php

class WeatherLoader
{
    public function __construct(
        WeatherService $service,
        WeatherParser $parser
        Logger $logger )
    {
        // ...
    }
    public function getWeatherForLocation( Location $location )
    {
        $data = $this->service->getWeather( $location );
        $this->logger->logDebug( 'Fetched data', $data );
        return $this->parser->parseData( $data );
    }
}
```

The Fix

- Never use `static` access
- Always inject dependencies
- Maybe use a dependency injection container (DIC)

Issue #3

```php
<?php

class WeatherService
{
    public function __construct( AppRegistry $registry )
    {
        // ...
    }
    public function getWeather( Location $location )
    {
        $httpClient = $this->appRegistry->get( 'http_client' );
        $url = sprintf( 'http://...?city=%s', $city );
        return $httpClient->get( $url );
    }
}
```

Mocking to Mock

```php
<?php

class Weathe      Test extends PHP      mework_TestCase
{
    public funct      tGetWeather(
    {
        $httpClient      $this->      ck( 'HttpClient' );
        $httpClientMo      pec      is->once() )
            ->method(      )
            /* ... */;

        $appRegistryMock      ->getMock( 'AppRegistry' );
        $appRegistryMoc      $this->once() )
            ->method(      )
            /* ... *

        $service      WeatherSe      appRegistryMock );
        $this->a      quals(
            '.
            $      ->getWeather( new      on() )
        );
    }
}
```

Using Productive Code in Tests

```
<?php

class Weather      Test extends PHP       mework_TestCase
{
    public functi       GetWeather
    {
        $httpClientM       $this        ck( 'HttpClient' );
        $httpClientMo       pe       is->once() )
            ->method(
            /* ... */;

        $appRegistry = n          gistry();
        $appRegistry->            lient', $httpClientMock );

        $service = n     ather      ( $appRegistry );
        $this->ass     ls(
            '...
            $se     getWeather( n       ation() )
        );
    }
}
```

The Real Issue

E_REACHING_THROUGH_OBJECTS

The Fix

```php
<?php

class WeatherService
{
    public function __construct( HttpClient $httpClient )
    {
        // ...
    }
    public function getWeather( Location $location )
    {
        $url = sprintf( 'http://...?city=%s', $city );
        return $this->httpClient->get( $url );
    }
}
```

The Fix

- Do not pull dependencies . . .
- . . . push them
- Do not reach through objects

Issue #4

```php
<?php

class Logger
{
    public function __construct( $fileName )
    {
        // ... error checks ...
        $this->fileHandle = fopen( $fileName, 'a' );
    }
    public function logDebug( $message, $data )
    {
        fwrite(
            $this->fileHandle,
            sprintf(
                "%s␣(%s)\n",
                $message,
                $data
            )
        );
    }
}
```

Accessing File System in Tests

```
<?php

class LoggerTes      ds PHPUnit_Fr      k_TestCase
{
    public functio      gDebugS      ()
    {
        $tmpLogFile =      ->c      FileName();

        $logger = new Lo      pLogFile );
        $logger->logDebu      message.', 'with␣data' );

        $this->assertE
            "Some␣me      (w      )\n",
            file_get      nts(      gFile )
        );
        unlink(      gFile );
    }
}
```

Accessing File System in Tests

- No file access in unit tests (slow!)
- Maintaining temporary files sucks
    - Creating
    - Cleanup
    - System differences

The Virtual File System

```
<?php

class LoggerT          nds PHPUnit_Fran          estCase
{
    public functi          LogDebugSu          )
    {
        vfsStream::          'test
        $logFile = vf          n::          test' ) . '/message.log';

        $logger = new L          gFile );
        $logger->logDebug          ␣message.', 'with␣data' );

        $this->assertT
            vfsStream          ::          t()->hasChild( 'message.log' )
        );
        $this->ass          als(
            "Som          ge.␣(with␣          ",
            file          ontents( $log
        );
    }
}
```

The Virtual File System

- Works, but . . .

The Real Issue

E_HARD_SYSTEM_DEPENDENCY

The Fix

```php
<?php

class Logger
{
    public function __construct( FileHandler $fileHandler )
    {
        $this->fileHandler = $fileHandler;
    }
    public function logDebug( $message, $data )
    {
        $this->fileHandler->write(
            sprintf(
                "%s␣(%s)\n",
                $message,
                $data
            )
        );
    }
}
```

The Fix

- Abstract system dependencies . . .
- . . . as low as possible

Outline

Testing issues

Conclusion

What have we seen?

- Single Responsibility Principle
- Open Close Principle
- Law of Demeter
- Dependency Inversion Principle

Conclusion

Testable Code

↕

Good OOD

SOLID

S	Single Responsibility Principle
O	Open / Close Principle
L	Liskov Substitution Principle
I	Interface Segregation Principle
D	Dependency Inversion Principle

Part III

Getting into Code

Coding Kata

- Very simple tasks to experiment with coding
- Implement code in pairs of two people
  - Person A implements failing Test
  - Person B makes test pass
  - Start over by switching Person A/B
- Push TDD to the extreme limits
  - No not-needed classes
  - No not-needed properties / methods
  - No UI

Requirements

- a game is over when all fields are taken
- a game is over when all fields in a column are taken by a player
- a game is over when all fields in a row are taken by a player
- a game is over when all fields in a diagonal are taken by a player
- a player can take a field if not already taken
- players take turns taking fields until the game is over

Constraints (optional)

- Change the requirements
- No naked primitives
- No conditional statements
- Only four lines per method
- Immutable types only
- Baby Steps
  - Recurring clock (2-5 minutes)
  - Implement one TDD cycle
  - Delete code when not finished after clock

Part IV

Final

Thanks for Listening

Rate this talk: `https://joind.in/7783`

Stay in touch

- Benjamin Eberlei
- `benjamin@qafoo.com`
- @beberlei

- Tobias (Toby) Schlitt
- `toby@qafoo.com`
- @tobySen

Get a training for your team:

`http://qafoo.com`