Testable PHP Applications

Confoo 2013

Tobias Schlitt (@tobySen)

2013-02-27

About me

- Degree in computer sience

About me

- Degree in computer sience
- Proessional PHP since 2000

About me

- Degree in computer sience
- Proessional PHP since 2000
- Open source enthusiast

About me

- Degree in computer sience
- Proessional PHP since 2000
- Open source enthusiast
- Passion for
  - Software Design
  - Automated Testing

Co-founder of

Co-founder of

Support teams mastering challenges in web application development.

http://qafoo.com

Co-founder of

Support teams mastering challenges in web application development.

`http://qafoo.com`

- Expert consulting
- Individual training

Outline

Classification of Tests

Unit Testing

Acceptance Testing

Testability

Build Pipeline

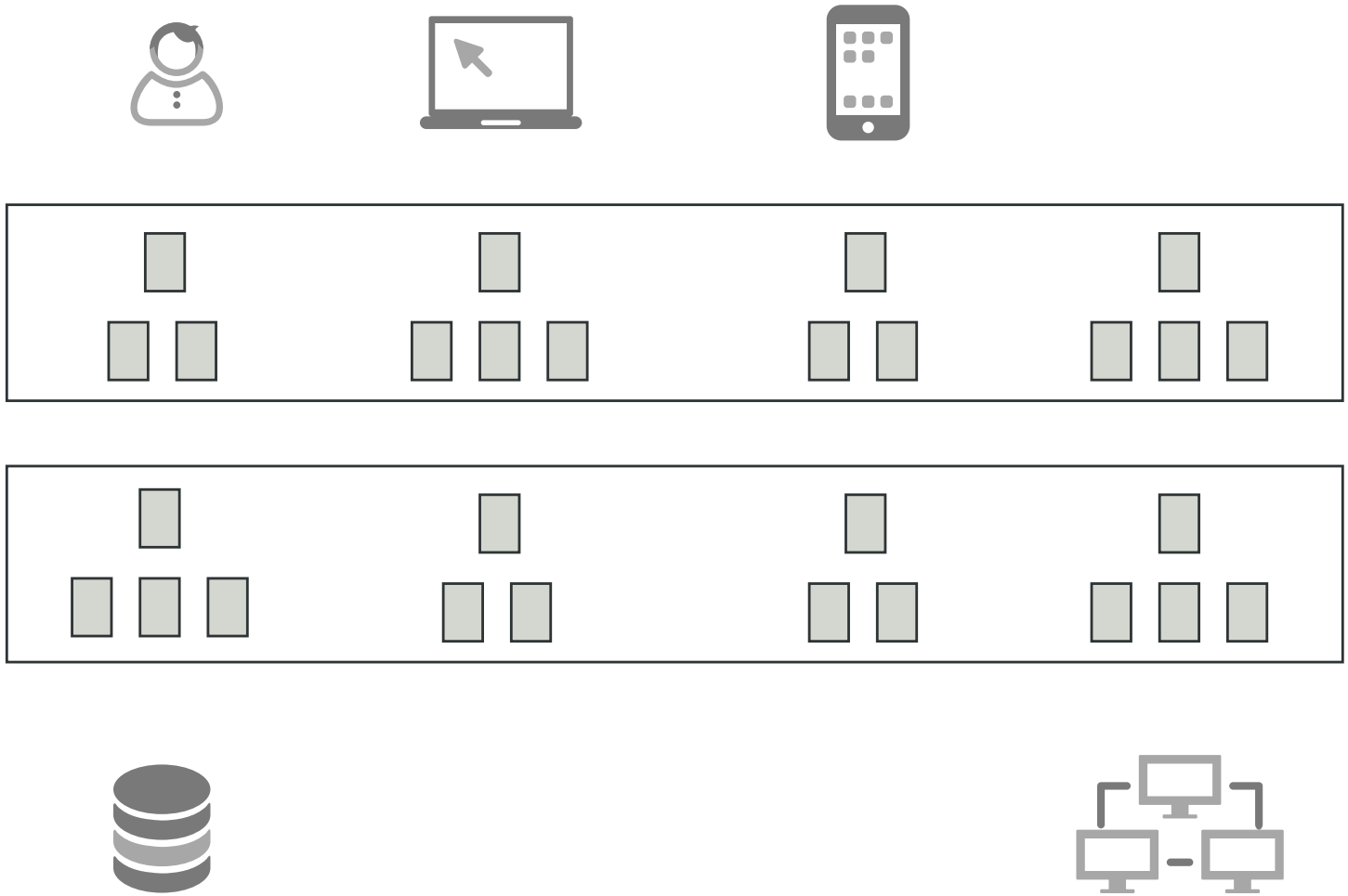
Conclusion

Outline

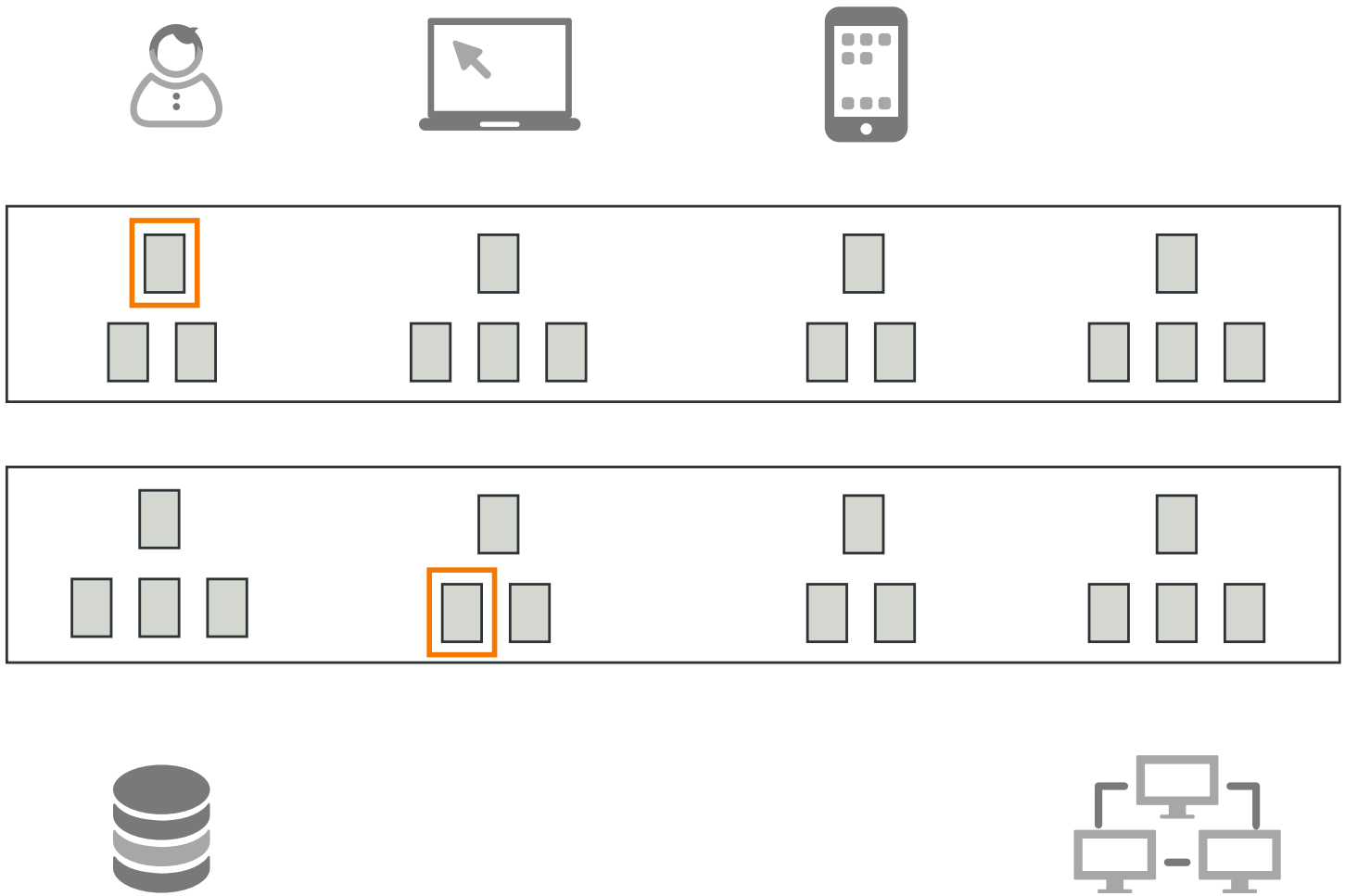
How Do You Test?

- Click & Play
- Unit Test
- Integration Tests
- Acceptance
- Performance Tests

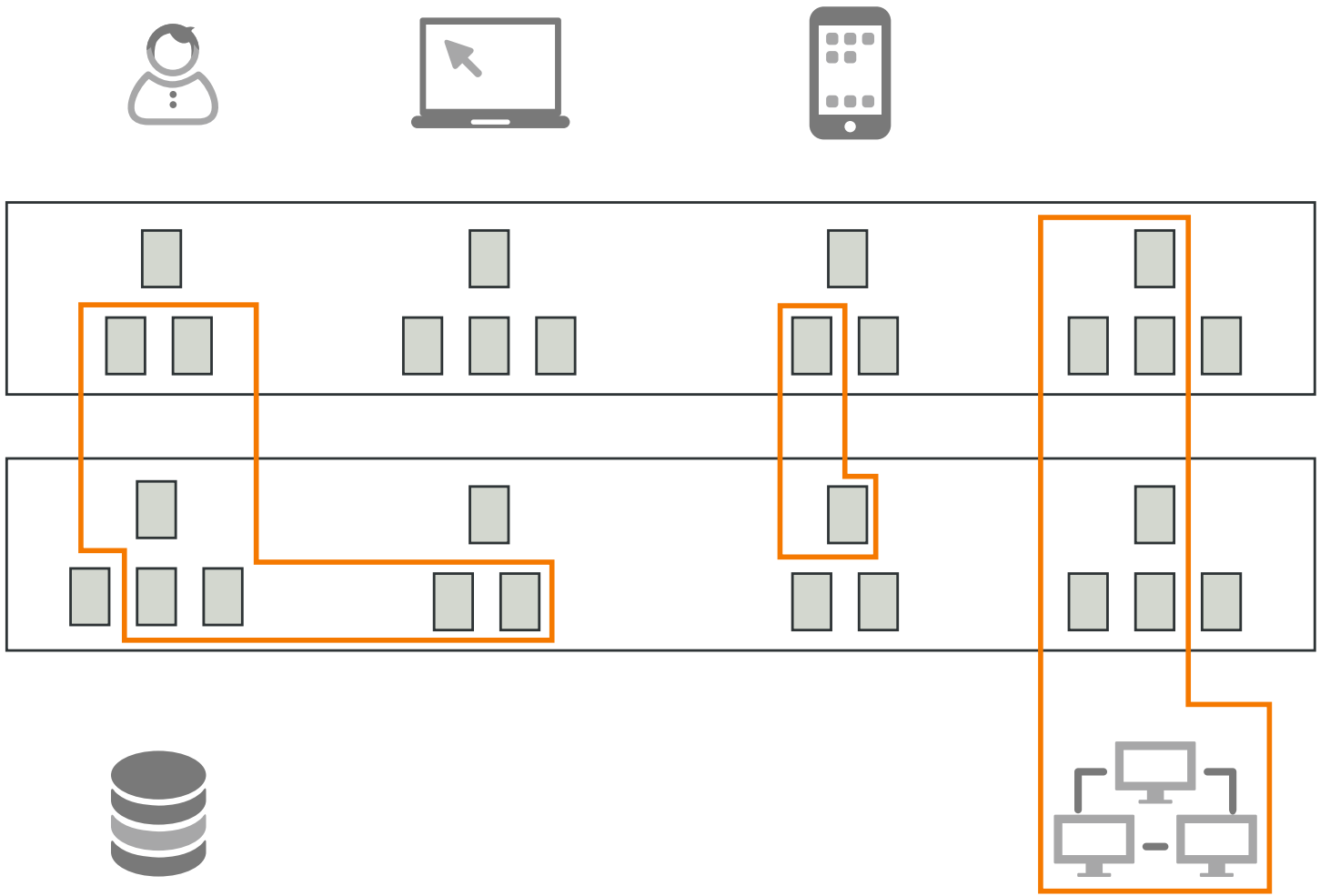
Common Test Mehods



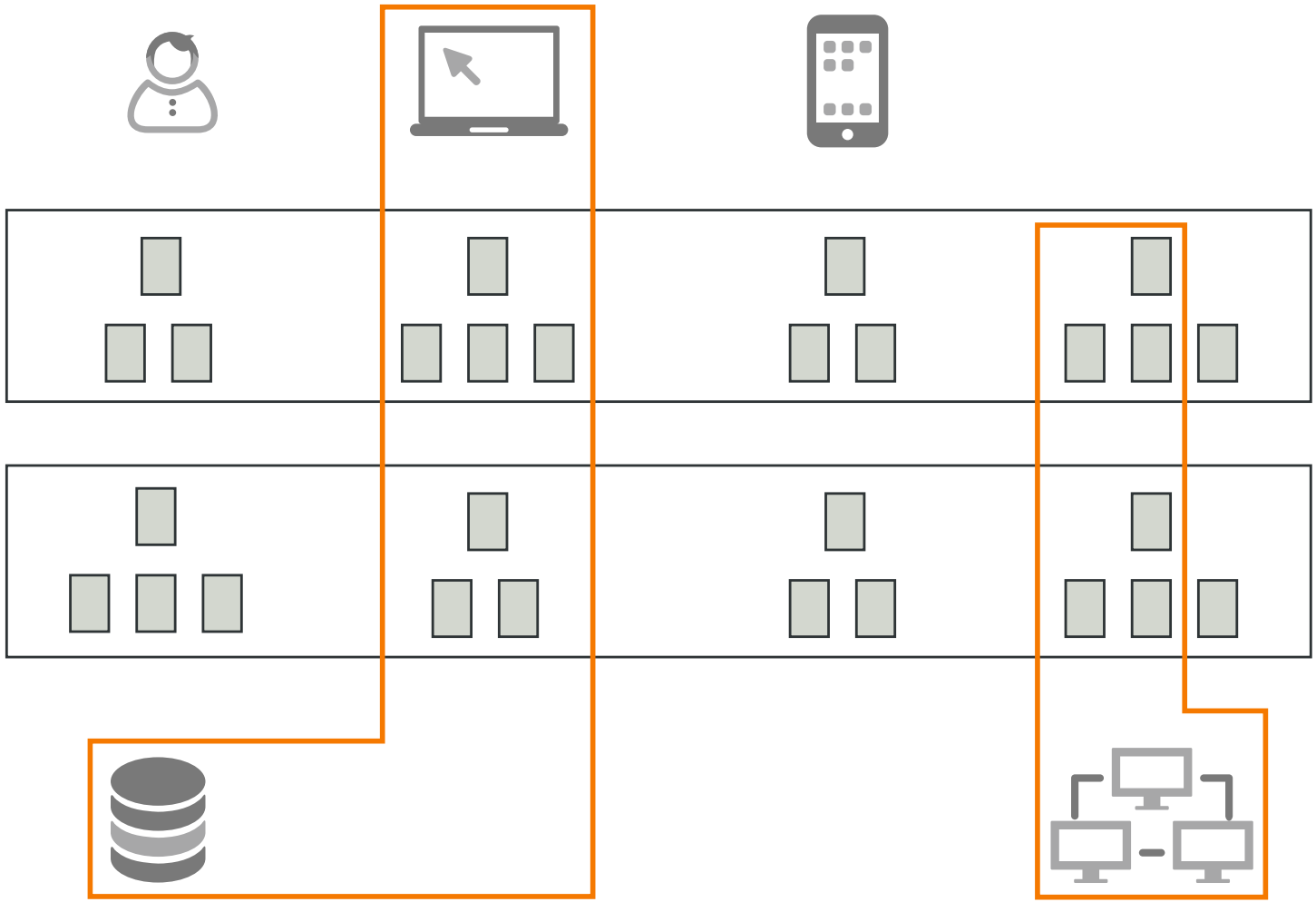
Unit Test



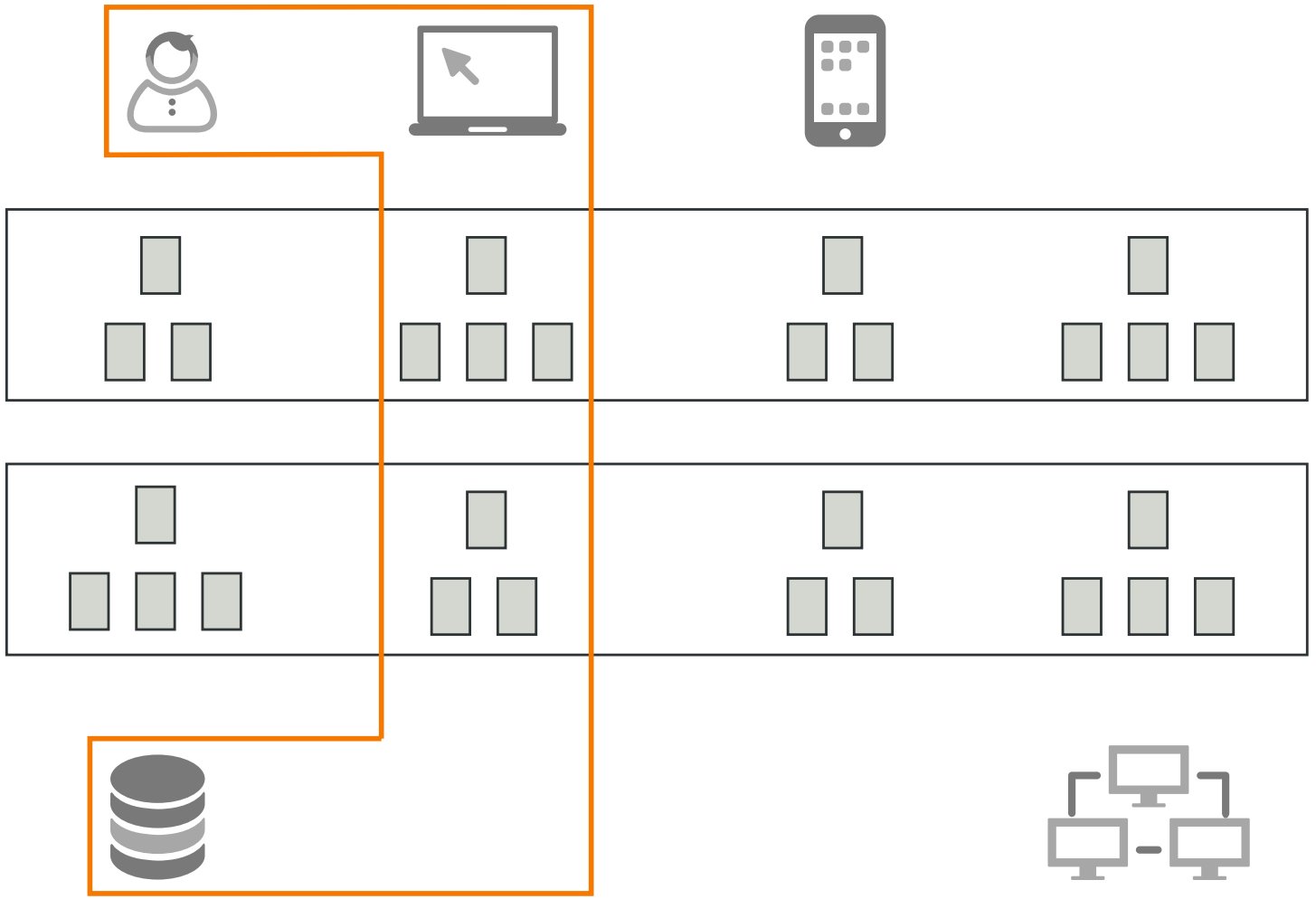
Integration Test



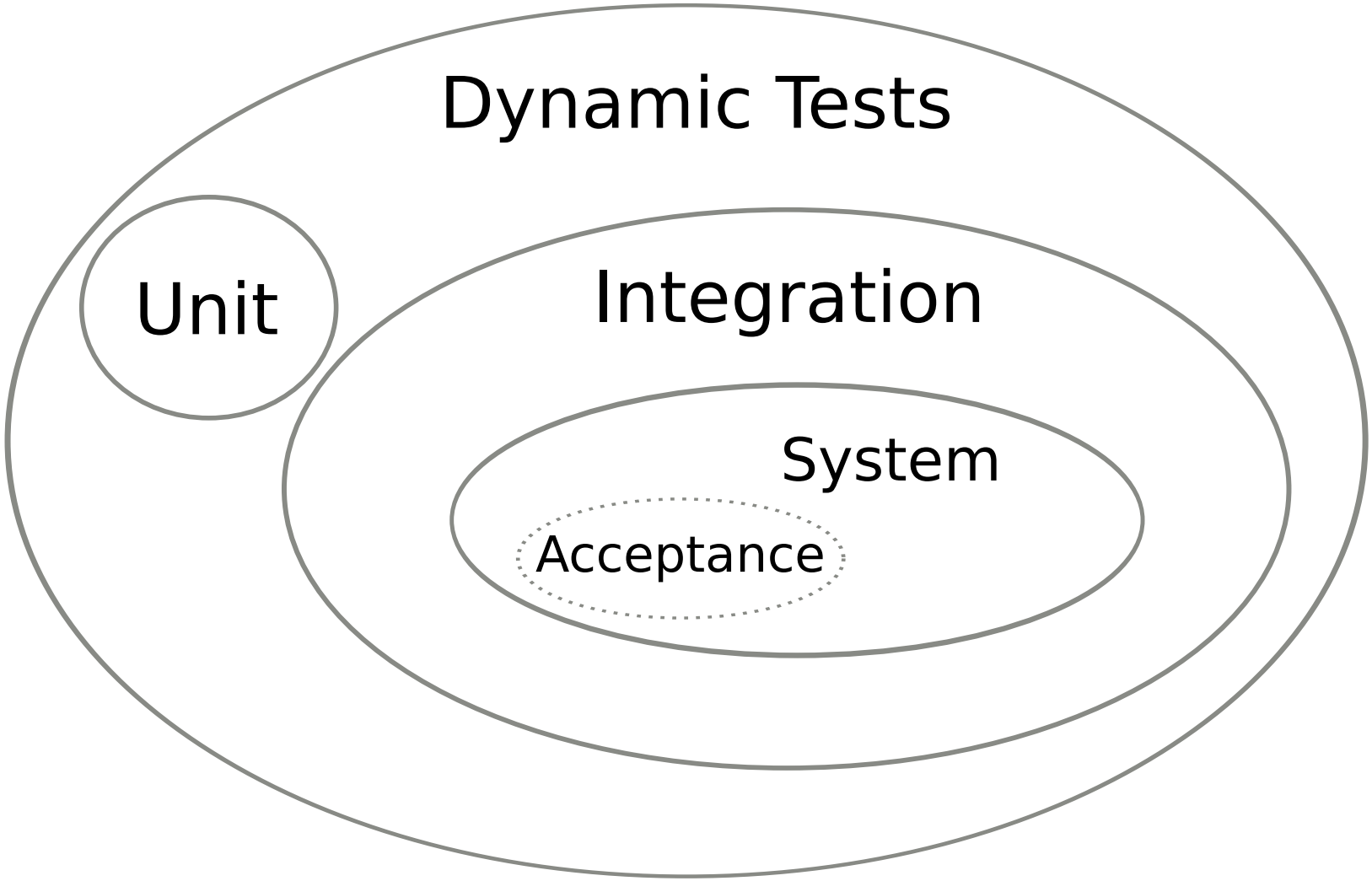
System Test



Acceptance Test



Test Relations



Outline

Unit Testing

- Most popular test method

Unit Testing

- Most popular test method
- xUnit implementation: PHPUnit
  - `http://phpunit.de`
  - Standard in the PHP world
- Test a **single** unit of code
  - Otherwise: Integration Test

Unit Testing

- Most popular test method
- xUnit implementation: PHPUnit
  - `http://phpunit.de`
  - Standard in the PHP world
- Test a **single** unit of code
  - Otherwise: Integration Test
- Best suited to
  - Test algorithms
  - Raise quality of OOD/OOP
  - Start with automated testing

Unit Testing

- Most popular test method
- xUnit implementation: PHPUnit
  - `http://phpunit.de`
  - Standard in the PHP world
- Test a **single** unit of code
  - Otherwise: Integration Test
- Best suited to
  - Test algorithms
  - Raise quality of OOD/OOP
  - Start with automated testing
- Test Driven Development anyone?

Example

```php
class Loader
{
    public function __construct( Service $weatherService, Logger $logger = null )
    {
    }
    public function getWeatherForLocation( Struct\Location $location )
    {
    }
}
```

```php
class LoaderTest extends \PHPUnit_Framework_TestCase
{
    public function testGetWeather()
    {
        $loader = new Loader( $this->getFunctionalServiceMock(), new Logger\Null() );

        $locatedWeather = $loader->getWeatherForLocation(
            new Struct\Location( 'Berlin', 'Germany' )
        );

        $this->assertInstanceOf(
            'Qafoo\\Weather\\Struct\\LocatedWeather',
            $locatedWeather
        );
    }
}
```

Example

```php
class Loader
{
    public function __construct( Service $weatherService, Logger $logger = null )
    {
    }
    public function getWeatherForLocation( Struct\Location $location )
    {
    }
}
```

```php
class LoaderTest extends \PHPUnit_Framework_TestCase
{
    public function testGetWeather()
    {
        $loader = new Loader( $this->getFunctionalServiceMock(), new Logger\Null() );

        $locatedWeather = $loader->getWeatherForLocation(
            new Struct\Location( 'Berlin', 'Germany' )
        );

        $this->assertInstanceOf(
            'Qafoo\\Weather\\Struct\\LocatedWeather',
            $locatedWeather
        );
    }
}
```

Example

```php
class Loader
{
    public function __construct( Service $weatherService, Logger $logger = null )
    {
    }
    public function getWeatherForLocation( Struct\Location $location )
    {
    }
}
```

```php
class LoaderTest extends \PHPUnit_Framework_TestCase
{
    public function testGetWeather()
    {
        $loader = new Loader( $this->getFunctionalServiceMock(), new Logger\Null() );

        $locatedWeather = $loader->getWeatherForLocation(
            new Struct\Location( 'Berlin', 'Germany' )
        );

        $this->assertInstanceOf(
            'Qafoo\\Weather\\Struct\\LocatedWeather',
            $locatedWeather
        );
    }
}
```

Example

```php
class Loader
{
    public function __construct( Service $weatherService, Logger $logger = null )
    {
    }
    public function getWeatherForLocation( Struct\Location $location )
    {
    }
}
```

```php
class LoaderTest extends \PHPUnit_Framework_TestCase
{
    public function testGetWeather()
    {
        $loader = new Loader( $this->getFunctionalServiceMock(), new Logger\Null() );

        $locatedWeather = $loader->getWeatherForLocation(
            new Struct\Location( 'Berlin', 'Germany' )
        );

        $this->assertInstanceOf(
            'Qafoo\\Weather\\Struct\\LocatedWeather',
            $locatedWeather
        );
    }
}
```

Example

```php
class Loader
{
    public function __construct( Service $weatherService, Logger $logger = null )
    {
    }
    public function getWeatherForLocation( Struct\Location $location )
    {
    }
}
```

```php
class LoaderTest extends \PHPUnit_Framework_TestCase
{
    public function testGetWeather()
    {
        $loader = new Loader( $this->getFunctionalServiceMock(), new Logger\Null() );

        $locatedWeather = $loader->getWeatherForLocation(
            new Struct\Location( 'Berlin', 'Germany' )
        );

        $this->assertInstanceOf(
            'Qafoo\\Weather\\Struct\\LocatedWeather',
            $locatedWeather
        );
    }
}
```

Example

```php
class Loader
{
    public function __construct( Service $weatherService, Logger $logger = null )
    {
    }
    public function getWeatherForLocation( Struct\Location $location )
    {
    }
}
```

```php
class LoaderTest extends \PHPUnit_Framework_TestCase
{
    public function testGetWeather()
    {
        $loader = new Loader( $this->getFunctionalServiceMock(), new Logger\Null() );

        $locatedWeather = $loader->getWeatherForLocation(
            new Struct\Location( 'Berlin', 'Germany' )
        );

        $this->assertInstanceOf(
            'Qafoo\\Weather\\Struct\\LocatedWeather',
            $locatedWeather
        );
    }
}
```

Unit / Integration Testing Issues

- Too many responsibilities
- Un-replaceable dependencies
- Too many dependencies

Outline

Acceptance Testing

- Normal: client
- Can partially be automated

Acceptance Testing

- Normal: client
- Can partially be automated
- Selenium / Sahi

Acceptance Testing

- Normal: client
- Can partially be automated
- Selenium / Sahi
- Promising approach: Behat
  - `http://behat.org/`
  - Human readable specifications
  - Partially formalized (Gherkin)
  - Executed as system tests

Acceptance Testing

- Normal: client
- Can partially be automated
- Selenium / Sahi
- Promising approach: Behat
  - `http://behat.org/`
  - Human readable specifications
  - Partially formalized (Gherkin)
  - Executed as system tests
- Best suited for
  - Customer communication
  - Complex business rules
  - Integration testing

Acceptance Testing

- Normal: client
- Can partially be automated
- Selenium / Sahi
- Promising approach: Behat
  - `http://behat.org/`
  - Human readable specifications
  - Partially formalized (Gherkin)
  - Executed as system tests
- Best suited for
  - Customer communication
  - Complex business rules
  - Integration testing
- Behavior Driven Development (BDD) anyone?

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Example

```
Feature:
  BDD tests for the eZ Publish Section API

  Scenario: Admin user can delete a section
    Given I have a section with name "MySection" and identifier "MySectionKey"
      And I am logged in as user "admin"
     When I delete section with identifier "MySectionKey"
     Then I expect no section for identifier "MySectionKey" exists
```

```
class FeatureContext extends BehatContext
{
    /**
     * @Given /^I have a section with name "([^"]+)" and identifier "([^"]+)"$/
     */
    public function iHaveASectionWithNameAndIdentifier( $name, $identifier )
    {
        $sectionService = $this->repository->getSectionService();

        $sectionCreate             = $sectionService->newSectionCreateStruct();
        $sectionCreate->name       = $name;
        $sectionCreate->identifier = $identifier;

        $this->runtimeData[$identifier] = $sectionService->createSection($sectionCreate);
    }
}
```

Acceptance Testing Issues

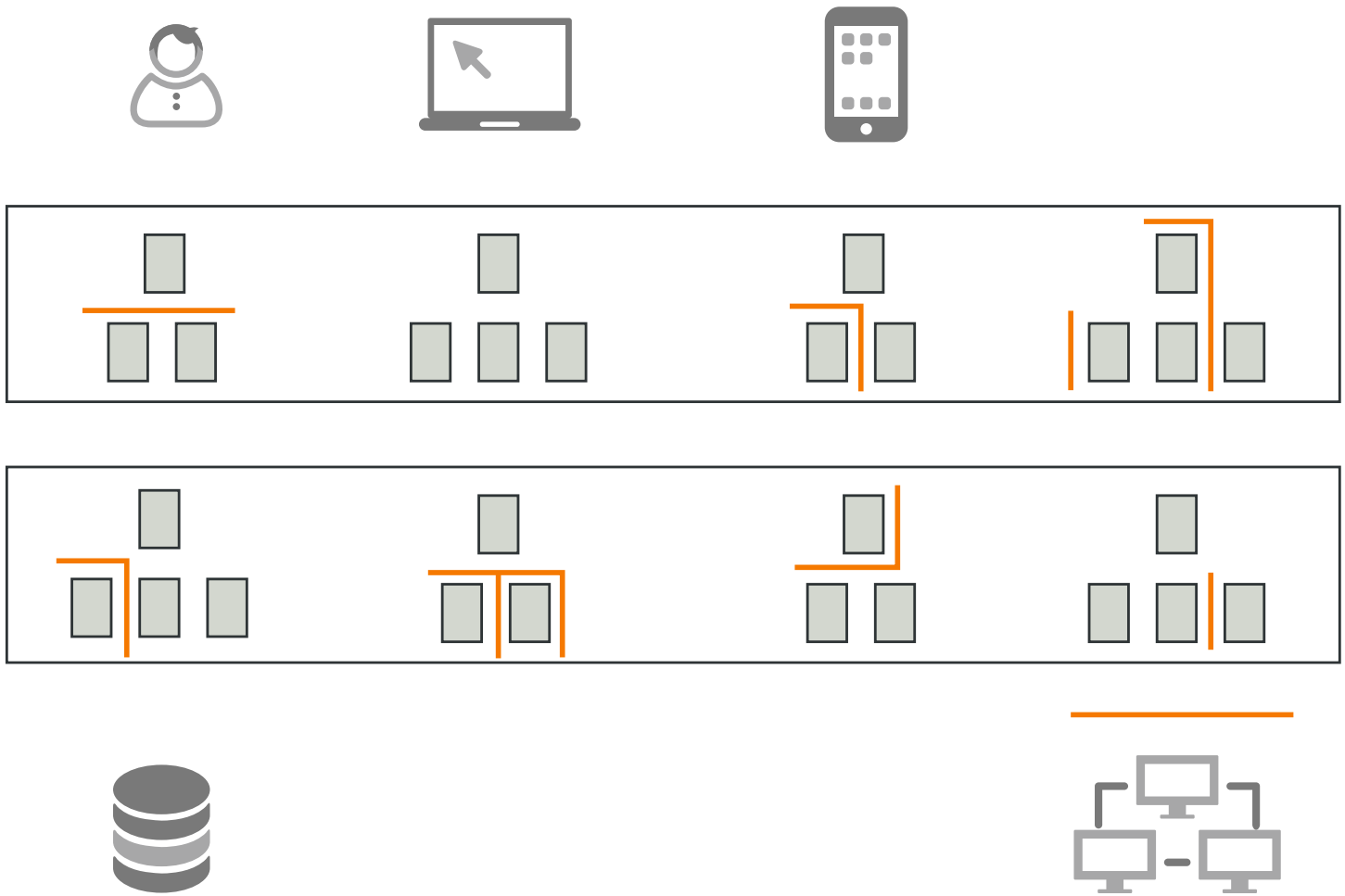
- Slow
  - Integration test are by definition
  - Sensible slicing points can help
- System not machine accesible
- Development overhead
  - if not done carefully

Outline

Testability Issues

- Tight coupling
- Too many responsibilities

Slicing your Application



Testable Code

- Single Responsibility per
  - Module
  - Class
  - Method
- Dependency Inversion
  - Provide cut points
  - Code against abstractions
  - Dependency injection
- Law of Demeter
  - Don’t “reach through” objects

Single Responsibility

```php
<?php

class GoogleWeatherService
{
    public function getWeatherForLocation( Location $location )
    {
        $xml     = $this->getData( $location );
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }

    /* ... */
}
```

Single Responsibility

```php
<?php

class GoogleWeatherService
{
    public function getWeatherForLocation( Location $location )
    {
        $xml     = $this->getData( $location );
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }

    /* ... */
}
```

Single Responsibility

```php
<?php

class GoogleWeatherService
{
    public function getWeatherForLocation( Location $location )
    {
        $xml     = $this->getData( $location );
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }

    /* ... */
}
```

Single Responsibility

```php
<?php

class GoogleWeatherService
{
    public function getWeatherForLocation( Location $location )
    {
        $xml     = $this->getData( $location );
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }

    /* ... */
}
```

Single Responsibility

```php
<?php

class GoogleWeatherService
{
    public function getWeatherForLocation( Location $location )
    {
        $xml     = $this->getData( $location );
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
        return $weather;
    }

    /* ... */
}
```

Single Responsibility Principle

```php
<?php

class GoogleWeatherService
{
    public function __construct(
        HttpClient $client, GoogleDataParser $parser )
    { /* ... */ }

    public function getWeatherForLocation( Location $location )
    {
        $data = $this->client->get( sprintf(
            'http://.../?city=%s',
            $location->city
        ) );
        return $this->parser->parseWeather( $data );
    }
}
```

Single Responsibility Principle

```
1  <?php
2
3  class GoogleWeatherService
4  {
5      public function __construct(
6          HttpClient $client, GoogleDataParser $parser )
7      { /* ... */ }
8
9      public function getWeatherForLocation( Location $location )
10     {
11         $data = $this->client->get( sprintf(
12             'http://.../?city=%s',
13             $location->city
14         ) );
15         return $this->parser->parseWeather( $data );
16     }
17 }
```

Single Responsibility Principle

```php
<?php

class GoogleWeatherService
{
    public function __construct(
        HttpClient $client, GoogleDataParser $parser )
    { /* ... */ }

    public function getWeatherForLocation( Location $location )
    {
        $data = $this->client->get( sprintf(
            'http://.../?city=%s',
            $location->city
        ) );
        return $this->parser->parseWeather( $data );
    }
}
```

Single Responsibility

- One responsibility per class
- Separation of concerns

Dependency Inversion

```php
<?php

class WeatherLoader
{
    public function __construct(
        GoogleWeatherService $weatherService, FileLogger $logger )
    {
        $this->weatherService = $weatherService;
        $this->logger         = $logger;
    }
    public function getWeatherForLocation( Location $location )
    {
        // ...
        $this->logger->log( 'Some log message.' );
        // ...
        $this->logger->writeToFile();
    }
}
```

Dependency Inversion

```php
<?php

class WeatherLoader
{
    public function __construct(
        GoogleWeatherService $weatherService, FileLogger $logger )
    {
        $this->weatherService = $weatherService;
        $this->logger         = $logger;
    }
    public function getWeatherForLocation( Location $location )
    {
        // ...
        $this->logger->log( 'Some log message.' );
        // ...
        $this->logger->writeToFile();
    }
}
```

Dependency Inversion

```php
<?php

class WeatherLoader
{
    public function __construct(
        GoogleWeatherService $weatherService, FileLogger $logger )
    {
        $this->weatherService = $weatherService;
        $this->logger         = $logger;
    }
    public function getWeatherForLocation( Location $location )
    {
        // ...
        $this->logger->log( 'Some log message.' );
        // ...
        $this->logger->writeToFile();
    }
}
```

Dependency Inversion

```php
<?php

class WeatherLoader
{
    public function __construct(
        WeatherService $weatherService, Logger $logger )
    {
        $this->weatherService = $weatherService;
        $this->logger         = $logger;
    }
    public function getWeatherForLocation( Location $location )
    {
        // ...
        $this->logger->log( 'Some log message.' );
        // ...
    }
}
```

Dependency Inversion

- Aloways code against abstractions
- Abstraction give you cutting points

Dependency Inversion

- Aloways code against abstractions
- Abstraction give you cutting points
- Abstractions hide complexity
- Depend on interfaces, not realizations

Dependency Inversion

- Aloways code against abstractions
- Abstraction give you cutting points
- Abstractions hide complexity
- Depend on interfaces, not realizations
- Dependency inversion anyone?

Law of Demeter

```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 );
    }
}
```

Law of Demeter

```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 );
    }
}
```

Law of Demeter

```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 );
    }
}
```

Law of Demeter

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

Outline

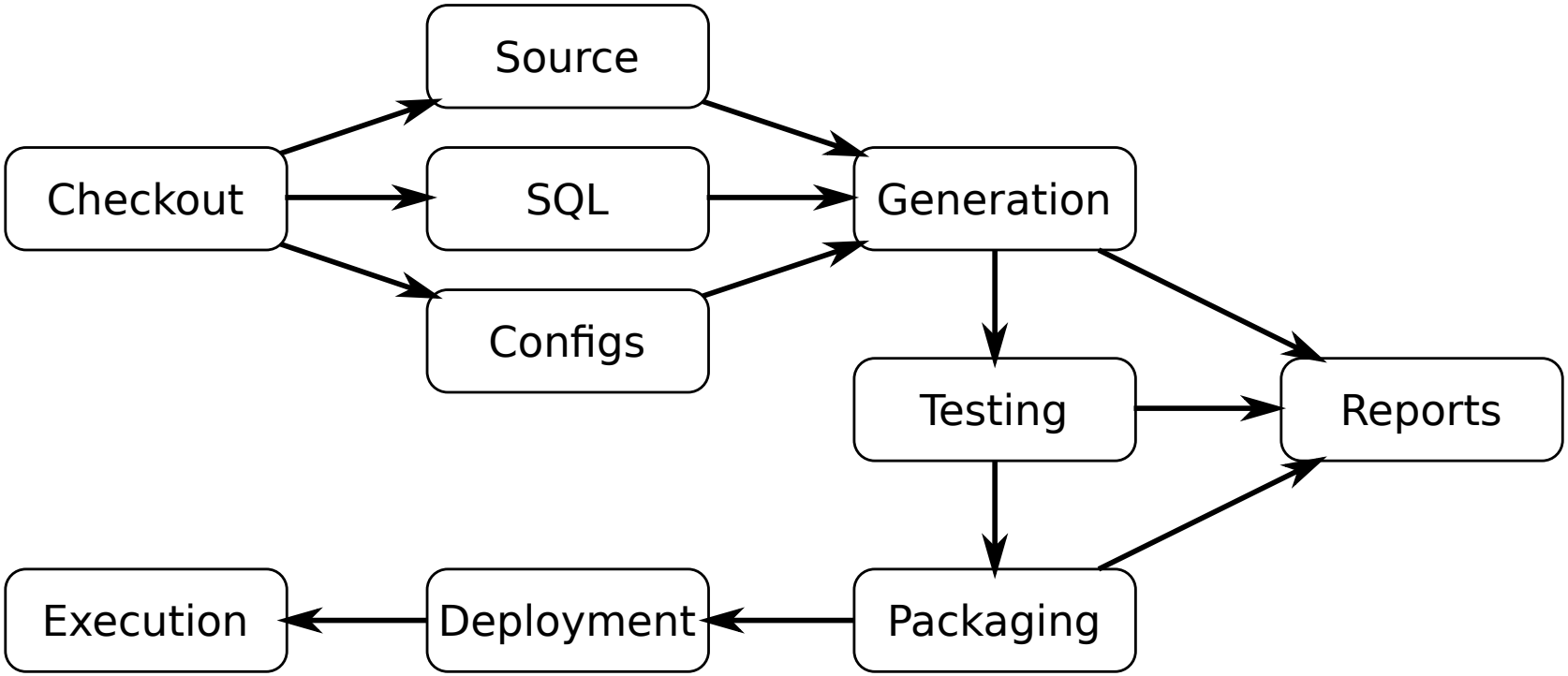
Software Metrics

- Find weak spots
- Find high impact code
- Watch change rate over time
- Watch quality over time

Software Metrics

- PDepend
  - `http://pdepend.org/`
  - Measure complexity / dependencies
- PHPMD
  - `http://phpmd.org/`
  - Detect metric violations
- PHP_CodeSniffer
  - `http://pear.php.net/PHP_CodeSniffer`
  - Enforce coding style

Build Process



Shameless Plug

Ant Build Commons

`http://abc.tools.qafoo.com/`

Outline

Conclusion

- Test mixture

Conclusion

- Test mixture
- Slicing your application

Conclusion

- Test mixture
- Slicing your application
- Good OO design

Conclusion

- Test mixture
- Slicing your application
- Good OO design
- Build pipeline

Thanks for Listening

Rate this talk: `https://joind.in/7852`

Thanks for Listening

Rate this talk: `https://joind.in/7852`

Stay in touch

- Tobias Schlitt
- toby@qafoo.com
- @tobySen / @qafoo

Rent a web quality expert:
`http://qafoo.com`