php·2012
zendcon
OCTOBER 22-25, 2012

Behavioral Driven Development with Behat and Zend Framework 2

by your friend:

Ryan Weaver
@weaverryan

KnpLabs

Who is this dude?

- The Symfony “Docs” guy
- KnpLabs US - Symfony & Behat consulting, training, and Kumbaya
- Writer for KnpUniversity.com screencasts
- Husband of the much more talented @leannapelham

@weaverryan

knplabs.com
github.com/weaverryan

KnpLabs

php·2012 zendcon

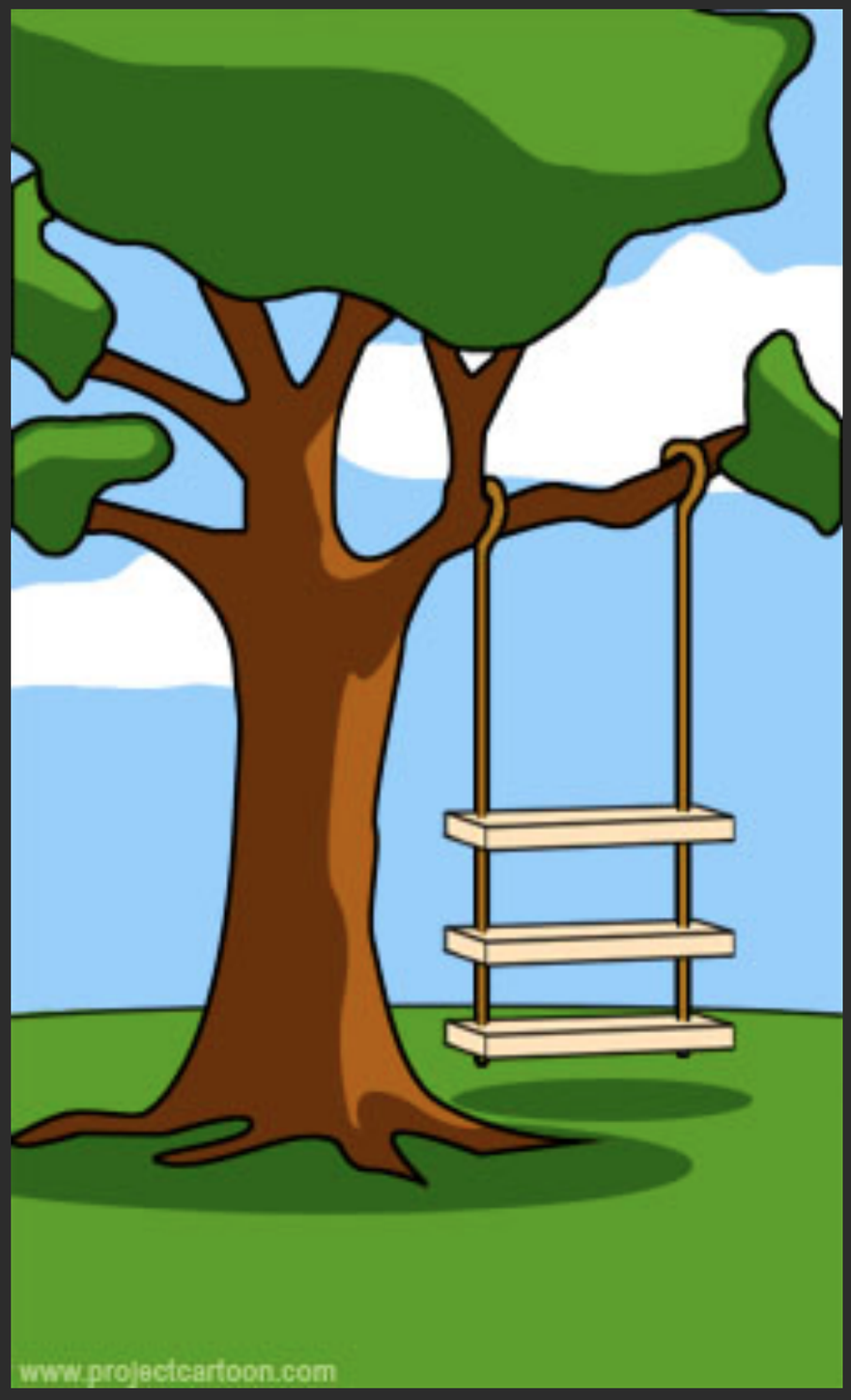
Intro

Plan, Work, Miscommunicate, Panic, Put out Fires, Repeat!

(aka Project Work)

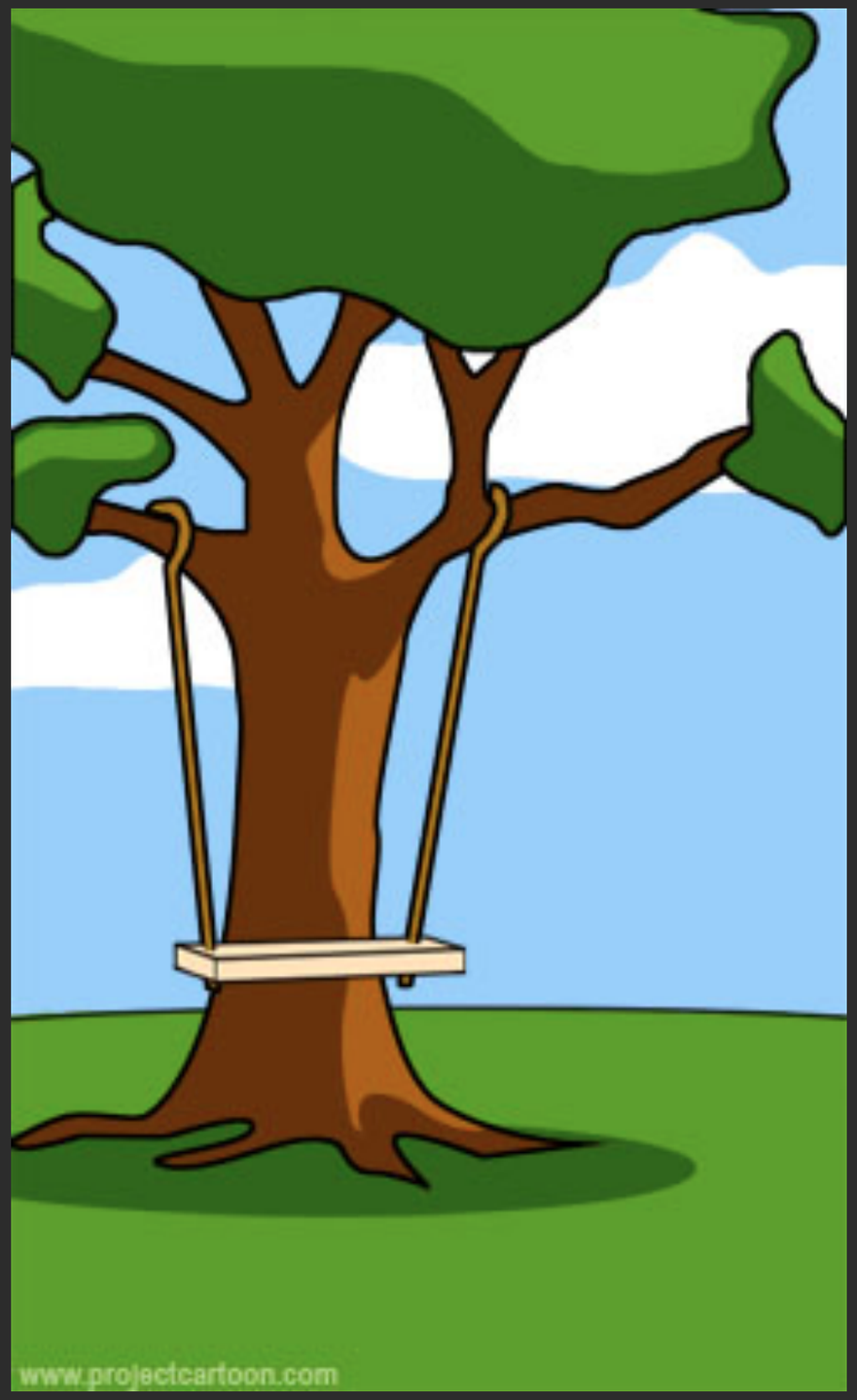
http://www.flickr.com/photos/lifeontheedge/416514144/

The Typical Project



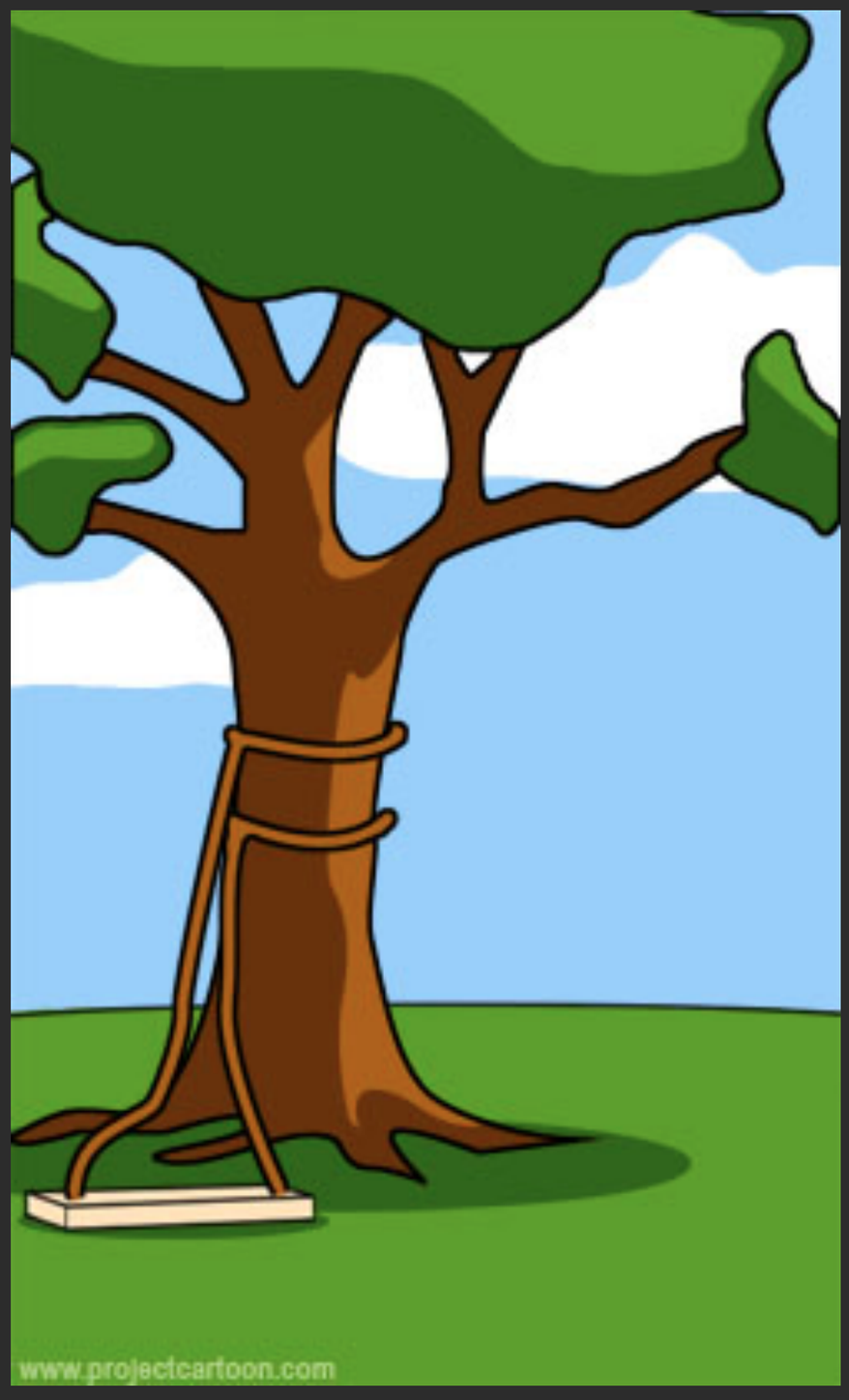
How the customer explained it

http://www.projectcartoon.com



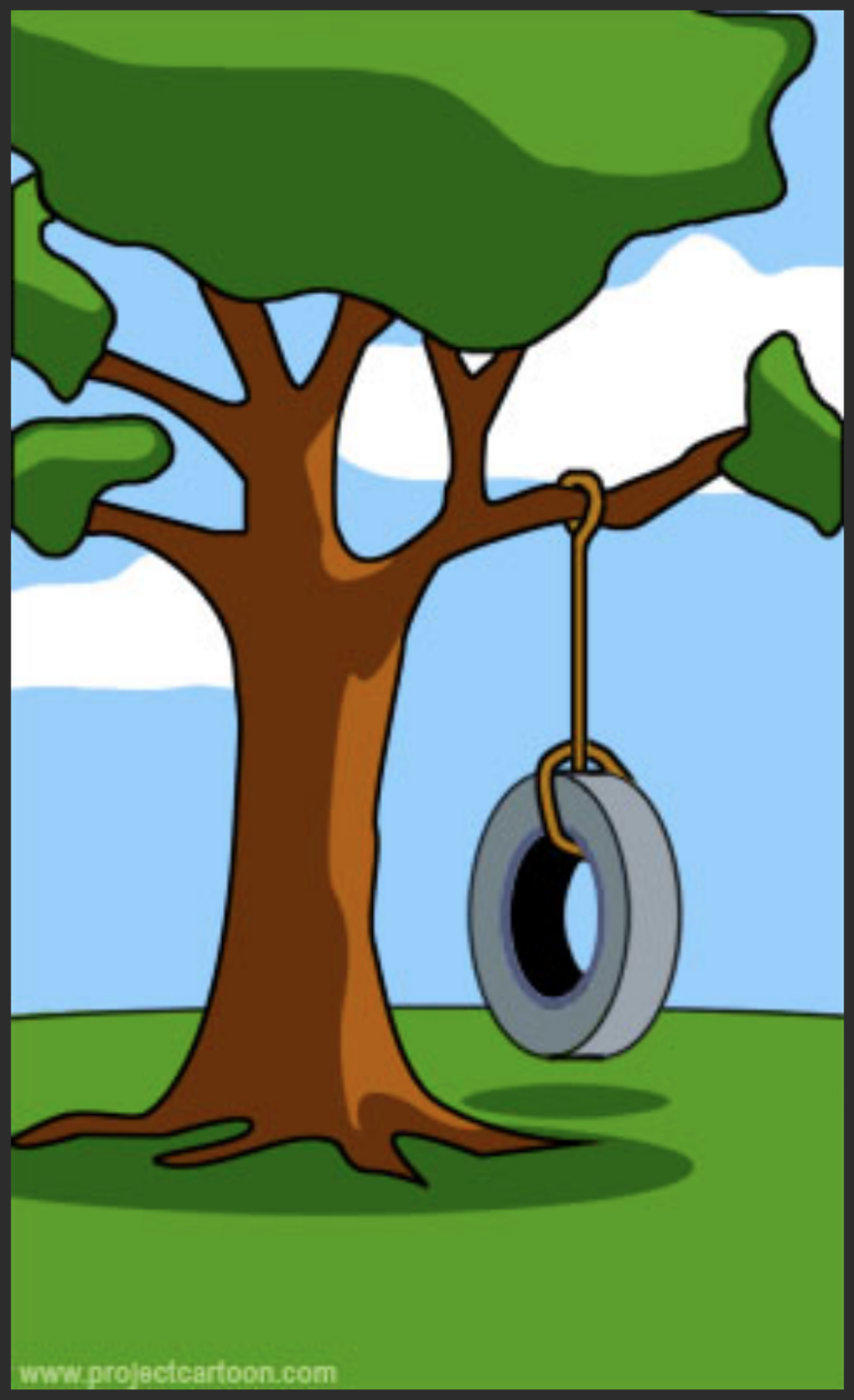
How the project leader understood it

http://www.projectcartoon.com



How the programmer wrote it

http://www.projectcartoon.com



What the customer really needed

http://www.projectcartoon.com

… and my personal favorite

What the beta testers received

http://www.projectcartoon.com

Computer Science?

Where it breaks down...

Different roles, different languages

@weaverryan

Your code and business values may not align

@weaverryan

I've just dreamt up this cool new feature that we should implement!

Why? *Because it's cool!*

Over-planning, under-planning, planning?

@weaverryan

Can you make me a plan to build a...

giant-crazy scary bridge!?

… and I need that plan today…

... I have no other details ...

... and I need a cost estimate ...

over-planning:

* over-budget before you start
* detailed plans for features you won't end up needing

under-planning:

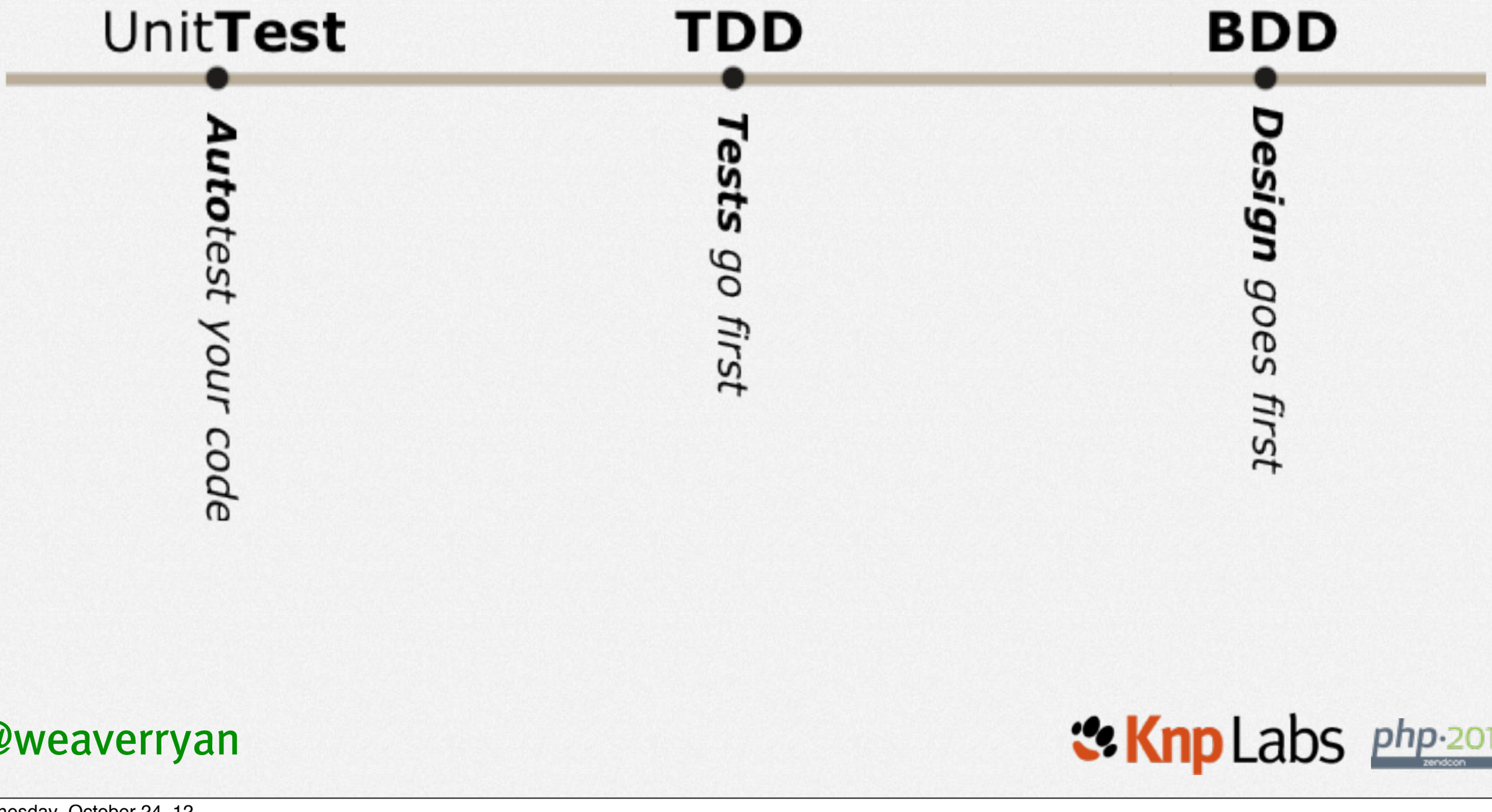
* maybe the feature doesn't have any business value?

* a lot of hidden complications

Act 1

Getting down with BDD

Evolution of Test-Driven Development



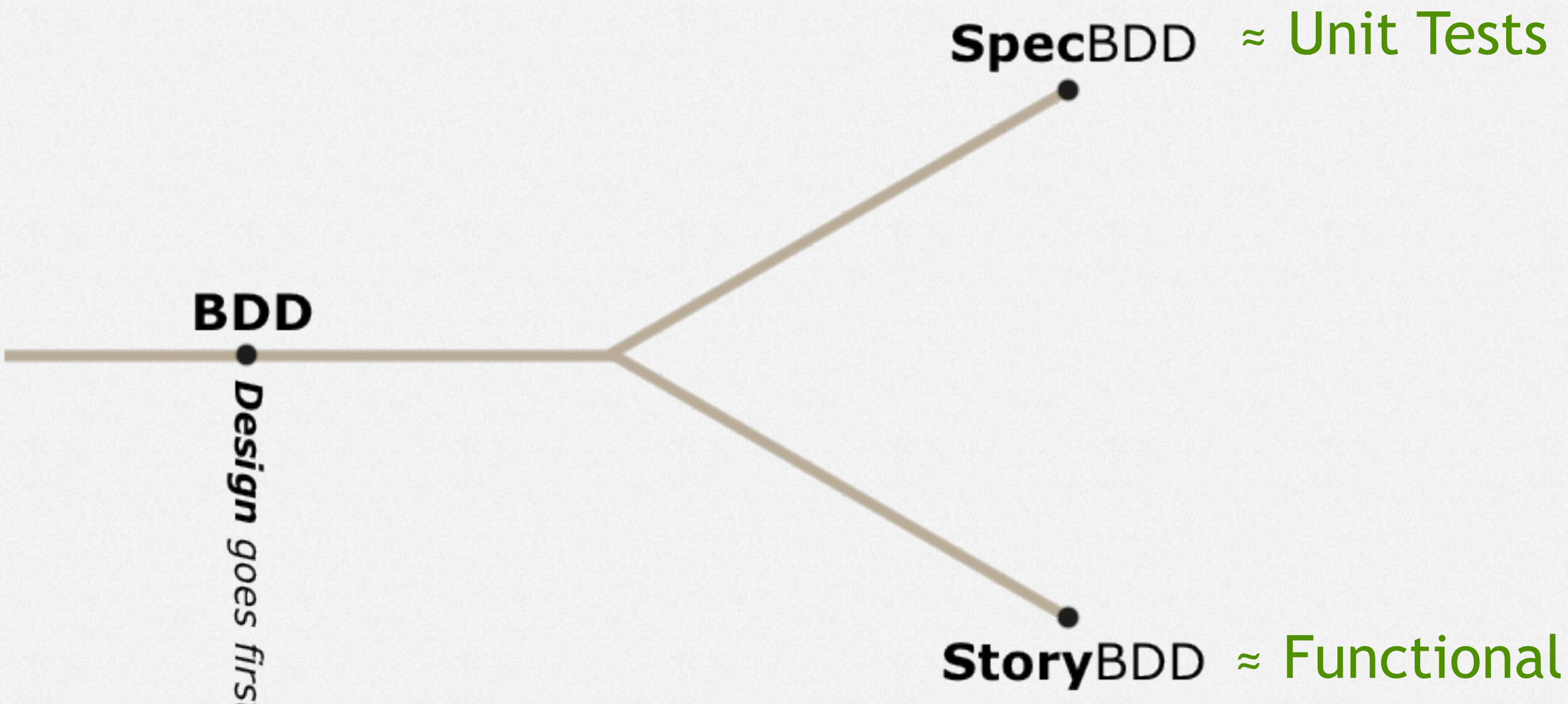
@weaverryan

KnpLabs

php·2012 zendcon

“Behaviour” is a more useful word, than “test”

The two BDD Styles



@weaverryan

Specification BDD

http://www.phpspec.net

Scenario-oriented BDD
(Story BDD)

Let’s create a single vocabulary

Fix Communication between...

- product **owners**
- business **analysts**
- project **managers**
- **developers**
- **testers**

@weaverryan

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

Act 2

Gherkin

A project consists of many features

Gherkin

==

a structure language to describe a feature

```
Feature: {custom_title}
  In order to {A}
  As a {B}
  I need to {C}
```

- **{A}** - the benefit or **value** of the feature
- **{B}** - the **role** (or person) who will benefit
- **{C}** - short feature **description**

> The person “writing” this feature - the “I”

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

read news in French

```
Feature: I18n
  In order to read news in french
  As a french user
  I need to be able to switch locale
```

read news in French

The **business** value

```
Feature: I18n
  In order to read news in french
  As a french user
  I need to be able to switch locale
```

read news in French

The **person** who benefits

+

The “author” of this feature

```
Feature: I18n
  In order to read news in french
  As a french user
  I need to be able to switch locale
```

read news in French

Description of the feature, the action the person will take

```
Feature: I18n
  In order to read news in french
  As a french user
  I need to be able to switch locale
```

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

prioritize...

1) Feature: News admin panel

2) Feature: I18n

3) Feature: News list API

Solution

1. **Define** business value for the features
2. **Prioritize** features by their business value
3. **Describe** them with readable scenarios
4. And only then - **implement** them

```
Feature: News admin panel
  In order to maintain a list of news
  As a site administrator
  I need to be able to edit news

  Scenario: List available articles
    Given there are 5 news articles
    And I am on the "/admin" page
    When I click "News Administration"
    Then I should see 5 news articles

  Scenario: Add new article
    Given I am on the "/admin/news" page
    When I click "new article"
    And I fill in "title" with "Learned BDD today"
    And I press "save"
    Then I should see "A new article have been added"
```

Scenarios

```
Scenario: Add new article
  Given I am on the "/admin/news" page
  When I click "new article"
  And I fill in "title" with "Learned BDD today"
  And I press "save"
  Then I should see "A new article have been added"
```

Given

Defines the initial state of the system for the scenario

Scenarios

```
Scenario: Add new article
  Given I am on the "/admin/news" page
  When I click "new article"
  And I fill in "title" with "Learned BDD today"
  And I press "save"
  Then I should see "A new article have been added"
```

When

Describes the action taken by the person/role

Scenarios

```
Scenario: Add new article
  Given I am on the "/admin/news" page
  When I click "new article"
  And I fill in "title" with "Learned BDD today"
  And I press "save"
  Then I should see "A new article have been added"
```

Then

Describes the observable system state after the action has been performed

Scenarios

```
Scenario: Add new article
  Given I am on the "/admin/news" page
  When I click "new article"
  And I fill in "title" with "Learned BDD today"
  And I press "save"
  Then I should see "A new article have been added"
```

And/But

Can be added to create multiple **Given/When/Then** lines

Gherkin

gives us a consistent language

for describing features and

their scenarios

… now let’s turn them into tests!

Act 3

Those tests ain't gonna write themselves,

Behatch

From the t-shirt: http://bit.ly/behatch-t

Having a standard way of describing features is cool...

... executing those sentences as functional tests is just awesome

What is Behat?

Behat does one simple thing:

It Maps Each step to a PHP Callback**

Behat "executes" your scenarios, reading each step and calling the function associated with it

** each line in a scenario is called a "step"

@weaverryan

Installing Behat

Behat is just a library that can be installed easily in any project via Composer

New to Composer? Free screencast cures it!
http://knpuniversity.com/screencast/composer

In your project directory...

1) Download Composer

```
$> curl -s http://getcomposer.org/installer | php
```

2) Create (or update) composer.json for Behat

http://bit.ly/behat-composer

```
{
    "require": {
        "behat/behat": "2.4.*@stable"
    },
    "minimum-stability": "dev",
    "config": {
        "bin-dir": "bin/"
    }
}
```

http://bit.ly/behat-composer

In your project directory...

1) Download Composer

```
$> curl -s http://getcomposer.org/installer | php
```

2) Create (or update) composer.json for Behat

`http://bit.ly/behat-composer`

3) Download Behat libraries

```
$> php composer.phar install
```

\o/ Woo!

The most important product of the installation is an executable bin/behat file

```
weaverryan@~/Sites/behat$ php bin/behat --help
Usage:
 behat [--init] [-f|--format="..."] [--out="..."] [--lang="..."] [--[no-]ansi] [-
no-]snippets-paths] [--[no-]multiline] [--[no-]expand] [--story-syntax] [-d|--def
cache="..."] [--strict] [--dry-run] [--rerun="..."] [--append-snippets] [features

Arguments:
 features                  Feature(s) to run. Could be:
                           - a dir (features/)
                           - a feature (*.feature)
                           - a scenario at specific line (*.feature:10).
                           - all scenarios at or after a specific line (*.feature:10
                           - all scenarios at a line within a specific range (*.feat

Options:
 --init                    Create features directory structure.

 --format (-f)             How to format features. pretty is default.
                           Default formatters are:
                           - pretty: Prints the feature as is.
                           - progress: Prints one character per step.
                           - html: Generates a nice looking HTML report.
                           - junit: Generates a report similar to Ant+JUnit.
                           - failed: Prints list of failed scenarios.
                           - snippets: Prints only snippets for undefined steps.
                           Can use multiple formats at once (splitted with ",")
 --out                     Write formatter output to a file/directory
```

Behat in a project

To use PHPUnit in a project you need:

1) Actual *Test.php files to be executed
2) (optional) A phpunit.xml configuration file

To use Behat in a project you need:

1) Actual *.feature files to be executed
2) A FeatureContext.php file that holds the PHP callbacks for each step
3) (optional) A behat.yml configuration file

@weaverryan

php·2012 zendcon

$> php bin/behat --init

```
weaverryan@~/Sites/behat$ php bin/behat --init
+d features - place your *.feature files here
+d features/bootstrap - place bootstrap scripts and static files here
+f features/bootstrap/FeatureContext.php - place your feature related code here
```

```php
<?php
// features/bootstrap/FeatureContext.php

use Behat\Behat\Context\ClosuredContextInterface,
    Behat\Behat\Context\TranslatedContextInterface,
    Behat\Behat\Context\BehatContext,
    Behat\Behat\Exception\PendingException;
use Behat\Gherkin\Node\PyStringNode,
    Behat\Gherkin\Node\TableNode;

class FeatureContext extends BehatContext
{
}
```

```php
<?php
// features/bootstrap/FeatureContext.php

use Behat\Behat\Context\ClosuredContextInterface,
    Behat\Behat\Context\TranslatedContextInterface,
    Behat\Behat\Context\BehatContext,
    Behat\Behat\Exception\PendingException;
use Behat\Gherkin\Node\PyStringNode,
    Behat\Gherkin\Node\TableNode;

class FeatureContext extends BehatContext
{
}
```

Nothing Interesting here

yet...

Pretend you’re testing the “ls” program

1) Describe your Feature

```
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current
directory's contents
```

features/ls.feature

2) Your First Scenario

"If you have two files in a directory, and you're running the command - you should see them listed."

2) Write Your First Scenario

** Write in the natural voice of “a UNIX user”

```
Scenario: List 2 files in a directory
  Given I have a file named "foo"
  And I have a file named "bar"
  When I run "ls"
  Then I should see "foo" in the output
  And I should see "bar" in the output
```

features/ls.feature

3) Run Behat

```
$> php bin/behat
```

```
weaverryan@~/Sites/behat$ php bin/behat
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current directory's contents

  Scenario: List 2 files in a directory
    Given I have a file named "foo"
    And I have a file named "bar"
    When I run "ls"
    Then I should see "foo" in the output
    And I should see "bar" in the output

1 scenario (1 undefined)
5 steps (5 undefined)
0m0.018s
```

Behat tries to find a method in FeatureContext for each step

```
You can implement step definitions for undefined steps with these snippets:

    /**
     * @Given /^I have a file named "([^"]*)"$/
     */
    public function iHaveAFileNamed($arg1)
    {
        throw new PendingException();
    }

    /**
     * @When /^I run "([^"]*)"$/
     */
    public function iRun($arg1)
    {
        throw new PendingException();
    }

    /**
     * @Then /^I should see "([^"]*)" in the output$/
     */
    public function iShouldSeeInTheOutput($arg1)
    {
        throw new PendingException();
    }
```

Matching is done via regex

For each step that doesn't have a matching method, Behat prints code to copy into FeatureContext

4) Copy in the new Definitions

```
class FeatureContext extends BehatContext
{
    /** @Given /^I have a file named "([^"]*)"$/ */
    public function iHaveAFileNamed($arg1)
    {
        throw new PendingException();
    }

    /** @When /^I run "([^"]*)"$/ */
    public function iRun($arg1)
    {
        throw new PendingException();
    }
    // ...
}
```

Quoted text maps to a method argument

5) Make the definitions do what they need to

```php
/**
 * @Given /^I have a file named "([^"]*)"$/
 */
public function iHaveAFileNamed($file) {
    touch($file);
}
```

```php
/**
 * @Given /^I have a directory named "([^"]*)"$/
 */
public function iHaveADirectoryNamed($dir) {
    mkdir($dir);
}
```

```php
/**
 * @When /^I run "([^"]*)"$/
 */
public function iRun($command) {
    exec($command, $output);
    $this->output = trim(implode("\n", $output));
}

/**
 * @Then /^I should see "([^"]*)" in the output$/
 */
public function iShouldSeeInTheOutput($string) {
    assertContains(
        $string,
        explode("\n", $this->output)
    );
}
```

```
weaverryan@~/Sites/behat$ php bin/behat
```

```
weaverryan@~/Sites/behat$ php bin/behat
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current directory's contents

  Scenario: List 2 files in a directory
    Given I have a file named "foo"
    And I have a file named "bar"
    When I run "ls"
    Then I should see "foo" in the output
    And I should see "bar" in the output

1 scenario (1 passed)
5 steps (5 passed)
0m0.024s
weaverryan@~/Sites/behat$
```

See the full FeatureContext class:

http://bit.ly/behat-ls-feature

What Behat *does*

Scenario Step

Given I have a file named “foo”

regex

@Given /^I have a file named "([^"]*)"$/

Definition

public function iHaveAFileNamed($file) {

do work

touch($file);

Pass/Fail: Each step is a “test”, which passes *unless* an exception is thrown

Creating files and directories in FeatureContext is nice...

but wouldn't it be really cool to command a browser, fill out forms and check the output?

Act 4

Mink

Mink!

- A standalone library to use PHP to command a “browser”

- One easy API that can be used to command Selenium, Goutte, ZombieJS, etc

@weaverryan

http://mink.behat.org/

php·2012 zendcon

A sample of Mink

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

// change *only* this line to run
// in Selenium, etc
$driver = new GoutteDriver();
$session = new Session($driver);
```

```php
// visit a page
$session->visit('http://behat.org');

 echo 'Status: '.$session->getStatusCode();

 echo 'URL : '.$session->getCurrentUrl();
```

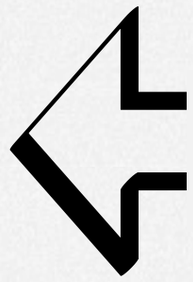
```php
$page = $session->getPage();

// drill down into the page
$ele = $page->find('css', 'li:nth-child(4) a');

echo 'Link text is: '.$ele->getText();
echo 'href is: '.$ele->getAttribute('href');

// click the link
// (you can also fill out forms)
$ele->click();
```

Mink inside FeatureContext

=>

Dangerous Combo for Functional Testing

Integration

Behat  MinkExtension  Mink

- An “Extension” is like a Behat plugin

- The MinkExtension makes using Mink inside Behat a matter of configuration

http://mink.behat.org/

@weaverryan

php·2012 zendcon

Install Mink & MinkExtension

- Update composer.json to include
  * Mink
  * MinkExtension
  * Goutte and Selenium2 Drivers for Mink

  http://bit.ly/behat-mink-composer

- Update the vendor libraries

```
$> php composer.phar update
```

```
{
    "require": {
        "behat/mink": "1.4@stable",
        "behat/mink-goutte-driver": "*",
        "behat/mink-selenium2-driver": "*",
        "behat/behat": "2.4@stable",
        "behat/mink-extension": "*"
    },
    "minimum-stability": "dev",
    "config": {
        "bin-dir": "bin/"
    }
}
```

http://bit.ly/behat-mink-composer

Goal: To easily use Mink inside FeatureContext

Bootstrap MinkExtension

```
# behat.yml
default:
  extensions:
    Behat\MinkExtension\Extension:
      goutte: ~
      selenium2: ~
      # The base URL to app you're testing
      base_url: http://en.wikipedia.org/
```

behat.yml is the Behat configuration file and can contain much more than you see here

http://bit.ly/behat-yml

Extend MinkContext

```
use Behat\MinkExtension\Context\MinkContext;

class FeatureContext extends MinkContext
```

Extending MinkContext gives us 2 things...

@weaverryan

php·2012 zendcon

1) Access to a Mink Session

```
class FeatureContext extends MinkContext
{
    public function doSomething()
    {
        $session = $this->getSession();
        $session->visit('http://behat.org');
    }

    // ...
}
```

Our custom definitions can now command a browser!

2) We inherit a pile of great definitions

Before extending MinkContext:

```
weaverryan@~/Sites/behat$ php bin/behat -dl
Given /^I have a file named "([^"]*)"$/
Given /^I have a directory named "([^"]*)"$/
 When /^I run "([^"]*)"$/
 Then /^I should see "([^"]*)" in the output$/
weaverryan@~/Sites/behat$
```

the -dl option prints all current definitions

After extending MinkContext:

```
weaverryan@~/Sites/behat$ php bin/behat -dl
Given /^I have a file named "([^"]*)"$/
Given /^I have a directory named "([^"]*)"$/
 When /^I run "([^"]*)"$/
 Then /^I should see "([^"]*)" in the output$/
Given /^(?:I )am on homepage$/
 When /^(?:I )go to homepage$/
Given /^(?:I )am on "(?P<page>[^"]+)"$/
 When /^(?:I )go to "(?P<page>[^"]+)"$/
 When /^(?:I )reload the page$/
 When /^(?:I )move backward one page$/
 When /^(?:I )move forward one page$/
 When /^(?:I )press "(?P<button>(?:[^"]|\\")*)"$/
 When /^(?:I )follow "(?P<link>(?:[^"]|\\")*)"$/
 When /^(?:I )fill in "(?P<field>(?:[^"]|\\")*)" with "(?P<value>(?:[^"]|\\")*)"$/
 When /^(?:I )fill in "(?P<value>(?:[^"]|\\")*)" for "(?P<field>(?:[^"]|\\")*)"$/
 When /^(?:I )fill in the following:$/
 When /^(?:I )select "(?P<option>(?:[^"]|\\")*)" from "(?P<select>(?:[^"]|\\")*)"$/
 When /^(?:I )additionally select "(?P<option>(?:[^"]|\\")*)" from "(?P<select>(?:[^"]|\\")*)"$/
 When /^(?:I )check "(?P<option>(?:[^"]|\\")*)"$/
 When /^(?:I )uncheck "(?P<option>(?:[^"]|\\")*)"$/
 When /^(?:I )attach the file "(?P<path>[^"]*)" to "(?P<field>(?:[^"]|\\")*)"$/
 Then /^(?:I )should be on "(?P<page>[^"]+)"$/
 Then /^the (?i)url(?-i) should match (?P<pattern>"([^"]|\\")*")$/
 Then /^the response status code should be (?P<code>\d+)$/
 Then /^the response status code should not be (?P<code>\d+)$/
 Then /^(?:I )should see "(?P<text>(?:[^"]|\\")*)"$/
 Then /^(?:I )should not see "(?P<text>(?:[^"]|\\")*)"$/
 Then /^(?:I )should see text matching (?P<pattern>"(?:[^"]|\\")*")$/
 Then /^(?:I )should not see text matching (?P<pattern>"(?:[^"]|\\")*")$/
 Then /^the response should contain "(?P<text>(?:[^"]|\\")*)"$/
 Then /^the response should not contain "(?P<text>(?:[^"]|\\")*)"$/
 Then /^(?:I )should see "(?P<text>(?:[^"]|\\")*)" in the "(?P<element>[^"]*)" element$/
 Then /^(?:I )should not see "(?P<text>(?:[^"]|\\")*)" in the "(?P<element>[^"]*)" element$/
 Then /^the "(?P<element>[^"]*)" element should contain "(?P<value>(?:[^"]|\\")*)"$/
 Then /^the "(?P<element>[^"]*)" element should not contain "(?P<value>(?:[^"]|\\")*)"$/
 Then /^(?:I )should see an? "(?P<element>[^"]*)" element$/
 Then /^(?:I )should not see an? "(?P<element>[^"]*)" element$/
 Then /^the "(?P<field>(?:[^"]|\\")*)" field should contain "(?P<value>(?:[^"]|\\")*)"$/
 Then /^the "(?P<field>(?:[^"]|\\")*)" field should not contain "(?P<value>(?:[^"]|\\")*)"$/
 Then /^the "(?P<checkbox>(?:[^"]|\\")*)" checkbox should be checked$/
 Then /^the "(?P<checkbox>(?:[^"]|\\")*)" checkbox should not be checked$/
 Then /^(?:I )should see (?P<num>\d+) "(?P<element>[^"]*)" elements?$/
 Then /^print last response$/
 Then /^show last response$/
weaverryan@~/Sites/behat$
```

In other words:

We can write some tests for our app without writing any PHP code

Suppose we're testing Wikipedia.org

```
# features/wikipedia.feature
Feature: Search
  In order to see a word definition
  As a website user
  I need to be able to search for a word

  Scenario: Searching for a page that does exist
    Given I am on "/wiki/Main_Page"
    When I fill in "search" with "Behavior Driven Development"
    And I press "searchButton"
    Then I should see "agile software development"
```

These 4 definitions all come packaged with MinkContext

Celebration!

```
weaverryan@~/Sites/behat$ php bin/behat features/wikipedia.feature
Feature: Search
  In order to see a word definition
  As a website user
  I need to be able to search for a word

  Scenario: Searching for a page that does exist
    Given I am on "/wiki/Main_Page"
    When I fill in "search" with "Behavior Driven Development"
    And I press "searchButton"
    Then I should see "agile software development"

1 scenario (1 passed)
4 steps (4 passed)
0m1.075s
weaverryan@~/Sites/behat$
```

Act 5

Behat with Zend Framework

Getting "under the hood"

- So far, we can do true "black-box" testing - using Mink to test any website (e.g. wikipedia)

- But if we're testing our app, we don't have access to it, we can't:
  a) access/clear/prepare the database
  b) use any code in our application

When testing: you should guarantee the starting condition of your environment

If we had access to Zf2 code,
we could clear the database,
add records, do anything else

Behat with Zend Framework2

1) Install Behat, Mink, MinkExtension

http://bit.ly/behat-mink-composer

2) Bootstrap ZF2 inside FeatureContext

3) Start doing stuff with your code!

2) Bootstrap ZF2 in FeatureContext

```php
class FeatureContext extends MinkContext
{
    /** @var \Zend\Mvc\Application */
    private static $zendApp;

    /** @BeforeSuite */
    static public function initializeZendFramework()
    {
        if (self::$zendApp === null) {
            $path = __DIR__
                .'/../../config/application.config.php';

            self::$zendApp = Zend\Mvc\Application::init(
                require $path
            );
        }
    }
}
```

```php
class FeatureContext extends MinkContext
{
    /** @var \Zend\Mvc\Application */
    private static $zendApp;

    /** @BeforeSuite */
    static public function initializeZendFramework()
    {
        if (self::$zendApp === null) {
            $path = __DIR__
                .'/../../config/application.config.php';

            self::$zendApp = Zend\Mvc\Application::init(
                require $path
            );
        }
    }
}
```

** This is how you bootstrap ZF2

- see public/index.php

```php
class FeatureContext extends MinkContext
{
    /** @var \Zend\Mvc\Application */
    private static $zendApp;

    /** @BeforeSuite */
    static public function initializeZendFramework()
    {
        if (self::$zendApp === null) {
            $path = __DIR__
                .'/../../config/application.config.php';

            self::$zendApp = Zend\Mvc\Application::init(
                require $path
            );
        }
    }
}
```

** This is the hook system
- http://bit.ly/behat-hooks

3) Start doing stuff with your code!

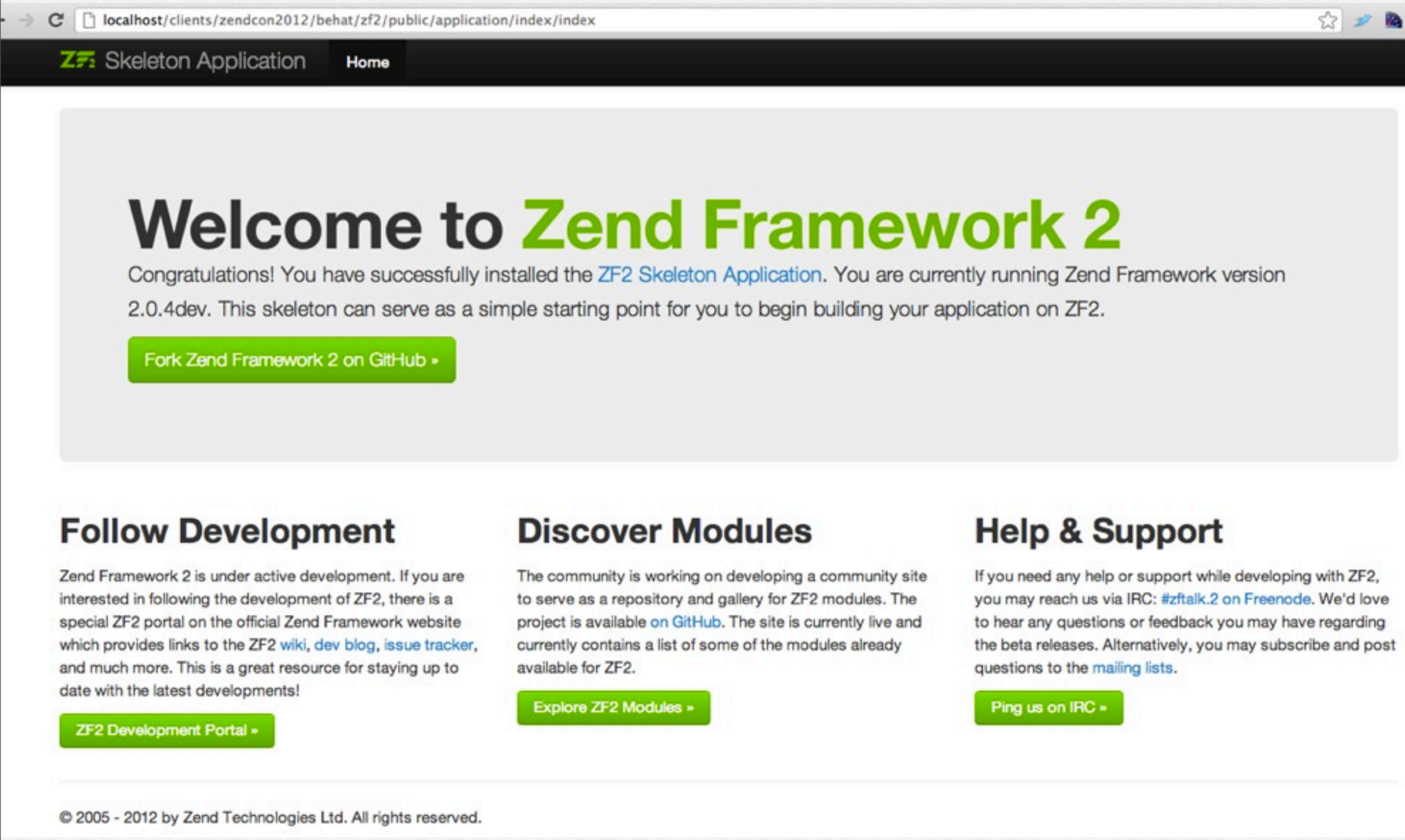
```php
/** @var \Zend\Mvc\Application */
private static $zendApp;

/** @BeforeScenario */
public function clearDatabase()
{
    /** @var $gateway \Zend\Db\TableGateway\TableGateway */
    $gateway = $this->getServiceManager()
        ->get('AlbumTableGateway');

    $gateway->delete(array());
}

/** @return \Zend\ServiceManager\ServiceManager */
private function getServiceManager()
{
    return self::$zendApp->getServiceManager();
}
```

```php
/** @var \Zend\Mvc\Application */
private static $zendApp;

/** @BeforeScenario */
public function clearDatabase()
{
    /** @var $gateway \Zend\Db\TableGateway\TableGateway */
    $gateway = $this->getServiceManager()
        ->get('AlbumTableGateway');

    $gateway->delete(array());
}

/** @return \Zend\ServiceManager\ServiceManager */
private function getServiceManager()
{
    return self::$zendApp->getServiceManager();
}
```

An example hook - run before *every* scenario to clear out parts of the database

Let’s test the demo page of the Zend Skeleton Application

https://github.com/zendframework/ZendSkeletonApplication



http://localhost/clients/zendcon2012/behat/zf2/public/application/index/index

@weaverryan

Adjust your behat.yml base_url

```
default:
  extensions:
    Behat\MinkExtension\Extension:
      goutte: ~
      selenium2: ~
      base_url: http://localhost/clients/zendcon2012/behat/zf2/public/
```

Write a Feature

```
# features/demo_page.feature
Feature: Demo page for ZF2
  In order to show people a functional page immediately
  As a new ZF2 user
  I can access a helpful page that routes through ZF2

  Scenario: View the demo page
    When I go to "/"
    Then I should see "Welcome to Zend Framework 2"
```

Execute it

```
weaverryan@~/Sites/behat/zf2$ ./bin/behat
Feature: Demo page for ZF2
  In order to show people a functional page immediately
  As a new ZF2 user
  I can access a helpful page that routes through ZF2

  Scenario: View the demo page
    When I go to "/"
    Then I should see "Welcome to Zend Framework 2"

1 scenario (1 passed)
2 steps (2 passed)
0m0.094s
weaverryan@~/Sites/behat/zf2$
```

Want the test to run in Selenium?

Add @javascript

```
# features/demo_page.feature
# ...

@javascript
Scenario: View the demo page
  When I go to "/"
  Then I should see "Welcome to Zend Framework 2"
```

Add @javascript

```
# features/demo_page.feature
# ...

  @javascript
  Scenario: View the demo page
    When I go to "/"
    Then I should see "Welcome to Zend Framework 2"
```

Yep, that's all you do!

Download and start Selenium

```
$> wget http://selenium.googlecode.com/files/
selenium-server-standalone-2.25.0.jar

$> java -jar selenium-server-standalone-2.25.0.jar
```

Re-run the tests

```
weaverryan@~/Sites/behat/zf2$ php bin/behat
Feature: Demo page for ZF2
  In order to show people a functional page immediately
  As a new ZF2 user
  I can access a helpful page that routes through ZF2

  @javascript
  Scenario: View the demo page
    When I go to "/"
    Then I should see "Welcome to Zend Framework 2"

1 scenario (1 passed)
2 steps (2 passed)
0m8.63s
weaverryan@~/Sites/behat/zf2$
```

Yes, add only 1 line of code to run a test in Selenium

Epilogue

You’re Turn!

1) Install Behat, Mink and MinkContext in your project

http://bit.ly/behat-mink-composer

http://extensions.behat.org/mink/

2) Bootstrap ZF2 in your FeatureContext

... and then work on a ZendFramework2Extension that would do this automatically (like the SymfonyExtension)

3) Write features for your app!

... and learn more about what you can do with Mink: http://mink.behat.org/

4) and come have some beer with me tonight!

php·2012
zendcon

Thanks...

Ryan Weaver

@weaverryan

SPECIAL thanks to Mr Behat

@everzet

KnpUniversity.com

PHP Tutorial Screencasts

Knp Labs

... and we love you!

php·2012 zendcon

https://joind.in/7391

Ryan Weaver
@weaverryan

Knp Labs