

# Behat

BDD, FUNCTIONAL TESTS &  
SELENIUM (IN DRUPAL!)



**KNP**University



# Hallo!

- > Lead of the Symfony documentation team
- > KnpLabs US - Symfony consulting, training & kumbaya
- > Writer for [KnpUniversity.com](http://KnpUniversity.com):  
PHP & Symfony screencasts  
packed with puns, unrelated  
(but entertaining) illustrations  
and coding challenges!
- > Husband of the much more  
talented @leannapelham



**KNP**University

knpuniversity.com  
[twitter.com/weaverryan](https://twitter.com/weaverryan)



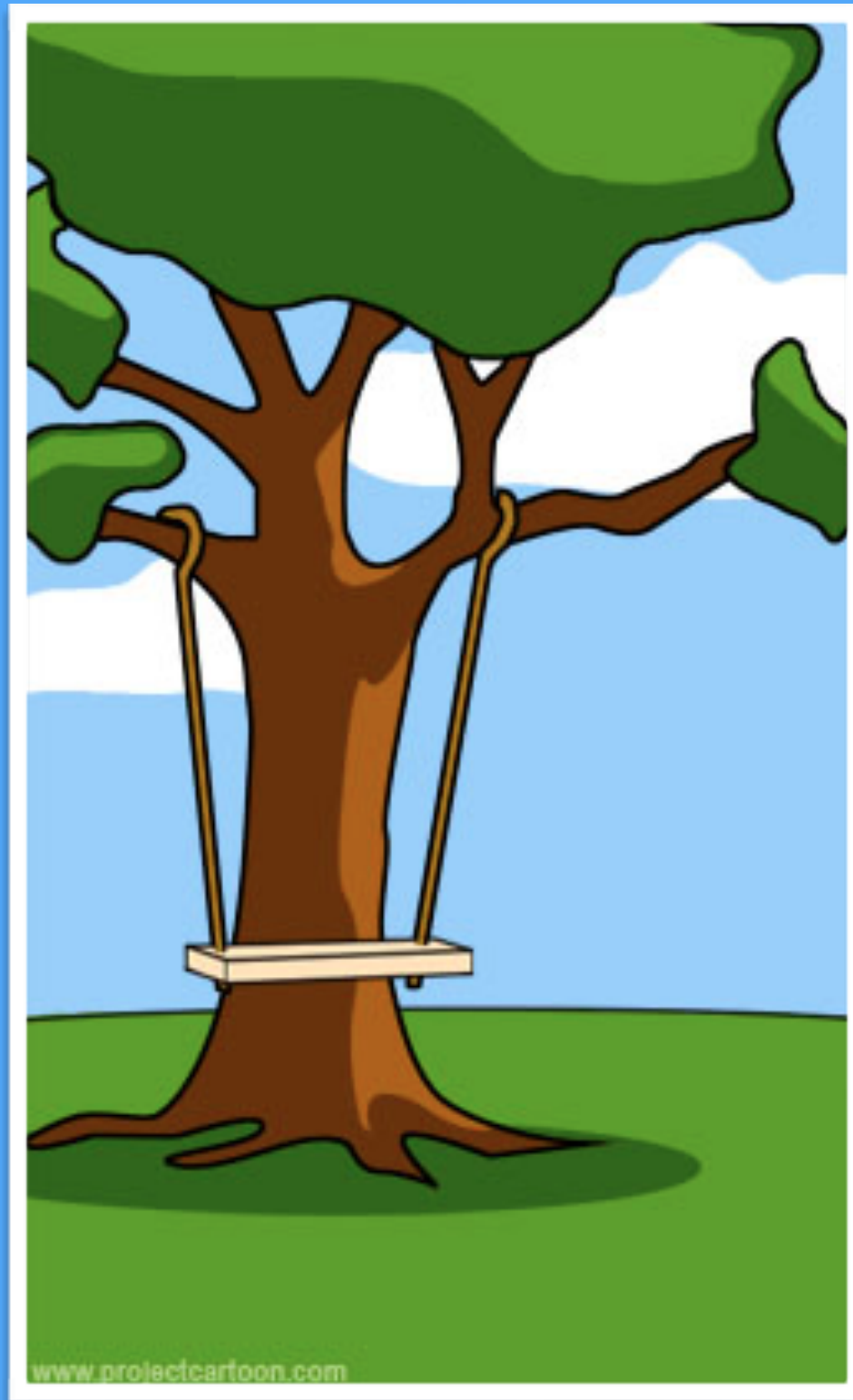
Plan, Work, Miscommunicate,  
Panic, Put out Fires, Repeat!



aka Project Work!

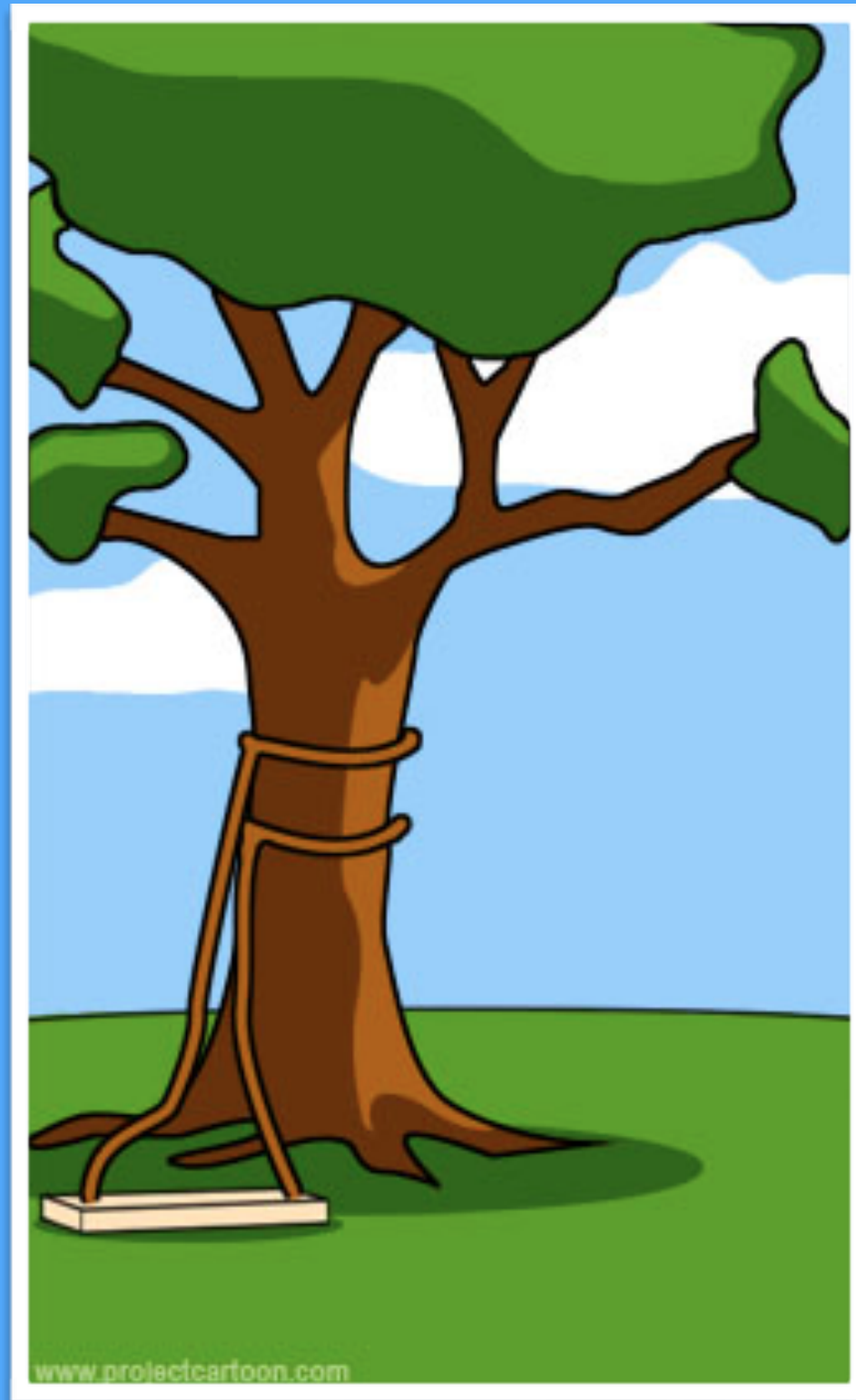


How the customer explained it

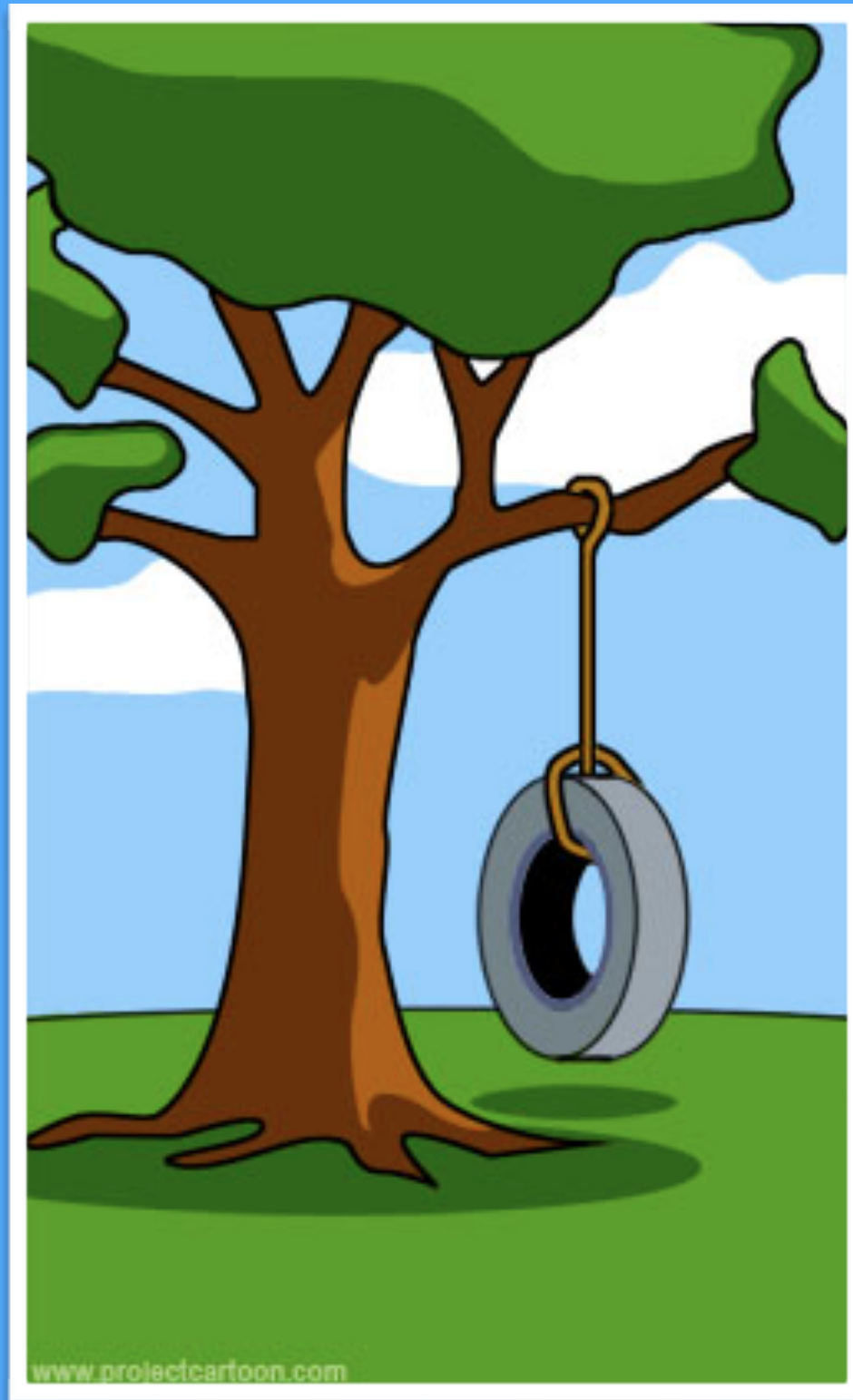


How the project leader understood it

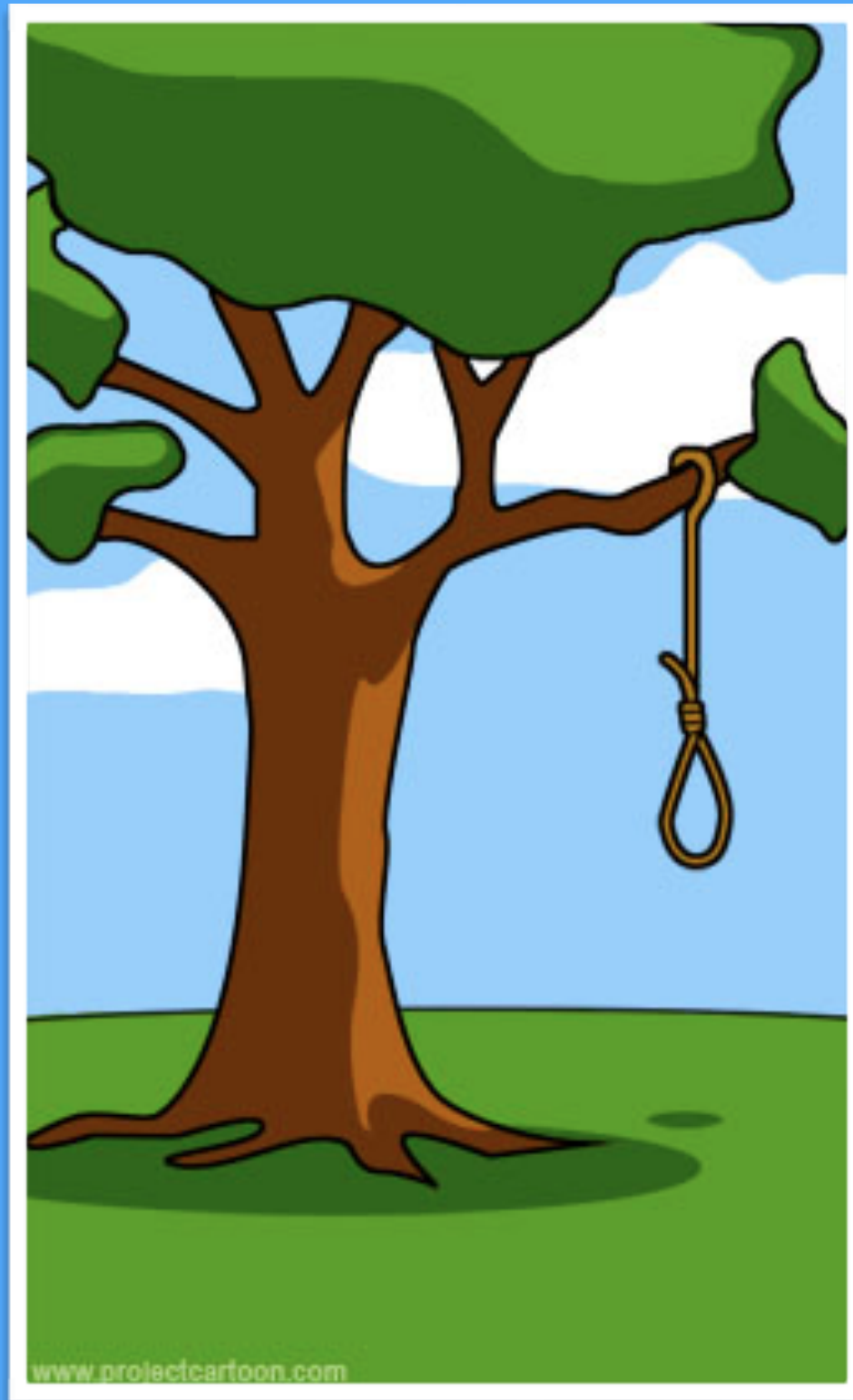




How the programmer wrote it



What the customer really needed



What the beta testers received





# Computer Science?

<https://www.flickr.com/photos/diueine/3604050776>

One

---

Different roles,  
different languages,  
miscommunication

Two

---

Your code and business  
values may not align

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

***Why? Because it's cool!***

Three

Over-planning,  
under-planning,  
planning...?

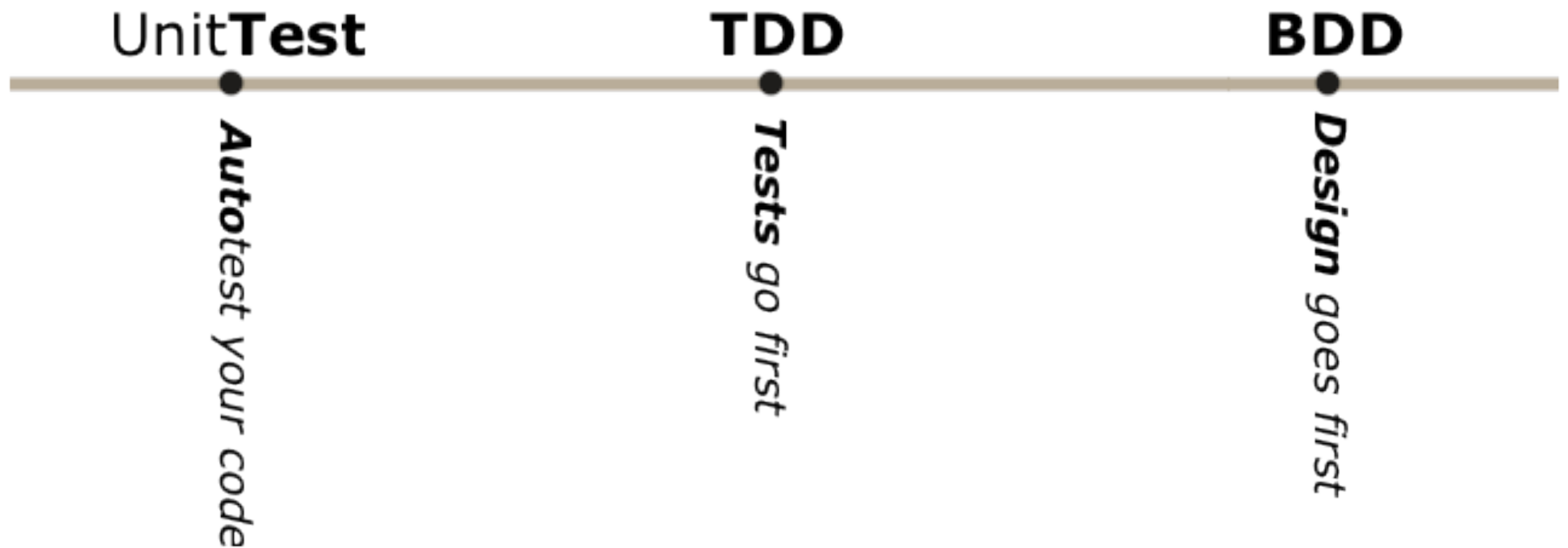


A golden disco ball is the central focus, hanging from a chain. It is covered in small, reflective facets that catch the light. The background is dark with warm, out-of-focus light spots, creating a bokeh effect. A semi-transparent horizontal band is overlaid across the middle of the image, containing the title text.

# Getting down with BDD



# Evolution of Test-Driven Development

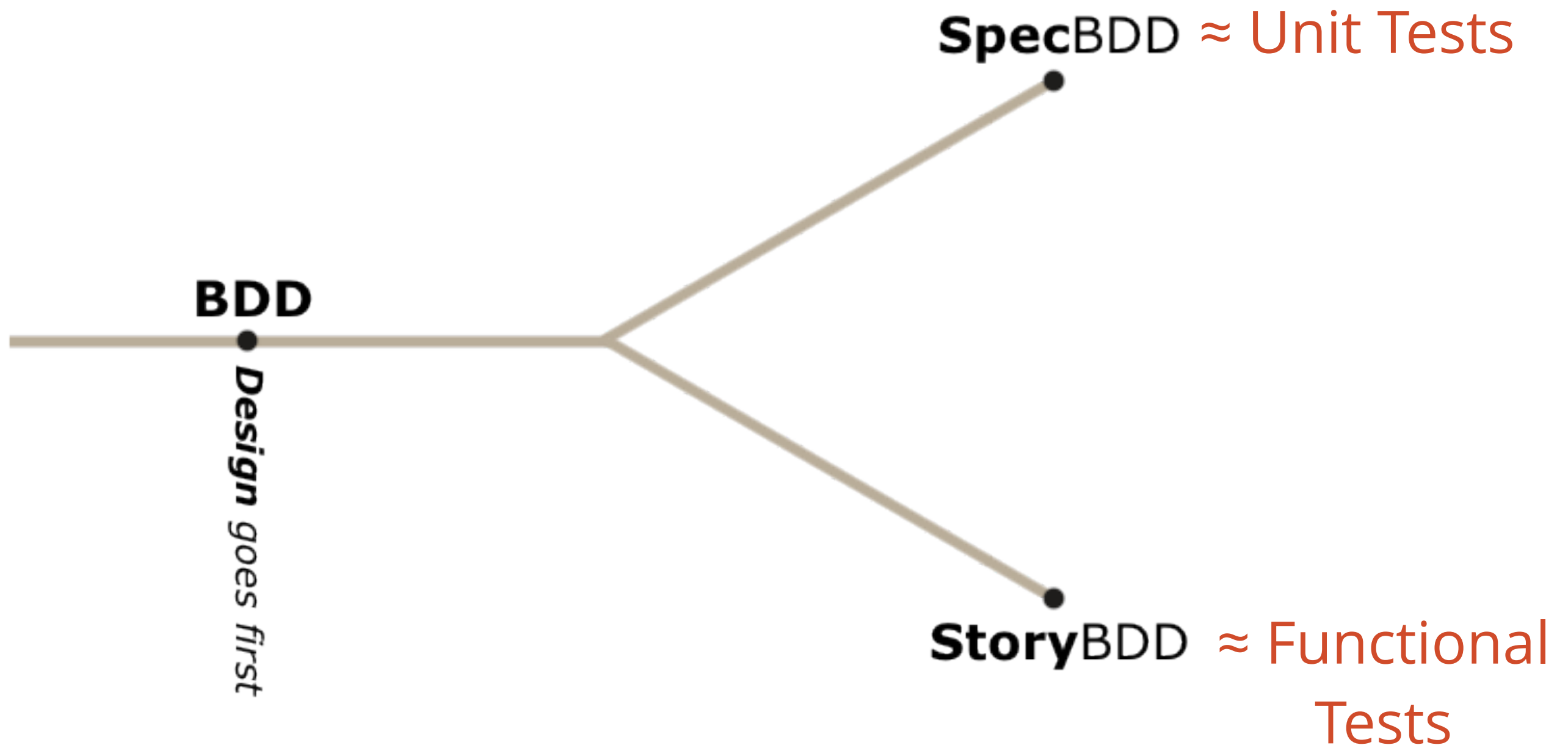


“Behaviour” is a more useful word, than “test”

- Dan North \*

\* the santa of behavior-driven development

# Evolution of Test-Driven Development



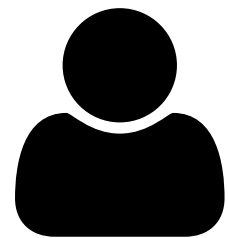
# Specification BDD



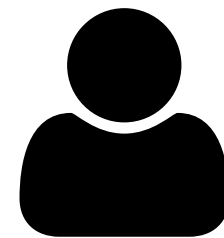
<http://www.phpspec.net>

# Scenario-oriented BDD (Story BDD)

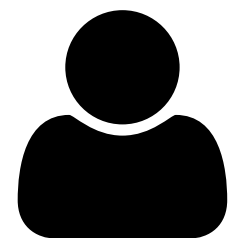
# Let's create a single vocabulary and process



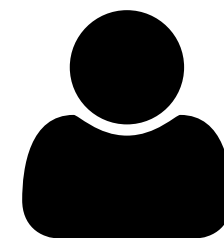
Project Managers



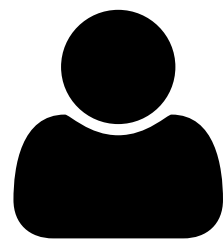
Business Analyst



Developers



Product Owners



QA



# Calming the Chaos

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





# Gherkin



# Gherkin



a structured 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"

# Calming the Chaos

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

# Calming the Chaos

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: Add new article

Given I am on the "/admin/news" page

When I click "New Article"

And I fill in "Title" with "Learned BDD"

And I press "Save"

Then I should see "A new article was 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"

And I press "Save"

Then I should see "A new article was 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"

And I press "Save"

Then I should see "A new article was 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"

And I press "Save"

Then I should see "A new article was 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"

And I press "Save"

Then I should see "A new article was added"

## And/But

Can be added to create multiple

Given/When/Then lines

# Example #2

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



# Gherkin

gives us a consistent  
language for describing  
**features** and their **scenarios**

... now let's turn them into  
tests!

*Those tests ain't gonna  
write themselves,*

***Behat****ch*

# 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”

# Installing Behat

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

New to Composer? Free screencast cures it!

[KnpUniversity.com/screencast/composer](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

```
$> php composer.phar require --dev behat/behat
```



```
{  
  "require-dev": {  
    "behat/behat": "^3.1"  
  }  
}
```

[bit.ly/behat3-composer](https://bit.ly/behat3-composer)

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

```
~/Sites/behats$ php vendor/bin/behats --help
```

## Usage:

```
behats [-s|--suite="..."] [-f|--format="..."] [-o|--out="..."] [--format] [--lang="..."] [--name="..."] [--tags="..."] [--role="..."] [--steps="..."] [--append-snippets] [--no-snippets] [--strict] [--rerun] [un] [paths]
```

## Arguments:

<b>paths</b>	Optional path(s) to execute. Could be: <ul style="list-style-type: none"><li>- a dir (<b>features/</b>)</li><li>- a feature (<b>*.feature</b>)</li><li>- a scenario at specific line (<b>*.feature:10</b>).</li><li>- all scenarios at or after a specific line (<b>*.feature:10-</b>)</li><li>- all scenarios at a line within a specific range (<b>*.feature:10-20</b>)</li><li>- a scenarios list file (<b>*.scenarios</b>).</li></ul>
--------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Options:

<b>--suite</b> (-s)	Only execute a specific suite.
<b>--format</b> (-f)	How to format tests output. <b>pretty</b> is default. Available formats are: <ul style="list-style-type: none"><li>- <b>progress</b>: Prints one character per step.</li><li>- <b>prettv</b>: Prints the feature as is.</li></ul>

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

```
$> php vendor/bin/behat --init
```

```
~/Sites/behat$ php vendor/bin/behat --init
```

```
+d features - place your *.feature files here
```

```
+d features/bootstrap - place your context classes here
```

```
+f features/bootstrap/FeatureContext.php - place your definitions,  
re
```

```
~/Sites/behat$ █
```

```
<?php
```

```
// features/bootstrap/FeatureContext.php
```

```
use Behat\Behat\Context\SnippetAcceptingContext;
```

```
use Behat\Gherkin\Node\PyStringNode;
```

```
use Behat\Gherkin\Node\TableNode;
```

```
/**
```

```
 * Behat context class.
```

```
 */
```

```
class FeatureContext implements SnippetAcceptingContext  
{  
}
```



```
<?php
```

```
// features/bootstrap/FeatureContext.php
```

```
use Behat\Behat\Context\SnippetAcceptingContext;
```

```
use Behat\Gherkin\Node\PyStringNode;
```

```
use Behat\Gherkin\Node\TableNode;
```

```
/**
```

```
 * Behat context class.
```

```
 */
```

```
class FeatureContext implements SnippetAcceptingContext  
{  
}
```

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.

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 vendor/bin/behat
```



```
~/Sites/behat$ php vendor/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)
```

```
0m0.04s (9.97Mb)
```

Behat tries  
to find a  
method in  
FeatureContext  
for each step

-- FeatureContext has missing steps. Define them with these snippets:

```
/**
 * @Given I have a file named :arg1
 */
public function iHaveAFileNamed($arg1)
{
    throw new PendingException();
}

/**
 * @When I run :arg1
 */
public function iRun($arg1)
{
    throw new PendingException();
}

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

Matching is done  
with  
simple wildcards

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 :file */
```

```
    public function iHaveAFileNamed($file)
```

```
    {
```

```
        throw new PendingException();
```

```
    }
```

```
    /** @When I run :command */
```

```
    public function iRun($command)
```

```
    {
```

```
        throw new PendingException();
```

```
    }
```

```
    // ...
```

```
}
```

Quoted text  
maps to a  
method  
argument

5) Make the definitions do what they need to

```
/**
 * @Given I have a file named :file
 */
public function iHaveAFileNamed($file) {
    touch($file);
}
```

```
/**
 * @Given I have a directory named :dir
 */
public function iHaveADirectoryNamed($dir) {
    mkdir($dir);
}
```

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

/**
 * @Then I should see :string in the output
 */
public function iShouldSeeInTheOutput($string) {
    if (strpos($this->output, $string) === false) {
        throw new \Exception('Did not see' . $string);
    }
}
```

```
~/Sites/behat$ php vendor/bin/behat █
```



```
~/Sites/behat$ php vendor/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)
```

```
0m0.04s (10.08Mb)
```

```
~/Sites/behat$ █
```

See the full FeatureContext  
class:

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

# What Behat \*does\*

Scenario Step



pattern



Definition



do work



Given I have a file named "foo"



@Given I have a file named :file



public function iHaveAFileNamed(\$file) {



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?



A mink with dark brown fur is perched on a light-colored wooden log. The mink is facing left, looking alertly. The background is a lush green environment with various plants, including tall grasses and small yellow flowers in the foreground. A semi-transparent grey banner is overlaid across the middle of the image.

# Mink

<https://www.flickr.com/photos/15016964@N02/5696367600>

# Mink!

- ▶ A standalone library to use PHP to command a “browser”
- ▶ One easy API that can be used to command Selenium, Goutte, PhantomJS, etc

A sample of Mink



```
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);
```

```
// visit a page
```

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

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

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

```
$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



- ▶ An “Extension” is like a Behat plugin
- ▶ The MinkExtension makes using Mink inside Behat a matter of configuration

# Install Mink & MinkExtension

- ▶ Update composer.json to include:
  - > Mink
  - > MinkExtension
  - > Goutte and Selenium2 Drivers for Mink

```
composer require --dev \
    behat/mink-extension \
    behat/mink-goutte-driver \
    behat/mink-selenium2-driver
```

```
{  
  "require-dev": {  
    "behat/behat": "^3.1",  
    "behat/mink-extension": "^2.2",  
    "behat/mink-goutte-driver": "^1.2",  
    "behat/mink-selenium2-driver": "^1.3"  
  }  
}
```

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



Goal:

To easily use Mink inside  
FeatureContext

# Bootstrap MinkExtension

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

# Extend MinkContext

```
use Behat\MinkExtension\Context\RawMinkContext;  
  
/**  
 * Behat context class.  
 */  
class FeatureContext extends RawMinkContext
```

# Access to a Mink Session

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

    // ...
}
```

Our custom definitions can now  
command a browser!

# More! Add MinkContext

```
# behat.yml
default:
  extensions:
    # ...
  suites:
    default:
      contexts:
        - FeatureContext
        - Behat\MinkExtension\Context
\MinkContext
```

Behat now parses definitions from *\*our\**  
class *\*and\** this MinkContext class

# We inherit a pile of great definitions

Before adding MinkContext:

```
~/Sites/behat$ php vendor/bin/behat -dl
default      Given I have a file named :file
default      When I run :command
default      Then I should see :string in the output

~/Sites/behat$
```

the **-dl** option prints all current definitions

# After adding MinkContext:

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

```
~/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!

```
~/Sites/behat$ php vendor/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)
```

```
0m0.59s (15.18Mb)
```

```
~/Sites/behat$ █
```

Behat in your application



Behat in your App

# Getting “under the hood”

- ▶ Black-box testing: the site lives out on the web
- ▶ Because of this, 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

How can we add **nodes**, add  
**users**, and configure  
**permissions** from inside  
Behat?



# Behat & Drupal

- ▶ Install Behat & Mink
- ▶ ??? Gain access to Drupal functionality from inside FeatureContext
- ▶ ~~PROFIT!~~ Create nodes, users, etc so that you're testing against a predictable dataset

Fortunately...

... there's a library made by  
the Drupal community ...

... which I did not help with ...

# DrupalExtension!



jhedstrom

A plugin (extension) for Behat and Drupal

<http://bit.ly/drupal-extension>

# DrupalExtension

- 1) Even more built-in sentences/definitions
- 2) Build nodes, add users, manage permissions inside Behat
- 3) Operating within Regions
- 4) Hooks to load more sentences/definitions from contrib modules

# features/node\_manage.feature

## Background:

Given I am logged in as a user with the "administrator" role

## Scenario: Edit Node

Given I am viewing a "page" with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

Creates a user and  
adds a role to it

# features/node\_man

Background:

Given I am logged in as a user with the "administrator" role

Scenario: Edit Node

Given I am viewing a "page" with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

# features/node\_manage.feature

## Background:

Given I am logged in

Creates a "page" node  
in the database

as administrator role

## Scenario: Edit Node

Given I am viewing a "page" with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm

And I press "Save"

Then I should see "Ipsumm" in the "Body" region





# And it's alive!

```
~/Sites/drupal$ ./bin/behat
```

```
Feature: Node Management
```

```
Background:
```

```
    Given I am logged in as a user with the "administrator" role
```

```
Scenario: Edit Node
```

```
    Given I am viewing a "page" node with the title "Cool beans!"
```

```
    When I click "Edit" in the "Content" region
```

```
    And I fill in the following:
```

```
        | Body | Ipsummm |
```

```
    And I press "Save"
```

```
    Then I should see "Ipsummm" in the "Content" region
```

```
1 scenario (1 passed)
```

```
6 steps (6 passed)
```

```
0m1.57s
```

```
~/Sites/drupal$ █
```

# The 3 Modes of the DrupalExtension

- 1) **blackbox**: test an external server, no access to the database
- 2) **drupal**: Bootstraps Drupal's code and calls functions
- 3) **drush**: Interacts with Drupal via drush

What if my page/test rely  
on JavaScript

Behat/Mink does *not*  
support testing pages that  
use JavaScript

jk!

# Add @javascript

# ...

@javascript

Scenario: Edit Node

Given I am viewing a "page" with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region

# Add @javascript

# ...

Yep, that's all you do!

@javascript

Scenario: Edit Node

Given I am viewing a "page" with the title "Cool beans!"

When I click "Edit" in the "Body" region

And I fill in the following:

Body	Ipsumm
------	--------

And I press "Save"

Then I should see "Ipsumm" in the "Body" region



# Download and start Selenium

```
$> wget http://selenium-  
release.storage.googleapis.com/2.53/selenium-  
server-standalone-2.53.0.jar
```

```
$> java -jar selenium-server-standalone-2.53.0.jar
```

# Re-run the tests

```
~/Sites/drupal$ ./bin/behat
```

```
Feature: Node Management
```

```
Background:
```

```
    Given I am logged in as a user with the "administrator" role
```

```
@javascript
```

```
Scenario: Edit Node
```

```
    Given I am viewing a "page" node with the title "Cool beans!"
```

```
    When I click "Edit" in the "Content" region
```

```
1 scenario (1 passed)
```

```
3 steps (3 passed)
```

```
0m5.036s
```

```
—
```

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

# Bonus!

Mink directly via PHPUnit?

See BrowserTestBase

\*\*and (the new) JavascriptTestBase

Ep·i·logue

It's Simple!



# 1) Install Behat

## Chapters

You're **43%** through this course



✓ From Install to JS Testing	7:27
✓ BDD Features 📄 📄 2 challenges	5:56
✓ Scenarios 📄 📄 2 challenges	4:24
✓ Behat 📄 1 challenge	6:08
✓ Behat Hooks Background 📄 1 challenge	6:49
✓ Mink	3:27
✓ Finding Elements by CSS and Name	6:48
Behat Loves Mink (Free Definitions from MinkExtension)	2:56
Scenario Outline	2:10

## What you'll be learning

Looking for Behat v2.5 of the tutorial? See

<https://knpuniversity.com/screencast/behav-v25>.

Behat is my absolute favorite library to use. First, it let's me think about the *behavior* of my features first, *before* I start developing. In this tutorial, we'll do that, and it'll change the way you develop. Second, Behat turns that written behavior into functional tests against your application. Does your feature behave correctly? Just run the automated robots to find out!

Along the way, we'll master Mink - the partner library to Behat - and solve all sorts of common problems, like:

- Running scenarios in a real browser
- Properly waiting on JavaScript events
- Leveraging Behat hooks
- Multiple contexts
- Loading Database fixtures and clearing data
- Bootstrapping your application (specifically Symfony)
- Using the authenticated user
- Tagging scenarios
- How to put BDD into practice
- Keeping the velociraptors in their pen.

Let's do this!



<http://knpuniversity.com/screencast/behav>

## 2) Write features for your app!

---

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

# 3) high-five your teammates





# THANK YOU!

Ryan Weaver  
@weaverryan



**KNP**University

