Making Drupal Behave

Automated Testing with Behat

http://bit.ly/1bNyvgo

Nice to meet you

Howard Tyson

- @tizzo on drupal.org, github, twitter, IRC, and everywhere else
- Drupaler for 7 years
- VP of Engineering at Zivtech

Frank Carey

- @frankcarey on drupal.org, github, twitter, IRC, and everywhere else
- Drupaler for 7 years
- VP of Product at Zivtech
- AI, Robotics, and Brain Science

The Problem

Regressions

- You change something somewhere
- but that breaks something somewhere else
- and you fix it
- breaking that first thing again…

The Solution

Testing, maybe you've heard of it?

Fail happens

Dealing with it is on you

Test Driven Development (TDD)

- Red -> Green -> Refactor
- Generally highly implementation specific
- Tests that the code does what the code does, not what the business needs it to do
- Writing tests is laborious

Behavior Driven Development (BDD)

- Shared language
- Shared understanding
- Tests composed of human readable pieces

Gherkin

- DSL for describing tests
- human readable
- but not just natural language

Start by explaining why this exists

```
1 Feature: Search for a module on drupal.org
2   In order to find a module that I want to use
3   As an anonymous user
4   I want to search for content on drupal.org
5   without trying very hard
```

Add scenarios that explain what you do

```
 7 @javascript
 8 Scenario: Anonymously search drupal.org
 9     Given I go to "http://drupal.org"
10       And I fill in "Search Drupal.org" with "Views"
11      When I press "Search"
12       And I follow "Modules"
13      Then I should see "Posted by merlinofchaos"
14       And I follow "Views"
15       And I should see "You need Views if"
```

Rubber, meet road

The Tools

- Behat
- Mink
- Mink Extension
- Drupal Extension

Behat

- Equivalent to Ruby's Cucumber
- Runs the feature files described earlier
- Maps each english line in the Domain Specific Language to pre-defined functions
- *NOT MAGIC* - uses regular expressions

Mink

- Equivalent to Ruby's Capybara
- Provides a common API to to multiple browsers via drivers
  - goutte
  - selenium
  - zombie.js
    - this one will eat your brains
    - no …really, you shouldn't use it…

Behat Mink Extension

- Provides the glue that ties Behat and Mink together
- Mostly a large set of reusable steps with a few utilities mixed in

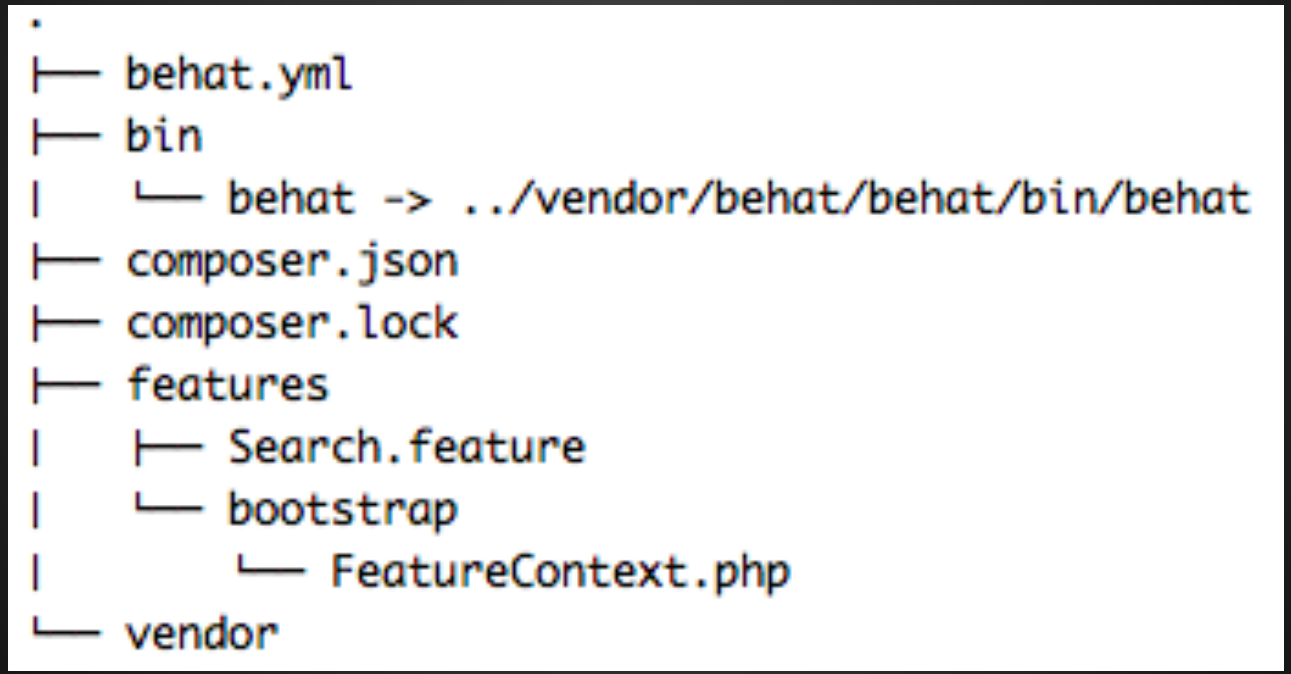
Drupal Extension

- Adds more Drupal specific steps
- Provides drivers with multiple ways to interface with a Drupal site
  - Native bootstrap
  - Drush

Let’s see it

Writing your own steps

The anatomy of a behat project



Defining your own functions

```
/**
 * @When /^visit a "([^"]*)" user edit page$/
 */
public function visitAUserEditPage($role) {
  $this->visit('/user/' . $this->getTestData('user', $role, 'uid') . '/edit');
}
```

Here, getTestData() is a method that fetches data about test content from an HTTP callback provided by a custom module.

Reuse existing steps

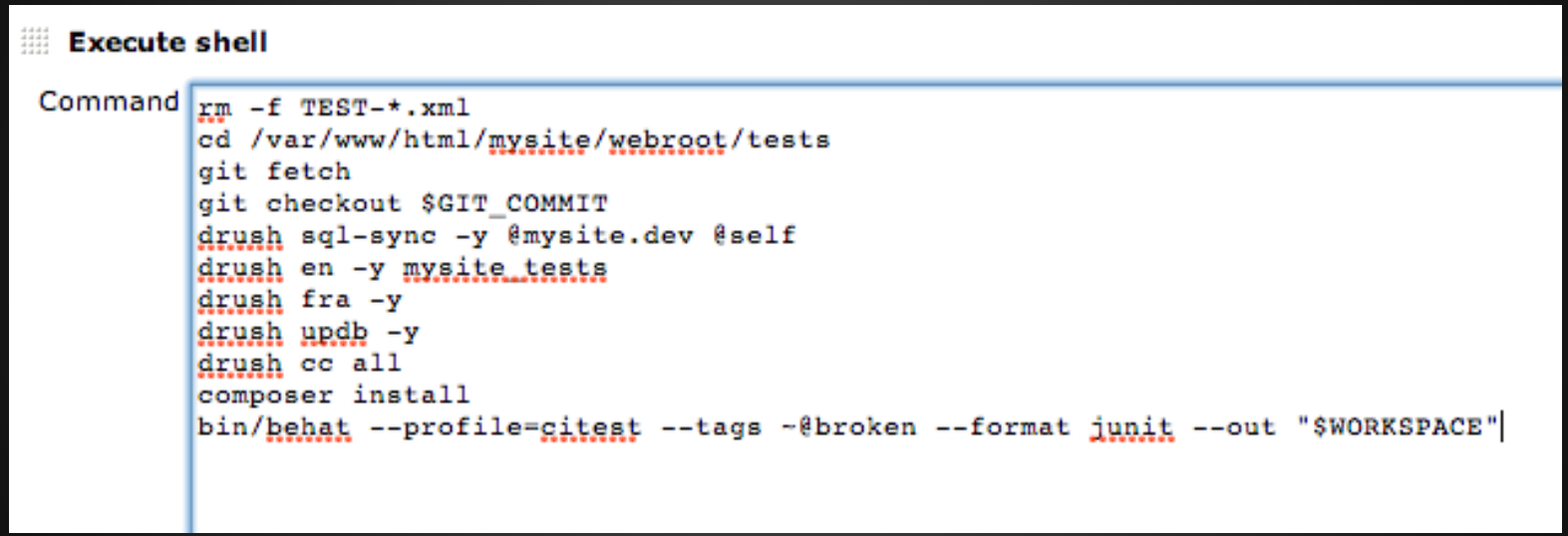
```php
/**
 * @Given /^I am logged in$/
 */
public function iAmLoggedIn() {
  $steps = array();
  $steps[] = new Then('I am on the homepage');
  $steps[] = new Then('I fill in "foo" for "user"');
  $steps[] = new Then('I fill in "bar" for "password"');
  $steps[] = new Then('I press "Sign in"');
  return $steps;
}
```

Tips

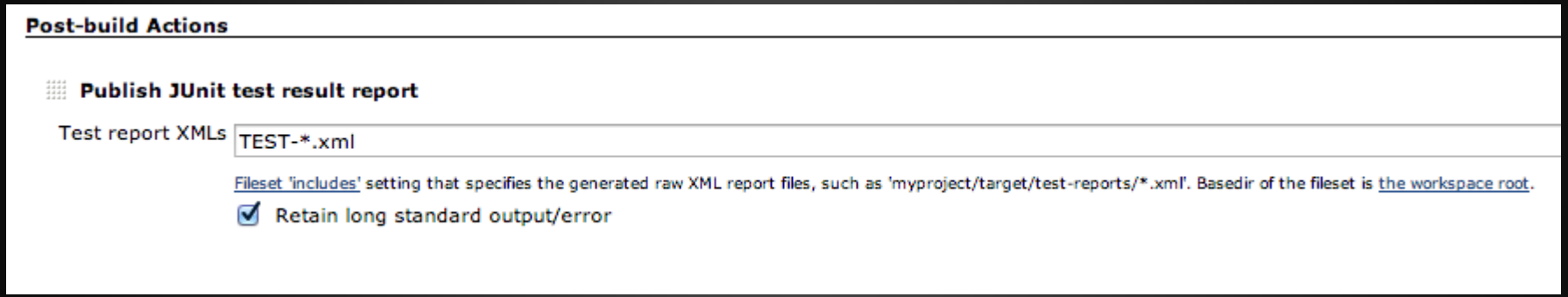
Tips

- Run tests regularly
- Run tests with every commit (or push)
- Speed is an important feature
- It’s insanely helpful to make a custom Drupal module that can facilitate test setup and teardown

Running tests with Jenkins



Reporting on test results



One-off steps

Given I’m logged in
Then I should see that node 11 is unpublished

The “11” should be a variable at least.
What about “unpublished” ?

Overly specific Scenarios

Create Meta-Steps that use sub-steps

```
/**
 * @Given /I entered "([^"]*)" and expect "([^"]*)"/
 */
public function complexStep($number, $result)
{
    return array(
        new Step\Given("I have entered \"$number\""),
        new Step\When("I press +"),
        new Step\Then("I should see \"$result\" on the screen")
    );
}
```

Magicky Handwavy Steps

Example:

Given I invent a time machine

Then I should get rich

These steps don’t give enough detail into what’s actually happening and what’s being tested.

Testing Variations

Useful to test variations or extremes of scenarios with Outline Scenarios

```
Scenario Outline: Eating
  Given there are <start> cucumbers
  When I eat <eat> cucumbers
  Then I should have <left> cucumbers

  Examples:
    | start | eat | left |
    |  12   |  5  |  7   |
    |  20   |  5  |  15  |
```

pre-req steps

Try to avoid having scenarios and features that depend on others being run first.

Best practice IMO is to use “Backgrounds” http://docs.behat.org/guides/1.gherkin.html#backgrounds

Gotchas!

It’s not all roses..

Gotchas!

- Classes Galore
- Javascript / Ajax
- @beforeFeature
- I should NEVER see..
- Inconsistently returned objects.

Gotchas! - So. Many. Classes.

Behat and it’s dependencies have a crap ton of very small classes which can make it a bit of a beast to track down what methods are available beyond the FeatureContext class.

Gotchas - Javascript / Ajax

Adding the time element…

For instance the “I press ” events do not block, so you need to wait, but how long?

Solutions:

Set a specific wait or polling..

Setting a specific Wait

Good:

- It’s easy to do.

Bad:

- Maybe the load takes longer sometimes
- What are you waiting for?
- Waits add up!

Wait Example

```php
<?php
  /**
   * @Given /^I wait (\d+) (second|seconds)$/
   */
  public function iWaitSeconds($seconds)
  {
    $this->getSession()->wait(1000*$seconds);
  }
```

http://pastebin.com/ptZYmCmr

Polling the “browser”

Good:

- You’re only waiting as long as necessary
- You ~~can~~ MUST set a timeout.

Bad:

- It’s not built in to any existing step functions
- Fails will wait the FULL timeout

Polling Example 1 - Spin()

```php
function spin($lambda, $wait = 60){
  for ($i = 0; $i < $wait; $i++) {
    try {
      if ($return = $lambda($this)) {
        return $return;
      }
    } catch (Exception $e) {
      // Uncomment to debug exceptions.
      //var_dump($e->getMessage());
    }
    sleep(1);
  }

  $backtrace = debug_backtrace();

  throw new Exception(
    "Timeout thrown by " . $backtrace[1]['class'] . "::" .
    $backtrace[1]['file'] . ", line " . $backtrace[1]['line
  );
}
```

http://pastebin.com/ZjbuT9KS

Polling Example 2 - Closures (PHP >= 5.3.0)

```
  function findTimeout($search, $selector = 'css', $timeout = 5) {
    return $this->spin( function($context) use ( $selector, $search) {
      $page = $context->getSession()->getPage();
      if ($selector == "css") {
        $el = $page->find('css', $search);
      }
      else {
        $el = $page->find('named', array($selector, $context->getSession(
>xpathLiteral($search)));
      }
      return($el);
    }, $timeout);
  }
```

http://pastebin.com/ZjbuT9KS

Polling Example 3 - Smarter Search

```
/**
 * @Then /^I click on "([^"]*)"$/
 * @Then /^I click on "([^"]*)" "([^"]*)"$/
 */
public function iClickOn($search, $selector = "css") {
  $el = $this->findTimeout($search, $selector);
  assertNotNull($el, "Couldn't find $search $selector on the page.");
  $el->click();
  // Wait for any ajax calls to finish.
  $this->getSession()->wait(10000, '(0 === jQuery.active)');;
}
```

http://pastebin.com/ZjbuT9KS

Gotchas! - I should NEVER see ..

“I should NEVER see.. PHP warnings/errors”

There isn’t really a good way to do this type of thing. In theory it should look for this on every request, but you’d need to override classes.

Gotchas! - @beforeWTF!

@beforeFeature
@beforeOutline
@beforeScenario
@beforeStep
- Param types are different for each
- @beforeFeature is static!
- More Magic that can be overlooked

Inconsistent Returns

find() returns an object if it finds it, but null if it doesn’t and it doesn’t throw an error when it doesn’t find the thing it was looking for.

If you aren’t careful, this will throw a PHP undefined method error and crash your whole test instead of just failing.

Testing the Tests

Testing Tests

Given Tests are in Code

And Code is written by Humans

And Humans make mistakes

When Tests have mistakes

Then We need to create tests for the tests

Writing behat tests for behat tests

Writing behat tests for behat tests

Kidding! - But a few things can go wrong..

- Your tests throw unexpected exceptions
- False Positive - Tests fail when they should pass
- False Negative - Test pass when they should fail
- Intermittent Fails
- WTF Fails

Simple Debugging

Getting more details

Pausing the action with breakpoints

Inspecting the page (browser)

Inspecting PHP Variables

How do we do this best in behat?

Getting more details

behat --expand -v : More details..

```
⚡ behat --expand
Feature: Some feature
  In order to ...
  As a ...
  I need ...

  Scenario Outline:                # features/example.feature:6
    Given some value "<input>"     # FeatureContext::someValue()
    When I do something            # FeatureContext::iDoSomething()
    Then I should see "<output>"   # FeatureContext::iShouldSee()

      Examples: | 23 | 55 |
        Given some value "23"      # FeatureContext::someValue()
        When I do something        # FeatureContext::iDoSomething()
          Some exception
        Then I should see "55"     # FeatureContext::iShouldSee()

      Examples: | 34 | 12 |
        Given some value "34"      # FeatureContext::someValue()
        When I do something        # FeatureContext::iDoSomething()
          Some exception
        Then I should see "12"     # FeatureContext::iShouldSee()

2 scenarios (2 failed)
6 steps (2 passed, 2 skipped, 2 failed)
0m0.048s
```

Details - Find the code

behat -di

Lists:

- ALL of the step definitions
- Step descriptions (if they exist)
- Actual method names (beware colors)

Details - Understand the code

Once you have the actual method name you should be able to find the code in your context or one of it's parent classes.

Take the time to understand what it's really doing to perform an action or search the page.

i.e. - Is it searching for first occurrence, html id, inner text, label?

Pausing the action

Create a custom “breakpoint step” you can add between steps to debug. It simply waits until you hit enter on the command line.

Gives you time to inspect the page (selenium), the site, or the database before moving on to the next step.

Breakpoint Example

```
  /**
   * @Given /^breakpoint$/
   */
  public function breakpoint() {
        fwrite(STDOUT, "\033[s \033[93m[Breakpoint] Press 
continue...\033[0m");
        while (fgets(STDIN, 1024) == '') {}
        fwrite(STDOUT, "\033[u");
        return;
  }
```

http://pastebin.com/K6Vx95R4

Inspecting Variables

custom dpm() (or dsm) for behat:

```php
<?php

/**
 * Print out a variable to the behat command line.
 */
function dpm($variable) {
  $this->printDebug(print_r($variable, TRUE));
  // You might also use the breakpoint method here.
}
```

http://pastebin.com/zP97EMJX

Inspecting Variables

Why not use debugger to run behat?

On my todo list, but I haven't tried it. I've seen places where they say xdebug needs to be off, but this suggests it might work.
http://bit.ly/1dfD0jz

Inspecting the Page

Two useful steps:

Then print last response

- Prints the html to the command line

Then show last response

- Opens html (tmp file) in browser
- Pauses steps until browser’s closed

Questions?

@tizzo

@frankcarey