PNWDS 2018

Jonathan Hedstrom

Automated Testing in Drupal 8

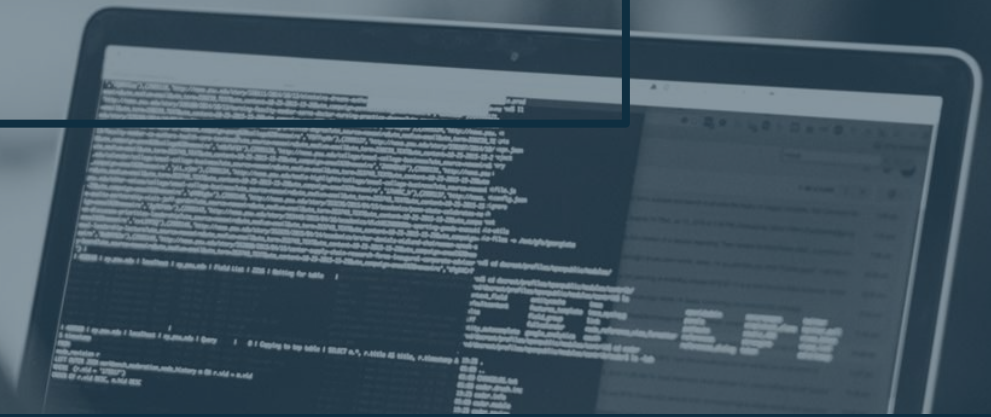
Introduction

- jhedstrom nearly everywhere
- jhedstro on Twitter

Testing in Drupal 8

1. Why test?
2. What to test?
3. Which type of tests?
4. Practical examples
5. Go forth and test!
6. Code examples

Why Test?



Testing leads to failure, and failure leads to understanding

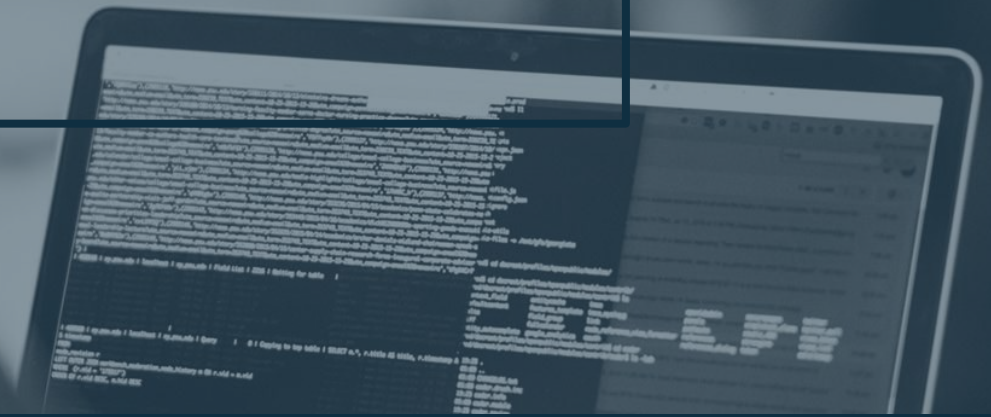
~ Burt Rutan

5

Why test?

- Prevent regressions
- Improves code quality
- Refactor with confidence
- Deploy with confidence
- Increases velocity
- They are required for contributions

What to test?



What to test?

- Access control
- Workflows
- Email notifications
- Form submissions
- API integrations

ANYTHING

YOU WOULD PREFER

NOT TO MANUALLY TEST!

Bug fixes

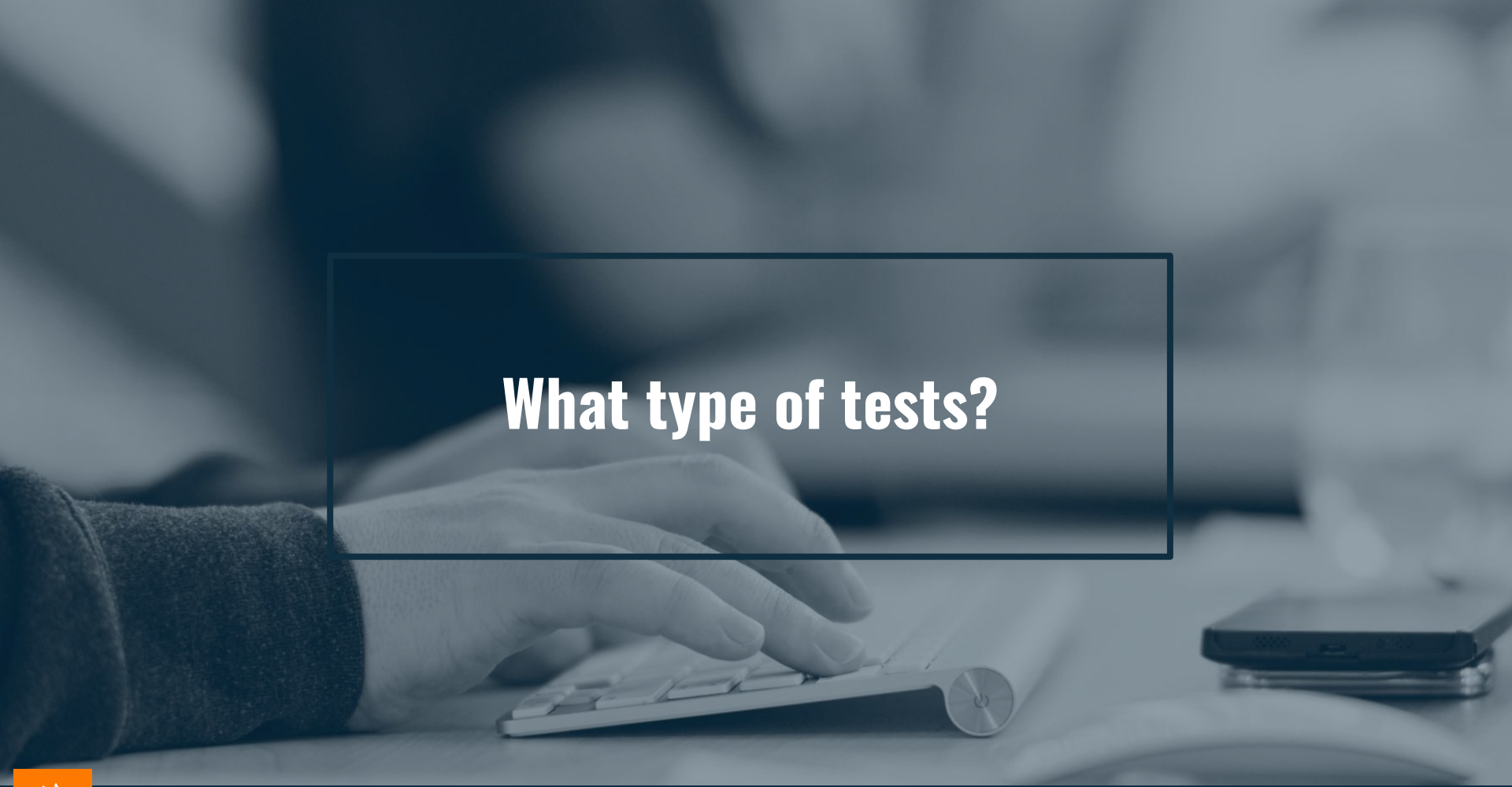
- Reproduce the bug via a test
- Ensure the test fails
- Fix the bug

Features

- Use acceptance criteria as a guide
- Don't just test the happy path

Automate

- Mock API return values
- Mimic many different user roles
- Fill out tedious form submissions
- Verify email


What type of tests?

1

Unit tests

- No database
- Just you and your code
- Every dependency of a class is mocked (use Prophecy)
- `UnitTestCase`

2

Kernel tests

- Only specified modules installed
- Only specified entity types available
- Only specified configuration and schemas installed
- `KernelTestBase`

3

Functional tests

- Full 'testing' install profile
- Can GET and POST requests
- Installs all configuration, schemas, etc.
- Installs module dependencies
- `BrowserTestBase`

4

Functional JavaScript

- Identical to functional tests, but utilizes WebDriver* to test in-browser javascript interactions
- Test module-specific JS
- `JavascriptTestBase`

* as of this morning. PhantomJs previously

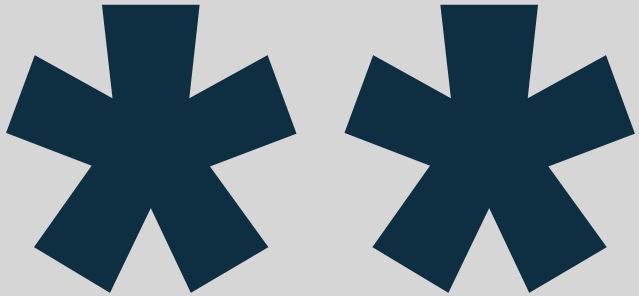
5

Behat

- Tests actual site/project
- Config-dependent scenarios
- Theme-dependent scenarios

Exceptions & Inline asserts

- In the code itself, not separate
- Assert things that should not be possible



Testing a complete site

There is a core issue (#2793445) to allow *Functional* and *FunctionalJavascript* tests to be run against an already installed site.

- Test complex configurations
- Existing user roles
- Existing content types
- Theme-specific tests*

Asserts and Exceptions

Place *asserts* and *exceptions* in custom code to verify implicit assumptions.

```
assert(in_array($node_type, ['alert_customer', 'alert_service']),
  'Unexpected node type: ' . $node_type);
```

```
if (!$object instanceof MyModuleService) {
  throw new UnexpectedServiceException("This isn't the service I was looking for.");
}
```

Boundary Conditions

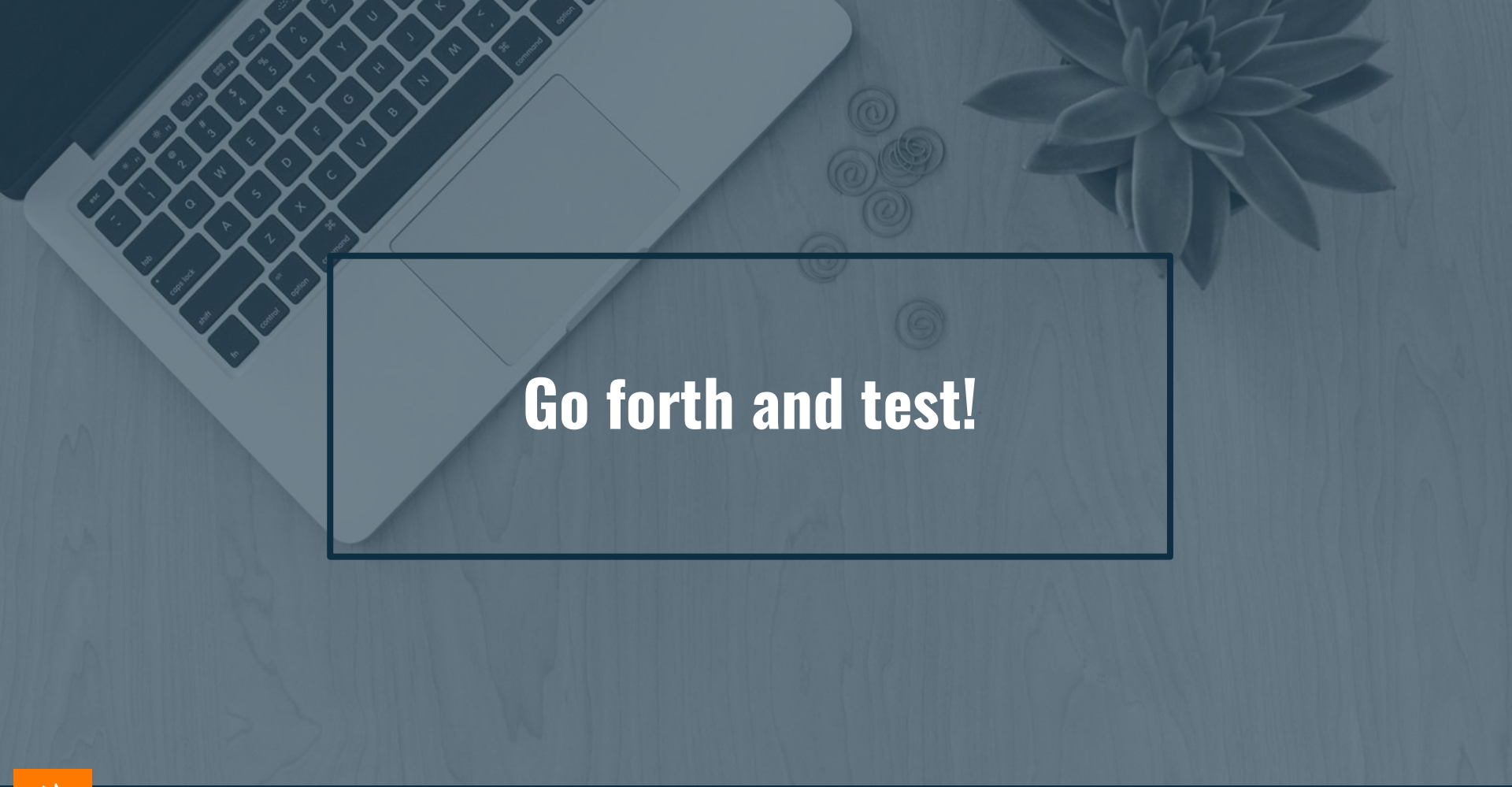
When to switch test type

- Too much mocking
- Too many manual dependency/config installations
- Too many modules
- Too dependent on configuration

Question

Name a suitable test type for this sample issue

- Move the ***Phone*** field below the ***Email*** field on user registration
- Relabel to ***Phone number***
- Make the field optional


Go forth and test!

Running tests

`vendor/bin/phpunit`

```
`vendor/bin/phpunit -c build/html/core/phpunit.xml.dist
--printer="\Drupal\Tests\Listeners\HtmlOutputPrinter"
./src/modules/custom/my_module
--filter=SampleTestClassNameOrMethodName`
```

phpunit.xml

Copy `core/phpunit.xml.dist` to project root as `phpunit.xml`

Edit and define variables

```
<env name="SIMPLETEST_BASE_URL" value="https://foo.dev"/>
<env name="BROWSERTEST_OUTPUT_DIRECTORY" value="/tmp"/>
```

Code coverage reports

```
# Run tests with the code coverage flag and directory.
$> vendor/bin/phpunit --coverage-html=~/code

# View the generated HTML report.
$> open ~/code/index.html

# NOTE: These take a lot of time to generate, so better
# done in a CI environment as a report.
```

Testing exceptions

```
$this->setExpectedException(
  \LogicException::class,
  'Optional expected exception message'
);

$this->doSomethingToTriggerException();
```

Re-usable code

- Traits
- Base classes
- Data Providers

Data providers

```php
/**
 * @dataProvider providerTestAccess
 */
public function testAccess(
  $expected_access,
  array $permissions = [],
  $is_admin = FALSE
) { }
```

```php
/**
 * Data provider for `::testAccess`.
 */
public function providerTestAccess() {
  return [
    'No access' => [FALSE, []],
    'Admin access' => [TRUE, [], TRUE],
    'Normal access' => [
      TRUE,
      ['do something permission']
    ],
  ];
}
```

Prophecy

- Creates a 'mock' of a provided class or interface
- Methods all return `null` by default
- Set expected return values to verify internal working of method being tested

Create the mock object

```php
$mock_service = $this->prophesize(LanguageManagerInterface::class);

$mock_service =
  $this->prophesize('\Drupal\Core\Language\LanguageManagerInterface');
```

Expected method calls

```
$mock_service->isMultilingual()->willReturn(TRUE);

$mock_service->getLanguageName('en')->willReturn('English');
```

The reveal

```
$service = $mock_service->reveal();
```

Getting started

- Find examples
- Write tests for core issues (search tag 'needs tests')
- Review tests for a module you know well in terms of functionality


Code examples


Questions?