# UNIT TESTING IN SASS

Lindsey Wild

SPARKBOX

# Who am I?

- Front End Developer at Sparkbox in Pittsburgh, PA
- Proud coder for over 6 years
- Previously worked for DICK’S Sporting Goods & Giant Eagle
- Volleyball player, Cleveland Indians fan, rock concert junkie

# WHAT WILL WE LEARN?

# Main Points

- **Refresher on Sass and Unit Testing**
- **Why & When**
  - Use cases
  - Real life scenarios
  - What to test
- **How**
  - Using True
  - Best practices for unit testing

# General Takeaways

- Unit testing best practices
- Evaluating if a project would be a good candidate for Sass unit testing

1 Design System, 6 Brands

6 Brands = Multiple websites

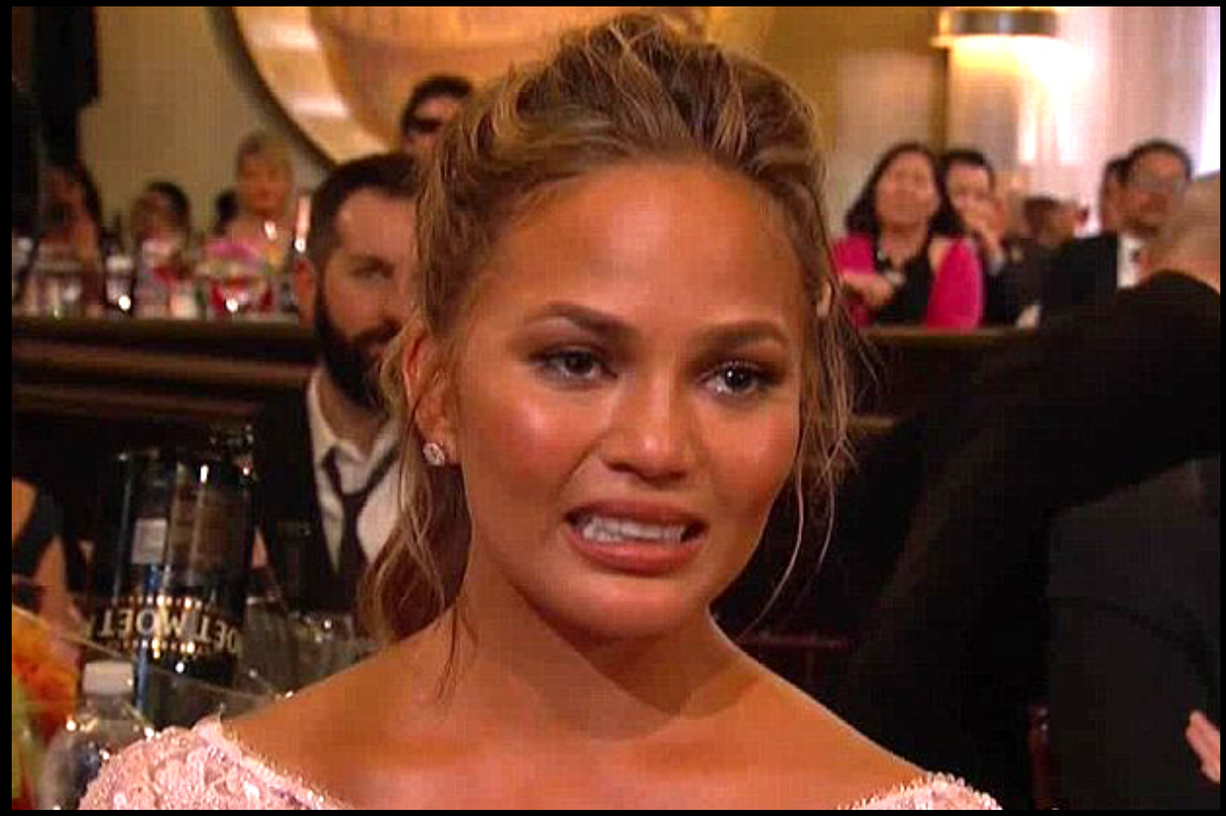
Multiple websites = so many teams!

So many teams = so many subscribers!

# We’re perfect though.

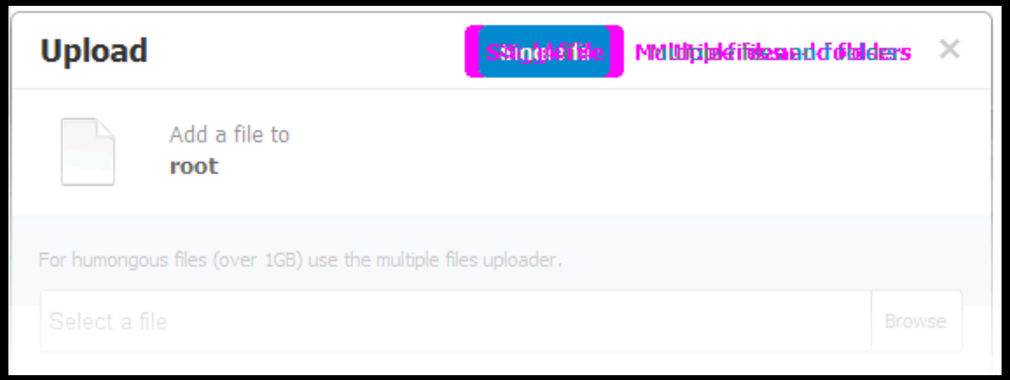
LOL JK

# We've made mistakes.



**38 SCSS Files**
**Each brand: ~8,000 lines of CSS**

# Visual Regression Testing Tools




Screenshot from PhantomCSS: https://github.com/HuddleEng/PhantomCSS

# Can you Unit Test your Sass?

# WHAT IS SASS?

# Sass is a CSS Preprocessor

- More efficient, can use variables, nesting
- Break files into smaller chunks (partials) for better organization, more component driven
- Write functions and mixins to output conditional CSS

# WHAT IS UNIT TESTING?

# Unit Testing: Testing “Units”

- Test small pieces of code, aka “units”
- Ensure functions and other logic is working as expected
- Isolate issues within smaller pieces for faster debugging

# WHY & WHEN TO UNIT TEST SASS

# Why?

- **Alleviate some manual UI regression testing**
- **Ensure output from mixins/functions is as expected**
  - Especially third-party libraries, or Sass itself
- **Double check important updates, such as changing the main color**
- **CSS could still compile without errors**
- **Ship less errors!**

# When

- You’re working on a large project with numerous Sass files/partials
- A component exists in many places with many contexts
- Your project includes a third-party library with complex Sass functions/mixins
- You have complex Sass functions/mixins that you’ve written
- Styles are very important and if they change, it could break the UI dramatically
- Your project supports multiple users or “subscribers”

# When *not* to

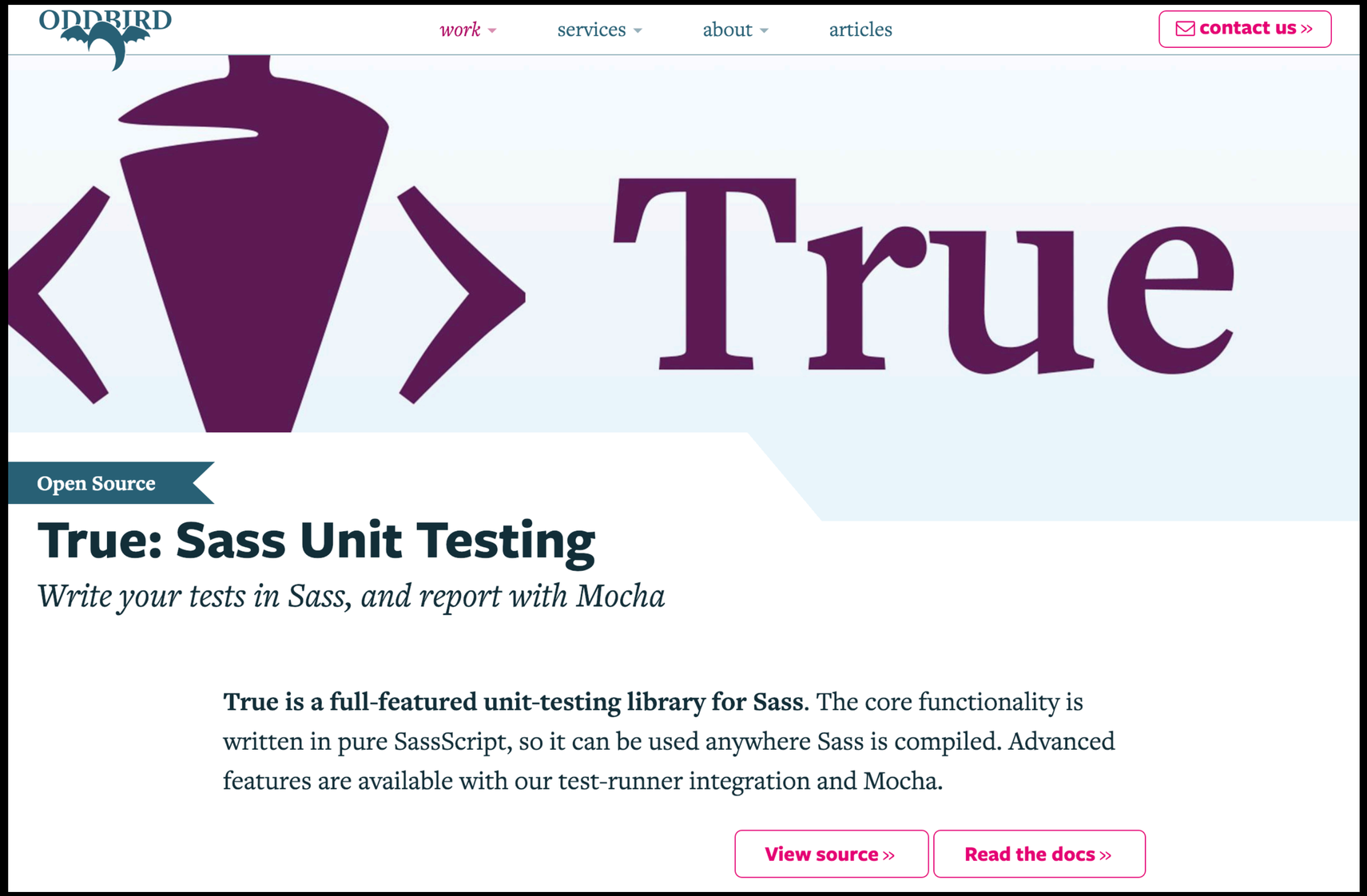
- Small project with little CSS, such as a landing page or small marketing site
- Static content websites
- Your project doesn’t heavily rely on a consistent UI
- Your project doesn’t use Sass mixins or functions

# Real Life Example!

# What to Unit Test

- **Functions**
  - Especially ones that have calculations or take parameters
- **Mixins**
- **Any important output**
- **Variables? Probably overkill**

# HOW TO SET IT UP IN YOUR PROJECT




https://oddbird.net/true

# Requirements

- A computer
- Yarn or npm
- Node-sass or another way to compile Sass to CSS
- Optional: A JS testing library such as Jasmine

# Install the `sass-true` Module

```
// In your Terminal
yarn add sass-true --dev
OR
npm install sass-true —save-dev
```

# Set Up Your File Structure

```
my-cool-project/                          # Root project directory
|-- src/                                  # Project files
|   |-- sass/                             # Sass files
|   |-- js/
|   |-- index.html
|-- spec/                                  # Jasmine tests & setup
|   |-- true-sass-tests/                   # Sass unit tests
|   |   |-- _typography-tests.scss         # Sass unit test partial
|   |   |-- _font-mixin-tests.scss
|   |   |-- _font-function-tests.scss
|   |   |-- sass_tests.scss                # Import Sass test partials
|   true-sass-spec.js                      # True setup
|   ...                                    # Other Jasmine tests/setup
```

# Import the Library & Your Sass

sass_tests.scss

```
// Path to module
@import "node_modules/sass-true/sass/true";
// Path to Sass file(s) to test
@import "../css/styles";
```

# Set Up The Sass Tests

true-sass-spec.js

```
// Node module that provides utilities to work with files & directory paths
const path = require('path');
// Require the module
const sassTrue = require('sass-true');

const sassFile = path.join(__dirname, 'true-sass-tests/sass-tests.scss');
sassTrue.runSass({ file: sassFile }, describe, it);
```

# Define Your Tests

- **Behavior Driven Development (BDD)**
  - Test behavior, not implementation
  - Phrased as “it should do x”
- **Test Driven Development (TDD)**
  - Write your tests before writing your code
  - Run test (will fail)
  - Write minimum code to pass test
  - Test, refactor, repeat

# Testing a Function

```
// Sass
@function multiply($value1, $value2) {
  @return $value1 * $value2;
}

// CSS where function is called
.header {
  width: multiply(12, 2); //24
}

// Test
@include describe('The multiply() function') {
  @include it('Returns the result of multiplication') {
    @include assert-equal(
      // ...
    );
  }
```

# Testing a Function

- Compare internal Sass values (variables and functions) by asserting is-equal, is-unequal, is-true, or is-false:

```
@include describe('The multiply() function') {
  @include it('Returns the result of multiplication') {
    @include assert-equal(
      multiply(12, 2),
      24
    );
  }
}
```

# Terminal Output

- **Passing Test**

```
The multiply() function
  ✓ Returns the result of multiplication
```

- **Failing Test**

```
The multiply() function
  ✗ Returns the result of multiplication
    - AssertionError [ERR_ASSERTION]: Returns the result of multiplication ("[number] 24" assert-equal "[number] 26")
```

# Testing a Mixin

- **Test CSS output of mixins with assert, output, and expect (or contains)**

```
@include describe('The large font mixin') {
  @include it('Returns correct font properties') {
    @include assert {
      @include output {
        // @include mixin to test
      }

      @include expect {
        // font properties
      }
    }
  }
}
```

# Testing a Mixin

```sass
// Sass
@mixin font-size($size) {
  @if $size == 'large' {
    font-size: 2rem;
    line-height: 3rem;
  }
}
```

```sass
// CSS using mixin
.header {
  @include font-size('large');
}
```

```sass
// Test
@include describe('The large font mixin') {
  @include it('Returns correct font properties') {
    @include assert {
      @include output {
        @include font-size('large');
      }

      @include expect {
        font-size: 2rem;
        line-height: 3rem;
      }
    }
  }
}
```

# Mixin Terminal Output

- **Passing Test**

```
The large font mixin
  ✓ Returns correct font properties
```

- **Failing Test**

```
    The large font mixin
      ✗ Returns correct font properties
        - AssertionError [ERR_ASSERTION]:  (".test-output {
  font-size: 2rem;
  line-height: 3rem;
}" equal ".test-output {
  font-size: 2rem;
  line-height: 2rem;
}")
```

# TESTING BEST PRACTICES

UNIT TESTING IN SASS

# Test Driven Development

- Write your tests before writing any code
- Helps to organize thoughts about functionality
- Can lead to finding additional features that need to be included
- Tests are already written!

# Testing Best Practices

- Write tests to fail first
- Only test your code, not a dependency
- Write tests like a readable sentence
- Test small chunks at a time (“unit”)
- Don’t test *everything*

# Write Behavior To Test

```
applies margin vertically and does not reset horizontal sides
applies margin to the bottom only
applies padding horizontally and resets vertical sides
```

# Write Expected Output

```
@include describe('The margin mixin') {
  @include it('applies margin to all sides') {
    @include assert {
      @include output {
        // Our mixin will go here
      }

      @include expect {
        margin: 1rem;
      }
    }
  }
}
```

# Add Mixin/Function To Test

```
@include describe('The margin mixin') {
  @include it('applies margin to all sides') {
    @include assert {
      @include output {
        @include margin-mixin(1rem, all);
      }

      @include expect {
        margin: 1rem;
      }
    }
  }
}
```

# Build The Mixin

```
@mixin margin-mixin($size, $sides) {
  @if $sides == 'all' {
    margin: $size;
  }

  @if $sides == 'top' {
    margin: $size 0 0 0;
  }
  // ...
}
```

# Terminal Output

- **Passing Test**

```
  The margin mixin
    ✓ applies margin to all sides
```

- **Failing Test**

```
    The margin mixin
      ✗ applies margin to all sides
        - AssertionError [ERR_ASSERTION]:  (".test-output {
  margin: 1rem;
}" equal ".test-output {
  margin: 1rem 0 0 0;
}")
```

# Recap

- **Refresher on Sass and Unit Testing**
- **Why & When**
  - Use cases
  - Real life scenarios
  - What to test
- **How**
  - Using True
  - Best practices for unit testing

# RESOURCES

# Links and Resources

- https://oddbird.net/true/docs/
- https://github.com/oddbird/true
- https://seesparkbox.com/foundry/how_and_why_we_unit_test_our_sass
- https://twitter.com/stananick

# THANKS!

@stananick

wild@heysparkbox.com

SPARKBOX