Extracted from:

Test-Driven React

Find Problems Early, Fix Them Quickly, Code with Confidence

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Test-Driven React

Find Problems Early, Fix Them Quickly, Code with Confidence

Trevor Burnham
edited by Jacquelyn Carter

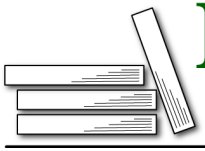
Test-Driven React

Find Problems Early, Fix Them Quickly, Code with Confidence

Trevor Burnham

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Susan Conant
Development Editor: Jacquelyn Carter
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-646-4
Book version: P1.0—June 2019

Introduction

I vividly remember the first time I wrote code. I was 10 years old and utterly obsessed with robots. The local public library must have lent me every book they had on the subject. One of those books had an appendix: "Write Your Own Robot in BASIC." I ran to my parents' computer, fired up `qbasic` (bundled with the cutting-edge MS-DOS operating system), and fed in the instructions for my robot companion.

The program was unimpressive by today's standards. It was a primitive version of what we would now call a "chatbot." It would give you a prompt like

```
>>> Greetings, human. How are you feeling today?
```

then wait for you to enter a recognizable string like `tired`, and give an appropriate response like

```
>>> I am sorry to hear that. How about a nice cup of coffee?
```

and so on. The only keywords in the entire program were `IF`, `THEN`, and `GOTO`.

Even though my chatbot wouldn't stand a snowball's chance in a Turing test, the exercise was a revelation to me: I could actually *create* something just by *typing*. Now it seemed that robots were old news. Computers were where it's at!

As computers have grown more capable, software has grown more complex, and that thrilling feeling has become more elusive. New layers of abstraction have empowered me to do more with less code, but at the cost of constant uncertainty: *Will my code do what I intended?*

Test-driven development (TDD) is the art of minimizing that uncertainty, allowing you to feel confident about your code from the moment you write it. How? By making a few assertions about that code beforehand. This groundwork sets up a short, satisfying feedback loop: as soon as you write your code, the tests light up green. Afterward, the tests remain in place, standing guard against regressions.

I don't always use TDD, but when I do, I feel a little bit closer to the magic of that first coding experience. All of the rigamarole of modern software development fades away. I can focus all my energies on reaching toward the green light.

What's in This Book

This is a book about React. But it's not like any other book about React. This is a book about writing React code in a joyful way. You might learn a few new things about React, but that's not my goal. My goal is to help you write better code, and to have more fun doing it.

In Chapter 1, Test-Driven Development with Jest, on page ?, you'll get a taste of test-driven development, a programming methodology that uses tests to create a feedback loop as you work. For our test framework, you'll use the lightning-fast Jest library.

Chapter 2, Integrated Tooling with VS Code, on page ? will introduce you to some of my favorite tools: VS Code, an amazingly powerful editor; ESLint and Prettier, the ultimate code beautification duo; and Wallaby.js, which I can only describe as pure magic. You'll experience the wonder of instantaneous feedback as you code.

Then in Chapter 3, Testing React with Enzyme, on page ?, you'll start writing React components and testing them with the Enzyme library. You'll get acquainted with Babel, the compiler that lets you write the JavaScript of the future, *today*. You'll build a complex component the TDD way, with 100% code coverage.

Chapter 4, Styling in JavaScript with Styled-Components, on page ? is all about style. You'll use the styled-components library to add pizzazz to our React components. All of these styles will be defined in JavaScript, making them easy to test. You'll start using one of Jest's most powerful features: snapshots. And you'll set up webpack so that you can view the fruits of your labor in the browser.

In Chapter 5, Refactoring with Higher-Order Components, on page ?, you'll learn some important techniques for refactoring React components. You'll extract pieces of functionality into higher-order components (HOCs), encouraging code reuse and allowing core components to stay small and easy to test. And you'll look at your components with X-ray vision through the power of the React Devtools.

Finally, in Chapter 6, Continuous Integration and Collaboration, on page ?, you'll meet all the tools you'll need to share what you have built with the world. You will run your tests in the cloud with Travis CI, enforce your project's rules with Husky, and create beautiful, interactive documentation with Storybook.

What's Not in This Book

This is not an introduction to JavaScript. If you're new to the language, or if you just want a refresher, I highly recommend Kyle Simpson's excellent *You Don't Know JS*[1] series. Most of the code in this book will employ features added to the language as part of the ECMAScript 6 (also known as ES6 or ES2015) standard. Here's a quick test:

```
const stringifyAll = (...args) => args.map(String);
```

If any of that syntax is confounding, you'll find clarity in the *ES6 & Beyond* volume of Simpson's book series.

Some familiarity with React is helpful, but not required. I'll give a brief explanation for each React concept we encounter. If you want a more thorough introduction, pick up Ludovico Fischer's *React for Real.*[2]

All tests in this book are unit tests, meaning the JavaScript code is tested in isolation. We won't be covering integration tests (e.g., a test where the JavaScript code talks to a database) or functional tests (e.g., a test where a tool like Selenium interacts with the code as it runs in a browser). While I don't quite believe that such tests are a scam,[3] the fact is that they require far more effort to implement, run, and maintain than unit tests. My advice is to achieve a high degree of code coverage with unit tests first, then add more layers of testing as needed.

How to Read the Code Examples

This book takes a hands-on, project-driven approach, which means that source files often change over the course of a chapter. When a code example is a work in progress, its file name (relative to the project root) is shown as a comment at the top of the snippet:

1. https://github.com/getify/You-Dont-Know-JS
2. https://pragprog.com/book/lfreact/react-for-real
3. http://blog.thecodewhisperer.com/permalink/integrated-tests-are-a-scam

```
// src/tests/MyComponent.test.js
import React from 'react';
import MyComponent from '../MyComponent';

describe('MyComponent', () => {
  it('renders a <div /> tag', () => {
    const wrapper = shallow(<MyComponent />);
    expect(wrapper.type()).toBe('div');
  });
});
```

As a source file changes over the course of a chapter, familiar sections are omitted with ... and new/edited lines are highlighted:

```
// src/tests/MyComponent.test.js
...

describe('MyComponent', () => {
  ...

  it('accepts a `className` prop', () => {
    const wrapper = shallow(<MyComponent className="test-class" />);
    expect(wrapper.hasClass('test-class')).toBe(true);
  });
});
```

The final version of a source file within a chapter has a download link at the top instead of a comment:

intro/src/tests/MyComponent.test.js

```
import React from 'react';
import MyComponent from '../MyComponent';

describe('MyComponent', () => {
  it('renders a <div /> tag', () => {
    const wrapper = shallow(<MyComponent />);
    expect(wrapper.type()).toBe('div');
  });

  it('accepts a `className` prop', () => {
    const wrapper = shallow(<MyComponent className="test-class" />);
    expect(wrapper.hasClass('test-class')).toBe(true);
  });

  it('triggers `onClick` when clicked', () => {
    const onClick = jest.fn();
    const wrapper = shallow(<MyComponent onClick={onClick} />);
    wrapper.simulate('click');
    expect(onClick).toHaveBeenCalled();
  });
});
```

Online Resources

You can find the source code for the projects in this book on the PragProg website.[4] You can also use the site to report errata. Help make this book better for other readers!

Mantra: Code with Joy

At its best, coding is an exercise in imagination and exploration, an exciting journey into the unknown. At its worst, it feels like stumbling in the dark. Which kind of experience you'll have is largely determined by feedback. The next time you're feeling frustrated, take a step back and ask yourself what kind of feedback would help you move forward. What question can you ask about your code that would bring clarity? Can you turn that question into a test?

I hope this book will help you bring more joy to your work by instilling a habit of seeking feedback early and often. Let's begin!

Trevor Burnham
trevorburnham@gmail.com
Cambridge, MA, June 2019

4. https://pragprog.com/book/tbreact/test-driven-react