Extracted from:

Test-Driving JavaScript Applications

Rapid, Confident, Maintainable Code

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

Test-Driving JavaScript Applications

Rapid, Confident, Maintainable Code

Venkat Subramaniam

Edited by Jacquelyn Carter

Test-Driving JavaScript Applications

Rapid, Confident, Maintainable Code

Venkat Subramaniam

The Pragmatic Bookshelf

Raleigh, North Carolina

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (index)
Liz Welch (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

Printed in the United States of America.
ISBN-13: 978-1-68050-174-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—October 2016

Introduction

The passion to code was seeded in me early. Decades later, though I'm still addicted to coding, the economics of software development equally intrigues me. The cost of change and the speed at which we can respond to change matters a lot. We work in an industry that's still developing and maturing. We get paid to write code, and if we do a poor job, we get paid to go back and fix the mess. That can become a vicious cycle. As professionals we have to raise that bar—a lot higher.

JavaScript is truly a black swan[1]—who would've thought it would grow into one of the most ubiquitous languages? It's powerful, highly flexible, and quite dangerous, all at the same time. I actually like JavaScript—quite a lot—for the power and flexibility it offers.

Shunning powerful and flexible solutions, just because they are risky, will set us back. Instead we have to embrace them with better engineering practices. Using automated testing and continuous integration are part of better engineering practices. We can rely on automated testing and short feedback loops to alleviate the pain that comes from the dangerous side of JavaScript.

There's been explosive growth in the JavaScript ecosystem in recent years. There's also been tremendous development of automated testing tools. Thanks to these tools, as well as short feedback loops, continuous integration, and so on, good engineering discipline is not a theory—it's what every programmer using JavaScript can apply today. I've written this book to motivate and inspire you, and guide you along so you can do your share to raise that bar, to elevate our field to higher standards. Thank you for reading this book.

1. https://en.wikipedia.org/wiki/Black_swan_theory

What's in This Book?

This book focuses on automated testing and related practices necessary to sustain a rigorous development pace. From this book, you'll learn to apply tools and techniques to automatically verify both client-side (including jQuery and Angluar) and server-side (Node.js and Express) JavaScript applications.

You'll learn to effectively use the following tools in this book:

- Karma
- Mocha
- Chai
- Istanbul
- Sinon
- Protractor

While exploring these tools, you'll also learn and use quite a few software design principles that lead to lightweight design and maintainable code. If you're already using other tools, like Jasmine or Nodeunit, for example, you can readily use the testing techniques presented in this book with those tools.

The book is organized into two parts. Part I covers the fundamentals of automated testing. Here, you'll learn how to write tests, both for synchronous and asynchronous functions. You'll also learn how to approach automated testing when the code involves intricate dependencies. In Part II you'll apply what you learned in Part I to write automated tests for both client side and server side, by test-driving a practical example.

The chapters are organized as follows:

Chapter 1, *Automation Shall Set You Free,* on page ? brings out the reasons why automated verification is critical for sustainable development.

Chapter 2, *Test-Drive Your Design,* on page ? walks you through the steps to create automated tests for both server-side code and client-side code with a small example. You'll learn to create a test list and develop code incrementally, implementing the minimum code for one test at a time.

Some asynchronous functions take in callbacks while some may return a `Promise`. Chapter 3, *Test Asynchrony,* on page ? presents the challenges asynchronous functions pose and different ways to test them.

Dependencies are pervasive in both client side and server side. They can make testing really hard, brittle, nondeterministic, and slow. Chapter 4, *Tactfully Tackle Dependencies,* on page ? demonstrates how to remove dependencies where possible. For intricate dependencies, it shows how to decouple and replace dependencies with test doubles to facilitate testing.

Chapter 5, *Test-Drive Node.js Apps*, on page ? walks you through test-driving a fully functional server-side application. It illustrates how to start with a high-level strategic design and to evolve the design using tests. You'll measure the code coverage to get a feel for how much the automated tests verified as the code evolved.

With Express you can breeze through writing web applications. Chapter 6, *Test-Drive Express Apps*, on page ? will teach you how to sustain that pace with automated tests to create maintainable code. You'll start with the design of automated tests for database connections; then you'll learn about the model functions, all the way through to the routes functions.

Chapter 7, *Working with the DOM and jQuery*, on page ? focuses on creating automated tests for the client side of the application developed in the previous chapter. It shows how to write tests for code that directly manipulates the DOM and also code that relies on the jQuery library.

AngularJS is declarative, reactive, and highly fluent. Besides making it easy to write client-side code, the framework also provides a powerful set of tools to write automated tests. In Chapter 8, *Using AngularJS*, on page ? you'll learn the techniques to test AngularJS 1.x applications by creating another version of the client side for the Express application.

To say that AngularJS has gone through a makeover is an understatement. Angular 2 is different from AngularJS 1.x in many ways—components instead of controllers, pipes instead of filters, more explicit dependency injection, annotation-based wiring—and it's written using TypeScript instead of JavaScript. In Chapter 9, *Test-Drive Angular 2*, on page ? you'll re-create the client side from the previous chapter, from the ground up, test first, using Angular 2 and good old JavaScript.

End-to-end, or UI-level, testing is essential but has to be minimal. It should focus and cover parts that aren't covered through the other tests. In Chapter 10, *Integrate and Test End-to-End*, on page ? you'll learn what areas to focus on, what's critical to test, and what should be avoided. The chapter also shows how to create fully automated integration tests, from the database layer, through the model functions, the routes, and all the way to the UI.

Chapter 11, *Test-Drive Your Apps*, on page ? brings the details from the rest of the book together. It discusses how we approached automated testing through the examples, the levels of testing, the size of tests, and the benefits reaped. It concludes the book with a discussion of how to take this forward for your own projects.

Who Is This Book For?

If you program in JavaScript, this book is for you. Programmers, hands-on architects, team leads, technical project managers, and anyone who wants to create maintainable JavaScript applications at a sustainable pace will benefit from this book.

The book assumes programmers are familiar with JavaScript—it does not teach any of the language features. Different parts of the book also assume familiarity with technologies used to create examples in that part. For example, Chapter 8, *Using AngularJS*, on page ? and parts of Chapter 10, *Integrate and Test End-to-End*, on page ? assume that the readers know AngularJS 1.x, but the rest of the book does not depend on that knowledge.

Every page in this book has something for programmers to directly apply on their projects, be it unit testing, integration testing, code coverage, or simply learning about reasons to practice the techniques. Architects and technical leads can use this book to guide their teams toward better technical practices that have an impact on a sustainable pace of development. Technical project managers can learn the reasons to test-drive JavaScript applications, understand the levels and size of testing, and decide how their team can approach automated testing. They can also use the book to motivate their teams to create applications with rapid feedback loops.

If you're already familiar with these techniques, you could use this book to influence and train other developers.

This book is written for software practitioners who extensively use JavaScript and care about their craft.

Online Resources

You can download all the example source code for the book from the Pragmatic Bookshelf website for this book.[2] You can also provide feedback by submitting errata entries or posting your comments and questions in the forum.

If you're reading the book in PDF form, you can click on the link above a code listing to view or download the specific examples.

2. http://www.pragprog.com/titles/vsjavas

A number of web resources referenced throughout the book are collected for your convenience in Appendix 1, *Web Resources*, on page ?.

Your Own Workspace

As you read along you'll want to practice the code examples. This requires installing various tools and creating directories and files for each code example. These steps can quickly get mundane and tedious. You can alleviate most of that pain by using the pre-created workspace provided as part of this book.

The workspace mainly contains package.json files that describe the dependencies and tools that are needed to code each programming example. It also includes the appropriate directory structures needed, along with files that you can readily open and start editing. When you start coding an example, you do not have to manually and repeatedly type in the installation commands for all the needed tools. Instead, once you change directory to an example project, a single npm install command will download everything you need to your system. Once the installation is done, you can conveniently start playing with the example by keying in the tests and the necessary code.

At this time, take a minute to download the workspace from the Pragmatic Bookshelf Media link[3] for this book. On Windows systems, download the file tdjsa.zip to the c:\ directory. On other systems, download it to your home directory. Once the download is complete, unzip the file using the command

```
unzip tdjsa.zip
```

Now you should see a directory named tdjsa with different subdirectories. Each chapter will direct you to the appropriate subdirectories for you to practice along as you make your way through the examples.

A word of caution: In general, copying and pasting code from a PDF reader into editors or IDEs does not work well. The outcome depends on both the source PDF reader and the destination editor/IDE. Depending on the tools used, after copying and pasting, you may have to reformat the code or it may not work. Your mileage may vary. Instead of copying and pasting from the PDF reader, you may download the code from the book's website as described in *Online Resources*, on page x.

Venkat Subramaniam

October 2016

3. https://media.pragprog.com/titles/vsjavas/code/tdjsa.zip