



Community Experience Distilled

Selenium Design Patterns and Best Practices

Build a powerful, stable, and automated test suite using Selenium WebDriver

Foreword by Jim Evans, core contributor to the WebDriver project

Dima Kovalenko

[PACKT] open source*
PUBLISHING community experience distilled

Selenium Design Patterns and Best Practices

Build a powerful, stable, and automated test suite using Selenium WebDriver

Dima Kovalenko



BIRMINGHAM - MUMBAI

Selenium Design Patterns and Best Practices

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: September 2014

Production reference: 1170914

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78398-270-7

www.packtpub.com

Cover image by Jeremy Segal (info@jsegalphoto.com)

Credits

Author

Dima Kovalenko

Reviewers

Anuj Chaudhary

Dave Haeffner

Dave Hunt

Alex Kogon

Commissioning Editor

Usha Iyer

Acquisition Editor

Llewellyn Rozario

Content Development Editor

Priya Singh

Technical Editor

Shiny Poojary

Copy Editors

Roshni Banerjee

Adithi Shetty

Project Coordinators

Judie Jose

Swati Kumari

Proofreaders

Simran Bhogal

Maria Gould

Ameesha Green

Indexers

Monica Ajmera Mehta

Rekha Nair

Priya Sane

Production Coordinator

Kyle Albuquerque

Cover Work

Kyle Albuquerque

Foreword

"But wouldn't we be much more efficient if we could just record our tests and play them back?" Chris, the QA manager, stood at my desk looking for confirmation.

I recall my mouth actually hanging open for a moment, then stammering something like, "What the...I don't even...Wait, what?"

I was working for a small company that produced off-the-shelf software for small- to medium-sized businesses. As part of the product line, it had a client-server desktop application, which also featured a web portal. I had spent the previous couple of years working with a small team of colleagues to create a successful automated testing framework for the desktop application. We built it from the ground up and automated a significant portion of the testing of the desktop application with it. We had intentionally left the testing of the web portal to be done manually, with the intention to automate it later. The company had also recently purchased another company that provided a web-only product intended for use by larger enterprise customers. With the purchase of the other company, automating the tests for the web products was becoming more important.

Additionally, we'd already gone through the process of tool evaluation for the automated testing of the web products. We knew that as a small company, we didn't have a huge budget to purchase expensive commercial testing tools. In fact, the budget was nonexistent; we'd have to make do with tools that were free or nearly so, and wire them together ourselves. Given that both web products supported multiple browsers, we had landed on Selenium as our solution, specifically choosing the newer WebDriver API over the older remote control API.

Mistakenly taking my apparent confusion for his having interrupted me from a tricky bit of coding, Chris pressed on to explain, "I mean, you've done great on the desktop application, but as you said, you need to be a programmer to effectively use those tests. That's great for you and Barbara because you've been working on the framework and understand how to code. The new guy, Derek, has some skills there too, and he's been able to use it pretty well. However, that kind of leaves out Cindy, Josh, and Brian. Wouldn't it be great if we could just use the Selenium IDE to record those test for the websites? Then, they could get automated tests into the suite too. We could even get Christian, the business analyst, in on it too!" Just like that, we'd started down a path; one that you may have started down yourself.

Dima Kovalenko's approach discusses problems that nearly every Selenium user has encountered at one time or another. His knowledge of the subject is born from years of experience, and that hard-won knowledge is now available to you in this very volume. By applying the patterns found here, you can navigate your way to efficient solutions to those problems. Additionally, Dima's writing style uses consistent examples throughout, and the language is engaging and easy to follow.

I envy you, dear reader. Douglas Adams, author of *The Hitchhiker's Guide to the Galaxy*, once wrote, "Human beings, who are almost unique in having the ability to learn from the experience of others, are also remarkable for their apparent disinclination to do so." If we had a book like this in the situation I described earlier, our Selenium implementation would have been much smoother. Whether you're reading this because you are looking to acquire more knowledge about Selenium on your own, or whether you've been told to use Selenium by someone else, you now have the opportunity to benefit from the experiences of those who have gone before. Seize that opportunity and enjoy working with Selenium.

Jim Evans

Core contributor to the WebDriver project, musician, and devoted husband and father

About the Author

Dima Kovalenko started his career in 2003 as a quality assurance intern during his summer internship at Rosetta Stone. Since then, he has spent many years testing software in both a manual and automated fashion in companies such as ThoughtWorks, Groupon, and many others. He has participated in many different types of projects, including language-learning software, web e-commerce stores, and legacy maintenance for telecommunication and airline companies. His experience includes support to Ruby, Java, iOS, Android, and PHP projects as an automated tester and software developer.

His first real experience with computers was at the age of 14, shortly after moving to the United States of America from Russia; this encounter has sparked a lifelong passion for technology.

Acknowledgments

This book would not exist without the help and support of the people in my life, who supported and encouraged this passion to grow and develop further. I'd like to thank my wife, Lena Kovalenko, for tolerating and putting up with my neediness and endless torrent of useless trivia. Without her support, and my constant desire to impress her, I would not have taken any rewarding risks in my career. I would also like to thank my parents Nikolay and Svetlana Kovalenko for letting my brother, Vadim, and me learn from our own mistakes and have ample computer time, that is, after the dishes were washed, naturally.

I'd like to thank my family and friends who were supportive in this project and helped me: Lil Kovalenko, Vadim Kovalenko, David Tolley, Steve Fournier II (Steve-o, formally known as "Scuba Steve"), Josiah Weaver, and Alfredo Velasquez.

This book would not be accurate without the help of Alex Kogon, Dave Haeffner, Dave Hunt, and Anuj Chaudhary. Thank you all for considering all of my insane ideas and theories and giving me good feedback.

I'd like to thank Seth Lochen, Andy Duncan, Shinji Kuwayama, and Virgil Bistriceanu for being incredible managers who encouraged me to learn new skills and grow to be a better person.

Finally, I'd like to also thank my coworkers, from whom I've learned more programming skills than any book could have ever taught me: Scott Muc, Isa Goksu, Jack Calzaretta, Surya Gaddipati, Michael Standley, Valdis Vitayaudom, Gregory Blike, Jason Lantz, and Greg Smith.

About the Reviewers

Anuj Chaudhary is a software engineer who enjoys working on software testing and automation. He has vast experience in different testing methodologies such as manual testing, automated testing, performance testing, and security testing. He has worked as an individual contributor and technical lead on various software projects dealing with all stages of the application development life cycle.

He has been awarded Microsoft MVP two times in a row. He also blogs at www.anujchaudhary.com.

He has also reviewed the book *Selenium WebDriver Practical Guide*, Satya Avasarala, Packt Publishing.

I would like to thank my wife, Renu, for always supporting me. I wouldn't have been able to spend extra hours on reviewing this book without her support.

Dave Haeffner is the writer of *Elemental Selenium* (<http://elementalselenium.com>) — a free, once-a-week Selenium tip newsletter that's read by thousands of testing professionals. He's also the creator and maintainer of ChemistryKit (<https://github.com/chemistrykit>), an open source Selenium framework, and the author of *The Selenium Guidebook* (<http://seleniumguidebook.com>). He's helped numerous companies successfully implement automated acceptance testing, including The Motley Fool, ManTech International, Sittercity, and Animoto. He's also the founder/co-organizer of Selenium Hangout and has spoken at numerous conferences and meetings about automated acceptance testing.

Dave Hunt lives in Kent, UK, with his wife and two sons. He has always had a passion for turning mundane tasks into one-click solutions, and when he discovered Selenium back in 2005, his career in software testing and automation development was sealed. He works from home for Mozilla, where he assists teams to create automated tests for their projects – ranging from Mozilla's web properties to the Firefox web browser and the Firefox OS mobile operating system.

Alex Kogon started programming in 1979 and has been working as an IT professional since 1985, helping small and large companies define and implement business software solutions. He has worked as everything from a Unix Systems Administrator and software tester to Internet start-up company CTO and has been a part of the senior management in a major global Investment Bank.

Since the late 1990s, Alex has been a major proponent of methodologies to improve the design and development of software, leveraging RAD techniques and developing his own pre-Agile methodologies to deliver projects to major global financial institutions in a fraction of the regular time. He now works as a Management Consultant helping organizations leverage Agile methodologies to be more efficient and effective through communication, collaboration, tools, automated testing, continuous integration, coding standards, and pair programming.

His ideas have been published in the Financial Times and Wall Street Journal and his seminal research on Additive Synthesis of Digital Signals is published and referred to frequently in research documents. Alex is currently working on a book on how to save money and improve results in corporate IT with Agile Methodologies.

I'd like to thank Ben and Tilda for providing a counterpoint in my life.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

| | |
|--|-----------|
| Preface | 1 |
| Chapter 1: Writing the First Test | 7 |
| Choosing Selenium over other tools | 8 |
| Right tool for the right job | 8 |
| Price | 8 |
| Open source | 9 |
| Flexibility | 9 |
| The Record and Playback pattern | 9 |
| Advantages of the Record and Playback pattern | 9 |
| Disadvantages of the Record and Playback pattern | 10 |
| Getting started with the Selenium IDE | 10 |
| Installing the Selenium IDE | 11 |
| Recording our first test | 15 |
| Saving the test | 18 |
| Understanding Selenium commands | 19 |
| Reading Selenese | 20 |
| Comparing Ruby to Selenese | 21 |
| Comparing Selenium commands in multiple languages | 24 |
| Writing a Selenium test in Ruby | 25 |
| Introducing Test::Unit | 26 |
| Introducing asserts | 28 |
| Interactive test debugging | 31 |
| Summary | 33 |
| Chapter 2: The Spaghetti Pattern | 35 |
| Introducing the Spaghetti pattern | 36 |
| Advantages of the Spaghetti pattern | 36 |
| Disadvantages of the Spaghetti pattern | 37 |

| | |
|--|-----------|
| Testing the product review functionality | 38 |
| Starting a product review test | 39 |
| Locating elements on the page | 42 |
| Using a browser's element inspector | 44 |
| Introducing locator strategies | 46 |
| Using advanced locator strategies | 48 |
| Writing locator strategy code | 53 |
| Using chained selector strategy methods | 53 |
| Using the CSS selector | 54 |
| Using XPath | 55 |
| Implementing clicks and assertions | 55 |
| Duplicating the product review test | 59 |
| Reasons for failures | 59 |
| The Chain Linked pattern | 61 |
| The Big Ball of Mud pattern | 61 |
| Summary | 62 |
| Chapter 3: Refactoring Tests | 63 |
| Refactoring tests | 64 |
| The DRY testing pattern | 64 |
| Advantages of the DRY testing pattern | 65 |
| Disadvantages of the DRY testing pattern | 65 |
| Moving code into a setup and teardown | 66 |
| Removing duplication with methods | 68 |
| Removing external test goals | 69 |
| Using a method to fill out the review form | 70 |
| Reviewing the refactored code | 70 |
| The Hermetic test pattern | 72 |
| Advantages of the Hermetic test pattern | 72 |
| Disadvantages of the Hermetic test pattern | 73 |
| Removing test-on-test dependence | 74 |
| Using timestamps as test data | 74 |
| Extracting the remaining common actions to methods | 76 |
| Reviewing the test-on-test dependency refactoring | 79 |
| Creating generic DRY methods | 80 |
| Refactoring with generic methods | 81 |
| The random run order principle | 82 |
| Advantages of the random run order principle | 82 |
| Disadvantages of the random run order principle | 83 |
| Summary | 84 |

| | |
|--|------------|
| Chapter 4: Data-driven Testing | 85 |
| Data relevance versus data accessibility | 86 |
| Hardcoding input data | 87 |
| Hiding test data from tests | 87 |
| Choosing the test environment | 90 |
| Introducing test fixtures | 92 |
| Parsing fixture data | 92 |
| Using fixture data in the tests | 93 |
| Using fixtures to validate products | 95 |
| Testing the remaining products | 97 |
| Using an API as a source of fixture data | 102 |
| Using data stubs | 104 |
| The default values pattern | 105 |
| Advantages of the default values pattern | 105 |
| Disadvantages of the default values pattern | 106 |
| Merging the default values pattern and the faker library | 106 |
| Implementing faker methods | 106 |
| Updating the comment test to use default values | 108 |
| Summary | 111 |
| Chapter 5: Stabilizing the Tests | 113 |
| Engineering the culture of stability | 114 |
| Running fast and failing fast | 114 |
| Running as often as possible | 115 |
| Keeping a clean and consistent environment | 115 |
| Discarding bad code changes | 116 |
| Maintaining a stable test suite | 116 |
| Waiting for AJAX | 116 |
| Testing without AJAX delays | 118 |
| Using explicit delays to test AJAX forms | 121 |
| Implementing intelligent delays | 123 |
| Waiting for JavaScript animations | 125 |
| The Action Wrapper pattern | 128 |
| Advantages of the Action Wrapper pattern | 128 |
| Disadvantages of the Action Wrapper pattern | 129 |
| Implementing the Action Wrapper pattern | 129 |
| The Black Hole Proxy pattern | 134 |
| Advantages of the Black Hole Proxy pattern | 135 |
| Disadvantages of the Black Hole Proxy pattern | 135 |
| Implementing the Black Hole Proxy pattern | 136 |
| Test your tests! | 139 |
| Summary | 139 |

| | |
|--|------------|
| Chapter 6: Testing the Behavior | 141 |
| Behavior-driven Development | 141 |
| Advantages of BDD | 142 |
| Disadvantages of BDD | 142 |
| Testing the shopping cart behavior | 143 |
| Describing shopping cart behavior | 146 |
| Writing step definitions | 147 |
| Is BDD right for my project? | 149 |
| Introducing Cucumber | 149 |
| Feature files | 150 |
| Step definition files | 151 |
| The configuration directory | 152 |
| Cucumber.yml | 152 |
| env.rb | 154 |
| Running the Cucumber suite | 154 |
| The write once, test everywhere pattern | 155 |
| Advantages of the write once, test everywhere pattern | 156 |
| Disadvantages of the write once, test everywhere pattern | 156 |
| Testing a mobile site | 156 |
| Updating the Selenium wrapper | 157 |
| Moving step definition files | 157 |
| Updating the Cucumber profile and tagging tests | 159 |
| Running and fixing incompatible steps | 160 |
| Testing the purchase API | 163 |
| Summary | 167 |
| Chapter 7: The Page Objects Pattern | 169 |
| Understanding objects | 170 |
| Describing a literal object | 170 |
| Object properties | 170 |
| Object actions | 171 |
| Objects within objects | 171 |
| Describing a programming object | 171 |
| Describing a web page with objects | 172 |
| The Page Objects pattern | 174 |
| Advantages of the Page Objects pattern | 174 |
| Disadvantages of the Page Objects pattern | 176 |
| Creating a Page Objects framework | 176 |
| Creating a page super class | 177 |
| Implementing sidebar objects | 179 |
| Implementing the SidebarCart class | 180 |

| | |
|---|------------|
| Adding Self Verification to pages | 182 |
| Implementing individual page classes | 183 |
| Increasing the number of sidebar objects as the website grows | 188 |
| Running tests with the Page Objects framework | 189 |
| Using Page Objects in the Test::Unit framework | 189 |
| Using Page Objects in different testing frameworks | 191 |
| Looking at the Cucumber implementation | 191 |
| Looking at the RSpec implementation | 193 |
| The test tool independence pattern | 194 |
| Advantages of the test tool independence | 195 |
| Disadvantages of the test tool independence | 195 |
| The right way to implement Page Objects | 195 |
| Making pages smarter than tests | 196 |
| Making tests smarter than pages | 197 |
| Using modules instead of inheritance | 198 |
| Placing logic in Page Objects | 198 |
| Summary | 199 |
| Chapter 8: Growing the Test Suite | 201 |
| <hr/> | |
| Strategies for writing test suites | 202 |
| Different types of tests | 202 |
| The smoke test suite | 203 |
| The money path suite | 204 |
| New feature growth strategy | 205 |
| Bug-driven growth strategy | 206 |
| The regression suite | 206 |
| The 99 percent coverage suite | 206 |
| Continuous Integration | 207 |
| Managing the test environments and nodes | 208 |
| Deploying new builds | 209 |
| CI environment management | 209 |
| Selenium Grid | 211 |
| Understanding standalone and grid modes | 212 |
| Installing Selenium Grid | 214 |
| Choosing the CI tool | 216 |
| Decoupling tests from tools | 217 |
| Frequently Asked Questions | 219 |
| How to test on multiple browsers? | 219 |
| Problem | 219 |
| Possible solutions | 219 |
| Which programming language to write tests in? | 220 |

Table of Contents

| | |
|--|------------|
| Should we use Selenium to test the JS functionality? | 221 |
| Problem | 221 |
| Possible solution | 222 |
| Why should I use a headless browser? | 222 |
| Possible solution | 222 |
| Which BDD tool should I use on my team? | 223 |
| Problem | 223 |
| Possible solutions | 224 |
| Can I use Selenium for performance testing? | 224 |
| Problem | 225 |
| Possible solutions | 226 |
| Summary | 226 |
| Appendix: Getting Started with Selenium | 227 |
| Setting up the computer | 227 |
| Using Command Line Interface | 228 |
| Using the terminal on Windows | 228 |
| Using the terminal on Mac OS X | 232 |
| Using the terminal on Linux | 233 |
| Configuring the Ruby runtime environment | 233 |
| Installing Ruby | 234 |
| Installing the selenium-webdriver gem | 234 |
| Installing Firefox | 234 |
| Understanding test class naming | 234 |
| Naming files | 235 |
| Naming classes | 235 |
| Understanding the namespace | 237 |
| Showing object inheritance | 237 |
| Summary | 238 |
| Index | 239 |

Preface

Selenium Design Patterns and Best Practices will help you write better tests!

It does not matter whether you are writing a Selenium WebDriver test to test your website or shell scripts to test the HTTP API of the backend services of your multibillion dollar enterprise application. This book is not purely theoretical work, but comes from years of experience of the author and his colleagues. A lot of the practices and ideas written in this book did not appear as soon as we started to test the software. Instead, they came from years of mistakes, frustrations, and slow but continuous improvement. We do not believe that the examples and topics described in this book are definitive and static solutions to every single problem that you may encounter in your career. Instead, this book shows you some very generic solutions to very common problems that we, an ever-growing community of automated software testers, have encountered. We hope that this book will not only provide quick fixes to the problem(s) you may encounter, but will also empower you to solve more and more complex problems in your career by showing you some very simple improvement techniques.

What this book covers

Chapter 1, Writing the First Test, will guide us through the process of writing a simple Selenium test and converting it to a programming language.

Chapter 2, The Spaghetti Pattern, will help us write our second test that will completely depend on the test we wrote in the first chapter. We will understand why having tests that completely depend on each other is a bad practice.

Chapter 3, Refactoring Tests, will fix some of the pitfalls and common mistakes we encountered so far. This chapter will concentrate on the introduction of good computer programming practices such as code reuse.

Chapter 4, Data-driven Testing, will guide us through making some initial improvements to your test suite, and it will eventually prepare us to examine one of the most difficult problems in software testing: test data.

Chapter 5, Stabilizing the Tests, will help us understand that writing tests alone is not enough. We will dedicate this chapter to making our test bug free and resistant to random instabilities in the test environment.

Chapter 6, Testing the Behavior, will help you discover why testing the application in its current iteration becomes unmaintainable in the long run. Instead, we will start testing the desired behavior of the application, not the implementation.

Chapter 7, The Page Objects Pattern, covers one of the most undervalued and misunderstood topics when it comes to User Interface testing, that is, Page Objects. In this chapter, we will create a working Page Object testing framework and demonstrate how the tests can keep up with the new feature development cycle.

Chapter 8, Growing the Test Suite, will conclude this book with some basic tips on how to prioritize the growth of the test suite. Along the way, we will discuss how to keep our test stable and relevant to the whole team, no matter how often or big the changes are to the application being tested.

Appendix, Getting Started with Selenium, covers the initial setup of the user's computer. We will learn how to use the Command Line Interface terminal on Windows, Mac OSX, and Linux. We will install the Ruby programming language and Selenium WebDriver Ruby gem, followed by installation of the Firefox web browser. It concludes by explaining the test file and class nomenclature so that individuals new to the Ruby programming language can easily follow along with the tests.

What you need for this book

To get started with this book, you will need a basic understanding of what Selenium is, what it does, and basic programming skills. If you are able to create a simple `click` command in Selenium WebDriver and write a simple `loop` program in any programming language, you should be able to keep up with every example in this book. We will take the time to explain every line of code written in this book so that you are able to create the desired outcome in any situation you may find yourself in. There are some very small and simple software prerequisites that are needed. We will need to have access to the Command Line Interface terminal, Ruby runtime environment, and Firefox web browser. You can find the simple step-by-step setup instructions for all of these prerequisites in the *Appendix, Getting Started with Selenium*.

Who this book is for

This book is for anyone who wishes to write better automated tests. Whether you are writing your first Selenium test or have written hundreds of them, you will find this book useful to create a good test suite. However, this book is not only meant for writing better Selenium tests. A lot of the examples and techniques discussed in this book apply not only to Selenium WebDriver, but also to any automated tests written in any programming language.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "You can even open the `search_test.html` file in your web browser and see how it looks!"

A block of code is set as follows:

```
more_info_buttons = special_items.collect do |special_item|
  special_item.find_element(:class, "more-info")
end
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
require 'rubygems'
require 'selenium-webdriver'


selenium = Selenium::WebDriver.for(:firefox)
selenium.get("http://awful-valentine.com/")
selenium.find_element(:id, "searchinput").clear
selenium.find_element(:id, "searchinput").send_keys("cheese")
selenium.find_element(:id, "searchsubmit").click
selenium.quit
```

Any command-line input or output is written as follows:

```
ruby run_tests.rb
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Click on **Install Now** when it becomes clickable after several seconds."

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Writing the First Test

"Self-education is, I firmly believe, the only kind of education there is."

-Isaac Asimov

In this book, we will simulate my personal experience of testing e-commerce systems. We will start by writing a very simple and crude test case, and we will refactor it and grow it into a *stable* and *reliable* test suite. A web store example might not apply to everyone's job, but the examples provided should be general enough to apply to any situation.

Today is our first day on the job; you and I are the sole members of the newly formed Quality Assurance team for the little start-up that sells Valentine's Day cards. It's a small company and the pay is not the greatest; however, just like any small start up, we get some company stock. This means that we can be very rich and famous if the website becomes popular. The website needs to stay operational and bug free, or our customers will never return and I will not be able to purchase that yacht I always wanted.

We know that we are short-staffed and need some automated tests to keep the quality high. However, first we need to convince the owner of the company that test automation is the right direction, instead of just testing everything by hand. We need to provide a cost-effective way to test the website and get quick results!

In this chapter, we will make an argument for using Selenium as our automation tool of choice and write a simple test to show how fast we can start building new tests. We will discuss the following topics along the way:

- Why you should use Selenium over other tools
- The Record and Playback pattern
- The Selenium IDE

- Recording a test with the Selenium IDE
- Selenium WebDriver
- Writing a test with Ruby
- The Test::Unit testing framework
- Interactive test debugging

Choosing Selenium over other tools

There are several reasons to use Selenium over other test automation tools out there:

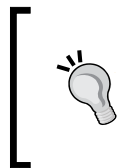
- It is the right tool for the right job
- It is free of cost
- It is open source
- It is highly flexible

Right tool for the right job

Selenium is a great tool for testing web applications and interacting with the application like a real user would. It uses a real browser to click, type, and fill out forms. It is as close to a human user as you can get. It's the perfect tool for testing the flow of the web application from start to finish.

Price

Nothing can beat the free price tag! While there are other commercial products that have more advanced features available for purchase, they tend to run into tens of thousands of dollars per license. Selenium is so cheap that you will be able to finish this book and build a whole test suite without spending another dollar.



As old anti-proverb states: *there is no free lunch, but there is always more cheese in the mousetrap*. A free tool does not mean that the tests will write themselves for free; there will always be expenditure on someone's time. By following good practices, we will not be able to eliminate this cost but will try to reduce it as much as possible in the long run.

