3rd Edition

Practical Web Test Automation

Test Web Applications Wisely

Zhimin Zhan

Practical Web Test Automation with Selenium WebDriver

Test web applications wisely with Selenium WebDriver

Zhimin Zhan

This book is for sale at http://leanpub.com/practical-web-test-automation

This version was published on 2023-10-26

I dedicate this book to my mother and father for their unconditional love.

Contents

Preface

On April 3, 2013, Wired published an article "The Software Revolution Behind LinkedIn's Gushing Profits"[1]. The revolution "completely overhauled how LinkedIn develops and ships new updates to its website and apps, taking a system that required a full month to release new features and turning it into one that pushes out updates multiple times per day." LinkedIn is not alone, Google has accomplished this long before that. As a matter of fact, LinkedIn's success is tracked back to luring a Google veteran in 2001. "Facebook is released twice a day"[2] and they claimed "keeping up this pace is at the heart of our culture"[3].

Release software twice a day! For many, that's unimaginable. You may wonder how they could ensure quality (and you know the high standard from them). The answer is, as the article pointed out, to use "automated tests designed to weed out any bugs."

After working on numerous software projects for a number of years, I witnessed and had been part of many what I call 'release panic syndromes'. That is, with the deadline approaching, the team's panic level rises. Many defects were found from the last round of manual testing by the testers. The manager started prioritizing the defects (or adjusting some to features), and programmers rushed to fix just the critical ones. Testers restarted the testing on the new build that had fixed some but not all the defects. Then here came the bad news: several previously working features are now broken, Argh!

I believe there is a better way to do software development that does not have to involve this kind of stress and panic. This is how my interest in automated testing started (in 2006). I made the right decision to use free, open-source, and programming based test frameworks. (It is quite obvious now, as Selenium WebDriver is the best sought after testing skill on the job market. Back then, people turned to record/playback commercial tools with vendor-proprietary test script syntax). The first test framework I used (for my pet projects) was Watir. I was quickly convinced that this approach was the answer.

In 2007, I had the opportunity to put my approach into practice in a government project. The outcome was beyond everyone's expectation: over two years and countless releases, there were no major defects reported by customers. The team had high confidence in the product. These automated tests also provided the safety net for some major refactorings,

[1] http://www.wired.com/business/2013/04/linkedin-software-revolution/
[2] http://www.facebook.com/notes/facebook-engineering/ship-early-and-ship-twice-as-often/10150985860363920
[3] http://www.seleniumconf.org/speakers/

which would have not been possible without them. A business analyst once said, “before every demonstration to our customers, it is a good feeling of knowing every release has been thoroughly tested.” The synergy of the flexible test framework, maintainable test design, team collaboration with the same simple testing tool and continuous integration supporting functional test execution really made a big difference.

There is now a clearly converging trend in web application development on technology choices, such as cloud deployment, light and productive web frameworks such as “Ruby on Rails”[4], JQuery JavaScript Library, Twitter BootStrap UI themes, Font Awesome icons, …, etc. The competitions among web applications are less on technologies, but weigh more on the development process to ensure pushing out high-quality releases frequently. A fact: Facebook was not the first social networking web site.

A friend of mine, who developed a quite successful public web application, told me in an uneasy tone that he just found out another competitor product at a cheaper price. This is inevitable, the competition among web applications is global, which means, there are people working at 10% of your hourly rate to compete against you. The only way to win the race, in my opinion, is to greatly enhance your productivity and reduce maintenance costs. This can be achieved by applying test automation and continuous integration with instant benefits without much effort (if doing it properly). My reply to my friend: “If your competitors start to invest in test automation seriously, you shall be worried.”

In Appendix II, I share my experience of developing ClinicWise, a modern web-based clinic management system. Thanks to comprehensive automated UI testing, ClinicWise is frequently released (daily) with new features and updates. ClinicWise is developed and maintained in my spare time.

The purpose of this book is to share my journey of test automation for web applications: from writing the first test to developing and maintaining a large number of automated test scripts.

Who should read this book?

Everyone who works on a software team (including testers, programmers, business analysts, architects, and managers) builds a web application and wants to improve the quality of software while saving time and money can benefit from reading this book. It may sound like a bold statement, but it is the outcome I obtained from some projects whose team members have embraced the techniques and practices presented in this book. Those projects delivered reliable software releases frequently, stress-free. You can achieve this too.

[4]http://yourstory.com/2013/02/startup-technologies-top-6-technologies-used-at-startups/

Prior experience with automated testing is not necessary. Basic programming concepts will help, but again, not necessary.

How to read this book?

I strongly recommend readers to read through Chapters 1-9 in order, only skip Chapter 4 if you have decided on the testing editor or IDE. Chapters 10-15 are largely independent of one another. You can, therefore, read them in the order that suits your interests. Readers can also skim through and come back for details later if necessary.

Some chapters contain hands-on exercises (with step by step guides). Typically it will take about 10-30 minutes to complete an exercise. Readers can choose to follow the exercises while or after reading a chapter. The main point is: to master test automation, you have to do it.

What's inside the book?

In part 1, I introduce Web Test Automation and its benefits, which many believe but few actually achieve it. I use a metaphor to illustrate practical reasons why most software projects conduct functional testing manually despite knowing the great benefits of test automation. Then the journey starts with a case study to help write your first Watir automated test in about 10 minutes.

In part 2, I present a brief introduction of test frameworks and tools, followed by a case study showing the development of Selenium WebDriver tests for a live test site with the help of a recorder. Along the way, some testing techniques are introduced.

In part 3, I present an intuitive and maintainable automated test design: using reusable functions and page objects, followed by a case study showing the transforming of recorded test scripts in a maintainable way. Then I introduce an important concept: functional test refactoring, a process of testers applying refactorings to test scripts efficiently with refactoring support in testing tools such as TestWise IDE[5].

With a growing number of automated tests, so is the test execution time. Long feedback loops really slow down development. In part 4, I show how team collaboration and continuous integration can help to improve the feedback time greatly.

[5]https://agileway.com.au/testwise

In Part 5, I switch the attention to several WebDriver backed variant frameworks: Watir, RWebSpec, and Capybara and introduce another test syntax framework Cucumber[6]. Then I will show how to apply the maintainable test design and techniques to them. Finally, I share some strategies to apply test automation to your project.

Test scripts, Screencasts and other resources

To help readers learn test automation more effectively, the book has a dedicated site at: http://zhimin.com/books/pwta[7], which contains the following resources:

- **Software**

  Test automation is not necessarily expensive. All test frameworks featured in this book are free and open-sourced. Testing tools used for the exercises in this book are also free, and there are instructions to cater to other text-based testing tools.

- **Sample test scripts**

  The sample test scripts for the exercises are ready-to-run. This book covers several popular test and syntax frameworks: Selenium-WebDriver, Watir, RWebSpec, RSpec and Cucumber.

  To help readers understand the differences, I have created 6 test projects with different combinations: https://github.com/testwisely/agiletravel-ui-tests[8].

- **Sample web sites**

  For readers who need web sites to try out automated test scripts, I have prepared two test sites for you:

  - *Agile Travel*: a simple flight booking site, which is used in the exercises.
  - *WhenWise*: a feature-rich service-booking web app.

- **Tutorial screencasts**

  There are screencasts for readers who will learn better with audio and video, so you will be able to see how it is done step by step.

For access code see the Resources section of this book.

[6]http://cukes.info/
[7]http://zhimin.com/books/pwta
[8]https://github.com/testwisely/agiletravel-ui-tests

Send me feedback

I will appreciate hearing from you. Comments, suggestions, errors in the book and test scripts are all welcome. You can submit your feedback on the book web site.

Acknowledgements

I would like to thank everyone who sent feedback and suggestions, particularly Mingli Zhou, Darren James, Tim Wilson, Lloyd Blake, Hoang Uong, Scott Cavness and Lien Nguyen, for their time and wisdom.

I owe a huge 'thank you' to people behind great open-source testing frameworks such as Selenium-WebDriver and RSpec, and of course, the beautiful Ruby language.

Functional testing via User Interface is practical and light on theory, so is this book. I hope you find this book useful.

Zhimin Zhan
Brisbane, Australia

1. What is Web Test Automation?

Web Test Automation, or automated functional testing for web applications via the Graphical User Interface (GUI), is the use of automated test scripts to drive test executions to verify that the web application meets its requirements. During the execution of an automated test for a web site, you see mouse and keyboard actions such as clicking a button and typing text in a text box in a browser, without human intervention. Web Test Automation sits under the category of black-box functional testing, where the majority of test efforts are in software projects.

Functional Testing vs Unit Testing vs Non-Functional Testing

Functional testing is to verify function requirements: **what** the system does. For example, "*User can request a password reset by providing a valid email*". Functional testing is the focus of this book.

Unit testing is a type of white box testing performed by programmers at the source code level. It is of no concern to software testers. Unit Test is a term that gets misused a lot. A more correct term would be "Programmer Test". A product that passes comprehensive programmer tests can still fail on many functional tests. That's because unit tests are from a programmer's perspective, and functional tests are from a user's perspective. A programmer test is a kind of automated test too.

Non-functional testing is the testing of **how** the system works. For example, "*The response time of the home page must not exceed 5 seconds.*" Some types of non-functional testings, load testing, in particular, utilize automated test tools as well.

1.1 Test automation benefits

The benefits of test automation are plenty. Below are four common ones:

- **Reliable**. Tests perform the same operations precisely each time they are run, therefore eliminating human errors.

- **Fast**. Test execution is faster than done manually.
- **Repeatable**. Once tests are created, they can be run repeatedly with little effort, even at lunchtime or after working hours.
- **Regression Testing**. “The intent of regression testing is to ensure that a change, such as a bug fix, did not introduce new faults” [Myers, Glenford 04]. Comprehensive manual regression testing is almost impossible to conduct for two reasons: the time required and human errors. As Steve McConnell pointed out, “The only practical way to manage regression testing is to automate it.” [McConnell]

What do I like about test automation?

If you want me to use only one adjective to describe web test automation, it is **fun**. As a software engineer, I enjoy creating something that can do the work for me. I have a habit of triggering the execution of a test suite before going out for lunch. I like the feeling of “still working” while enjoying a meal. When I come back, a test report is there.

As a product owner, I think it is essential to release software frequently without fear. To achieve that, comprehensive test automation via UI is a must.

1.2 Reality check

With more software projects adopting agile methodologies and more software application developments moving towards the Web, you would assume web test automation would be everywhere now. The answer is, sadly, no. In fact, functional testing in many projects is still executed in pretty much the same way: manually. For the past two decades, I have seen automated UI testing was done poorly or not at all in numerous “agile” projects, however, it has been talked a lot. Michael Feathers, a renowned agile mentor and the author of Working Effectively with Legacy Code[1], summarized better than what I can in this blog article.

[1]http://www.amazon.com/Working-Effectively-Legacy-Michael-Feathers/dp/0131177052

UI Test Automation Tools are Snake Oil - Michael Feathers

It happens over and over again. I visit a team and I ask about their testing situation. We talk about unit tests, exploratory testing, the works. Then, I ask about automated end-to-end testing and they point at a machine in the corner. That poor machine has an installation of some highly-priced per seat testing tool (or an open-source one, it doesn't matter), and the chair in front of it is empty. We walk over, sweep the dust away from the keyboard, and load up the tool. Then, we glance through their set of test scripts and try to run them. The system falls over a couple of times and then they give me that sheepish grin and say "we tried." I say, "don't worry, everyone does." [Feathers 10]

Michael Feathers was spot on summarizing the status of automated end-to-end testing. Over the last 20 years, I pretty much experienced the same: software teams were either 'pretending automated UI testing' or 'not doing it at all'.

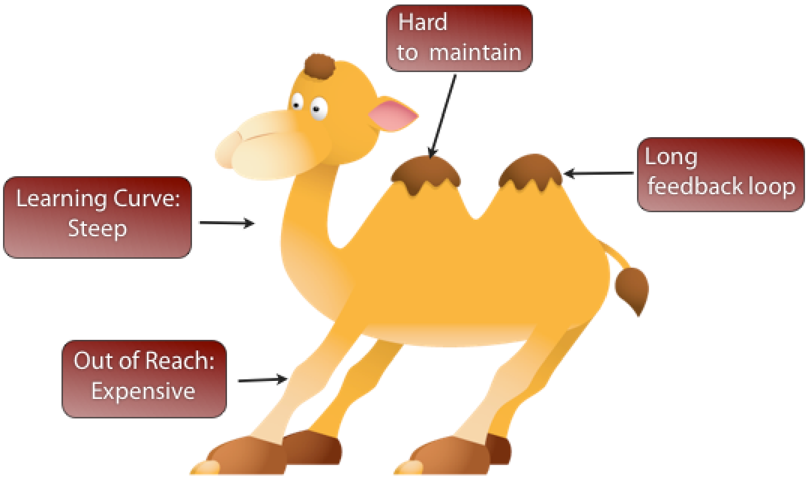
1.3 Reasons for test automation failures

The software testing survey conducted by Innovative Defense Technologies in 2007 [IDT07] shows *"73% of survey respondents believe Automated Testing is beneficial but few automate"*. The top reasons for survey participants not automating their software testing (while agreeing with the benefits) are:

- lack of time
- lack of budget
- lack of expertise

These reasons sound right to most people. However, saving time and money are two benefits of test automation, isn't that a contradiction (for lack of time and budget)? What are the real difficulties or challenges, apart from political or project management ones, that projects encounter during their adventures in automated testing?

To make it easy to understand, we can compare a project's test automation attempt with a person who is trying to climb over a standing two-hump camel from the front. Let's consider each of the following challenges he faces:



Test Automation Camel

Figure 1-1 Test Automation Camel (graphics credit: www.freevectordownload.com)

1. Out of reach: Expensive

Commercial testing tools are usually quite expensive (I won't list prices here, in fact, I couldn't get prices for some so-called leading testing tools on their web sites, which is telling in itself). Automated testing is one of a few activities in software projects that the whole team can contribute to and benefit from. Besides testers, programmers may run automated tests for self-verification and business analysts may utilize automated tests for customer demonstrations. However, the high price of commercial testing tools makes the whole team's adoption of automated testing unfeasible.

There are free, open-source testing frameworks, such as Selenium WebDriver and Watir, both of which are featured in the classic book 'Agile Testing[2]' by Lisa Crispin and Janet Gregory. However, the idea of free and open-source testing frameworks is still not appealing to many test managers. (Update: the previous statement was written in 2010, the situation has changed now as Selenium WebDriver is the dominant testing framework). Lack of skills, dedicated tools, and support are their main concerns.

2. Steep Learning Curves: Difficult to learn

Traditional commercial tools are usually focused on a Record and Playback approach with test scripts in a vendor-proprietary syntax. It looks easy when you watch the sales presentations. Unfortunately, it is a quite different story in real life (a programmer's minor

[2]http://www.agiletester.ca/

change to the application can ruin your hours of recording). When testers have to open the raw test scripts (generated by recorders) to edit, reality bites.

Open source test frameworks, on the other hand, require some degree of programming efforts, Selenium-WebDriver and Watir are among the popular ones. With programming, they provide the flexibility needed for automated testing. However, the fact is that the majority of software testers do not possess programming skills, and many of them feel uncomfortable to learn it. Besides, there are few dedicated testing tools supporting these open-source test frameworks designed to suite testers. (Programming IDEs are designed for programmers, not for testers who may find them complicated and overwhelming).

3. Hump 1: Hard to maintain

Software under development changes frequently, and automated UI test scripts are vulnerable to application changes. Even a simple change to the application could cause many existing test scripts to fail. This, in my view, is the most fundamental reason for test automation failures.

4. Hump 2: Long feedback loop

Compared to programmer tests (which if written well, should have an execution time under 0.1 second), automated functional tests through UI are relatively slow. There is practically very little that testers can do to speed up the execution of functional tests. With the number of test cases growing, so will be the test execution time. This leads to a long feedback gap, from the time programmers committed the code to the time test execution completes. If programmers continue developing new features or fixes during the gap time, it can easily get into a tail-chasing problem. This will hurt the team's productivity, not to mention the team's morale.

New Challenges for testing Web applications

Specifically to web applications, with the adoption of AJAX (Asynchronous JavaScript and XML) and increasing use of JavaScript, websites nowadays are more dynamic, therefore, bring new challenges to web test automation.

1.4 Successful web test automation

Having identified the reasons for test automation failures in projects, it becomes clear what it takes to succeed in web test automation:

1. Test scripts must be easy to read and maintain.
2. Test framework must be reliable and flexible.

3. Testing tool must be easy to learn, affordable and support team collaboration.
4. Test execution must be fast.

Is that all possible? My answer is ‘Yes’. The purpose of this book is to show how you can achieve these.

1.5 Learning approach

This is not just another book on testing theories, as there is no shortage of them. In this book, we will walk through examples using test framework Selenium WebDriver and functional testing IDE TestWise. The best way to learn is to start doing it.

My father is a well respected high school mathematics teacher in our town. His teaching style is “teaching by examples”. He gets students to work on his carefully selected math exercises followed by concise instruction, then guides students who face challenges. By working with many testers, I found this is the most effective way for testers to master automated testing quickly.

For most web applications, regardless of technologies they are developed on, Microsoft Windows is often the target platform (at least for now). It will be the main platform for our exercises in this book:

Web Browser:	Chrome, Internet Explorer and Firefox
Test Framework:	Selenium WebDriver
Testing Tool:	TestWise IDE

It is worth noting that the practices and test scripts work for macOS and Linux as well. If you are Mac user, like myself, the learning process is the same (majority of the test scripts run without change) except the screenshots shown in the book look different. All the techniques and test scripts are directly applicable for cross-browser testing.

On testing tools, I use TestWise, a functional testing IDE that supports Selenium WebDriver, Watir and Appium *(TestWise maybe used free)*, in this book. For readers who prefer their own favorite editors or IDEs, you can still use them, as all test scripts shown in this book are plain text. I will also provide instructions on how to execute tests from the command line.

Example test scripts for chapters in this book can be downloaded at the book site, and you can try them out by simply opening in TestWise and run. I have provided screencasts there as well, readers can watch how it is done.

In this book, we will focus on testing standard web sites (in HTML), excluding vendorspecific and deprecated technologies such as Flash and SilverLight. The techniques shown in this book are applicable to general testing practices.

1.6 Next action

Enough theory for now. Let's roll up sleeves and write some automated tests.

2. First Automated Test

A journey of a thousand miles must begins with a single step.

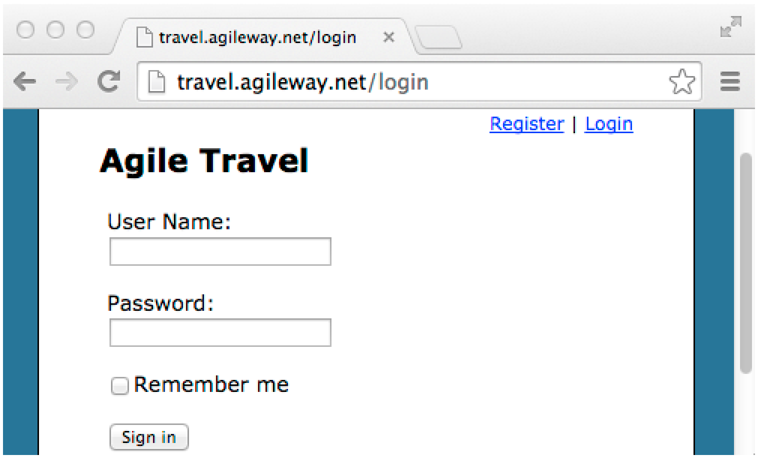
– Lao Tzu

Let's write an automated web test. If you are new to automated testing, don't feel intimidated. You are going to see your first automated test running in a Chrome browser in about 10 minutes, and that includes installing the test tool!

2.1 Test Design

A test starts with a requirement (called User Story in agile projects). Quite commonly, the first simple requirement to test is User Authentication. We will use this requirement for our first test in this exercise.

By analyzing the requirement and the application (see the screenshot below),



we can start to collect the test data:

Site URL: https://travel.agileway.net
User Login/Password: **agileway/testwise**

and design the test steps:

1. Enter username “agileway”
2. Enter password “testwise”
3. Click the “Sign in” button
4. Verify: “Welcome agileway” appears
5. Click the “Sign off” link

You might by now be saying “*there is no difference from manual testing*”. Your observation is correct. If you currently work as a manual tester, you probably feel relief at knowing your test design skills can apply to automated testing. As a matter of fact, we usually perform the test steps manually as verification of test design before writing automated test scripts.

Now we are going to automate it. The main purpose of this exercise is to help you write an automated Selenium WebDriver test case and watch it running in a browser, in a matter of minutes. Don’t pay attention to details yet, as it will become clear as we move on. If you get stuck, follow the screencast for this exercise at http://zhimin.com/books/pwta[1].

2.2 Installing TestWise (about 2 minutes)

In this exercise, we will use TestWise, a Functional Testing IDE *(created by me)* built for Selenium WebDriver. You can use TestWise free, no need to pay license fees.

Prerequisite

- A PC with MS Windows 10+ or macOS or Linux
- Chrome browser

Download

TestWise (two editions) can be downloaded at https://agileway.com.au/testwise/downloads[2]. If you are an absolute beginner to test automation, I strongly suggest starting with the TestWise Ruby edition first.

- **TestWise Ruby edition** *(Windows only)*

  TestWise Ruby edition (29MB in size) bundles Ruby, ChromeDriver and required libraries (called Gems) required.

[1]http://zhimin.com/books/pwta
[2]https://agileway.com.au/testwise/downloads

- **TestWise Standard edition** *(for macOS/Linux/Windows)*

  TestWise Standard editions are available for all three desktop platforms. Different from TestWise Ruby edition, the stanard edition does not provide the test execution environment (executing tests from the command line), which you can install yourself. It is quite straightforward, you can find the installation instructions in the next Chapter.

TestWise Ruby Edition = TestWise Standard Edition + Ruby + Testing Libraries + ChromeDriver.

Install

- **TestWise IDE**

  - Windows platform

    Double click *TestWise-x.x-ruby-setup.exe* to install, accept all default options. The default installation folder is *C:\agileway\TestWise6*. Launch TestWise after the installation completes.

    If you using TestWise Ruby edition on Windows, can skip the below in the 'Install' and 'Setup' sections below.

  - macOS

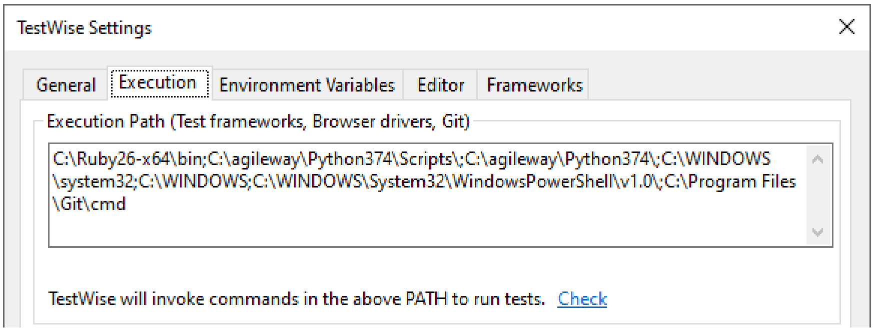
    Standard installation procedure: open the DMG file, drag the app to the 'Applications' folder.

  - Linux

    `tar -zxvf testwise-x.x.zip` then run `install.sh` there.

- **Set up Execution Path in TestWise**

  If you are using standard TestWise edition, after installation, make sure add your RUBY path and ChromeDriver path to the TestWise's settings (as below).


TestWise Settings

General | Execution | Environment Variables | Editor | Frameworks

Execution Path (Test frameworks, Browser drivers, Git)

C:\Ruby26-x64\bin;C:\agileway\Python374\Scripts\;C:\agileway\Python374\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Git\cmd

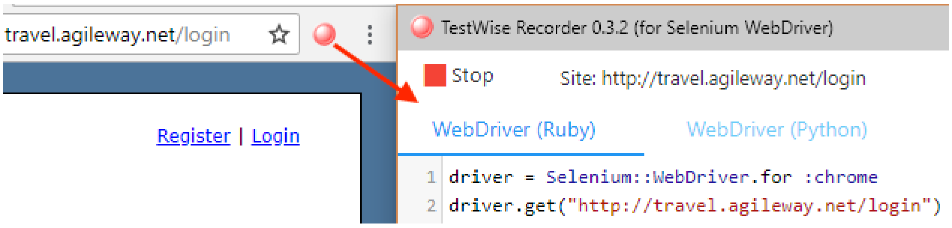
TestWise will invoke commands in the above PATH to run tests. Check

- **TestWise Recorder** or **Selenium IDE** *(optional)*

  TestWise Recorder is a Chrome extension (created by me), which records your operations into executable Selenium WebDriver and Watir test scripts while you navigate through your web application in Chrome. To install, open `chrome://extensions` in Chrome, then download the recorder[3] (a zip file) and drag it to the Chrome tab.

  > You will see a warning or "Errors" shown next to the extension. This is because TestWise Recorder was developed based on Chrome Extension manifest version 2 specification, which will be deprecated in 2023. For this reason, "TestWise Recorder" has been removed from the Chrome Web Store page[a].
  >
  > I had no plans to update TestWise Recorder, as I don't use recorders for test automation. Also, I don't recommend beginners use it either, most attendants to my one-day training expressed they like handcrafting the selenium tests.
  >
  > [a]https://chrome.google.com/webstore/detail/testwise-recorder/febfogamlejngokejcaimklgcfphjbok

  Click the recorder icon on the toolbar to enable recording.


travel.agileway.net/login
Register | Login
TestWise Recorder 0.3.2 (for Selenium WebDriver)
Stop
Site: http://travel.agileway.net/login
WebDriver (Ruby)
WebDriver (Python)

```
driver = Selenium::WebDriver.for :chrome
driver.get("http://travel.agileway.net/login")
```

[3]https://agileway.com.au/testwise/recorder

Selenium IDE is the official recorder created by the Selenium team. It has more features than TestWise Recorder. Please be aware of the name, it is not an IDE, but rather a record-n-playback tool. The new version (v4) is a lot better than its precessors in terms of accuracy. Still, I don't use it as hand-craft scripting creates better automated test steps, and it is quite easy too.

You may use Selenium IDE for this exercise, or better, not using a recorder at all.

Unlike other most Test Automation Tools, TestWise does not come with a recorder becuase it discourage recording (but not excluding it either). In TestWise, you can create a test script file normally, have the freedom to incorporate recorded test steps into your script as necessary.

2.3 Create Automated Test

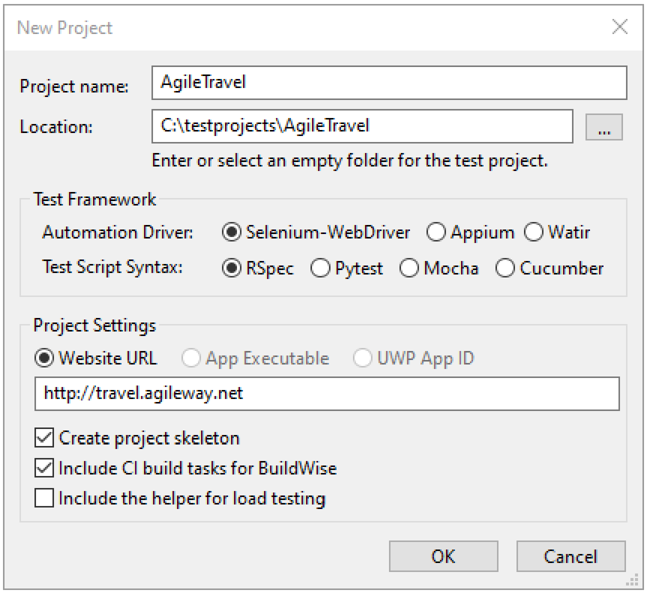
Now we are ready to create the test for our requirement: *"User can log in the site"*. Hope you still remember the test design steps and test data.

Create a new test project

TestWise has a project structure to organize test scripts. This structure is simply a folder containing all test related files such as test scripts and test data.

As we start from scratch, we need to create a new project first. If a sample project is already opened in TestWise, we need to close it.

Select menu **File** → **New Project**, which will bring up the window shown below.


New Project

Project name: AgileTravel

Location: C:\testprojects\AgileTravel

Enter or select an empty folder for the test project.

Test Framework

Automation Driver: [✓ Selenium-WebDriver] Appium Watir

Test Script Syntax: [✓ RSpec] Pytest Mocha Cucumber

Project Settings

[✓ Website URL] App Executable UWP App ID

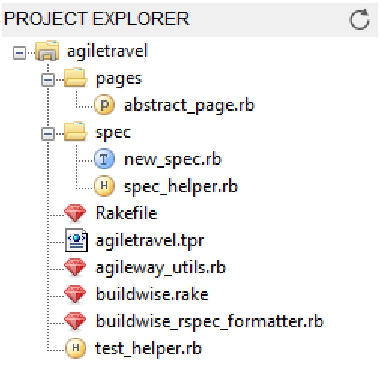
http://travel.agileway.net

[✓ Create project skeleton]

[✓ Include CI build tasks for BuildWise]

Include the helper for load testing

Enter a project name, project folder *(a newly created one)* and the URL of the web site to be tested. In this case, we enter "*Agile Travel*", "*C:\testprojects\AgileTravel*" and "*http://travel.agileway.net*" respectively, then click 'OK' button. TestWise will create the project with skeleton files.

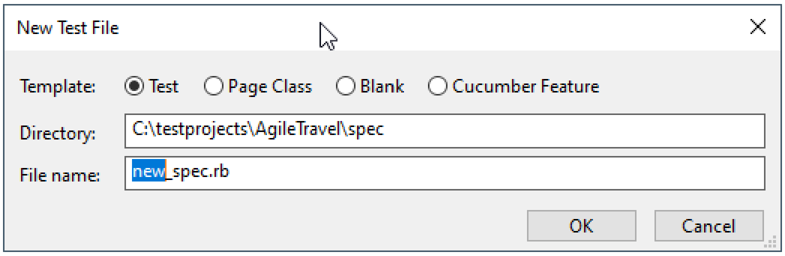

PROJECT EXPLORER

```
agiletravel
  pages
    abstract_page.rb
  spec
    new_spec.rb
    spec_helper.rb
  Rakefile
  agiletravel.tpr
  agileway_utils.rb
  buildwise.rake
  buildwise_rspec_formatter.rb
  test_helper.rb
```

Please note, we just enter the host name of URL, not `https://travel.agileway.net/login`, *rather* `https://travel.agileway.net`.

Create a test script file

Now create the test script file for our test. Select '**File**' → '**New File**',


New Test File

Template: [✓ Test] Page Class Blank Cucumber Feature

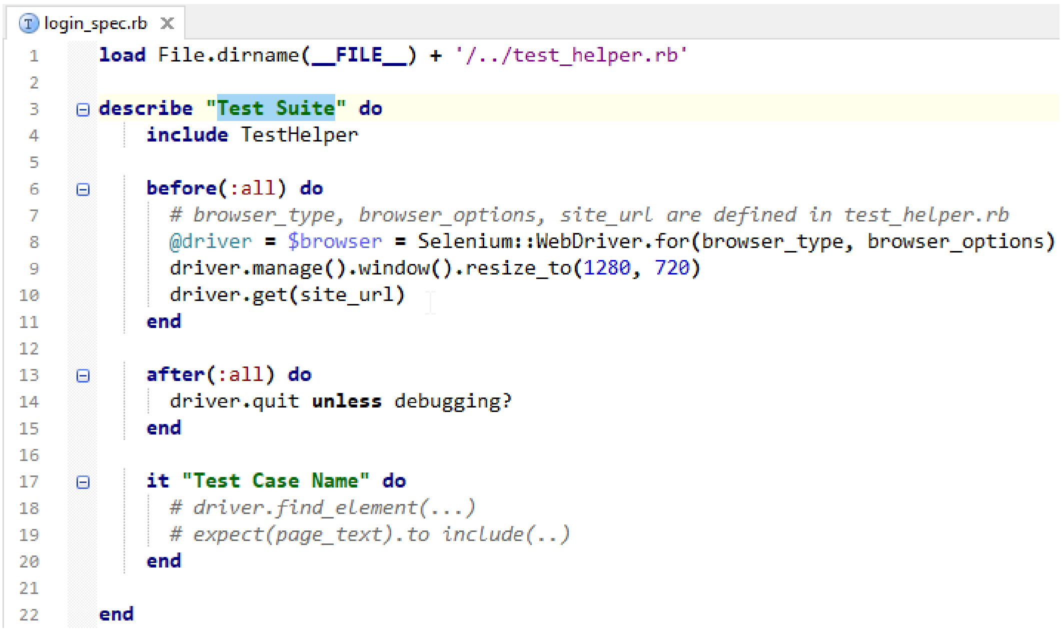
Directory: C:\testprojects\AgileTravel\spec

File name: new_spec.rb

Type text 'login' and press Enter to create new test script file: *login_spec.rb*

Try naming the test script file something related to the requirement, so you can find it easily later.

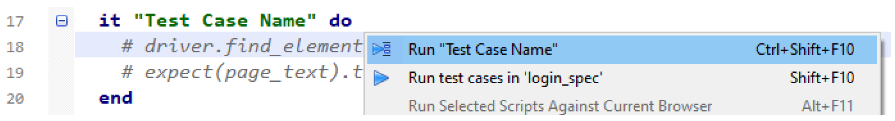
A new editor tab is created and opened with a test skeleton:



```
 1  load File.dirname(__FILE__) + '/../test_helper.rb'
 2
 3  describe "Test Suite" do
 4    include TestHelper
 5
 6    before(:all) do
 7      # browser_type, browser_options, site_url are defined in test_helper.rb
 8      @driver = $browser = Selenium::WebDriver.for(browser_type, browser_options)
 9      driver.manage().window().resize_to(1280, 720)
10      driver.get(site_url)
11    end
12
13    after(:all) do
14      driver.quit unless debugging?
15    end
16
17    it "Test Case Name" do
18      # driver.find_element(...)
19      # expect(page_text).to include(..)
20    end
21
22  end
```

Run the empty test case

Currentlly, except comments (lines started with #), our test case (between `it` to `end`) is empty. Right click a line within the test case (line 17 to 21), then select 'Run "Test Case Name"'.



```
it "Test Case Name" do
  # driver.find_element
  # expect(page_text).t
end
```

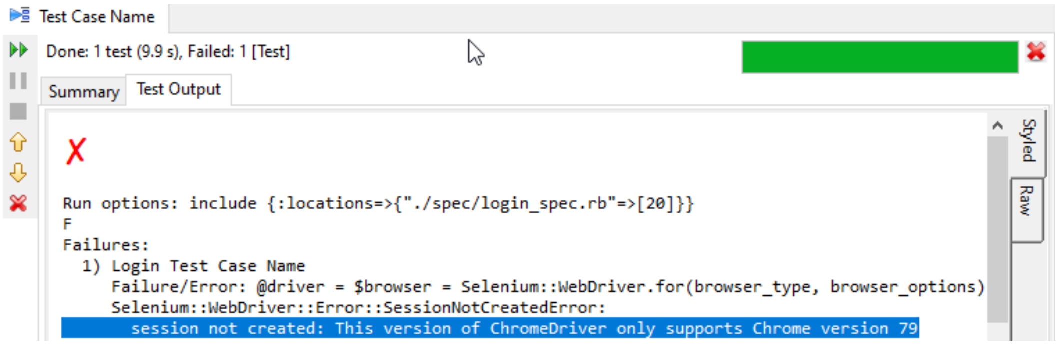
Run "Test Case Name" Ctrl+Shift+F10
Run test cases in 'login_spec' Shift+F10
Run Selected Scripts Against Current Browser Alt+F11

You shall see a new Chrome browser launching and open your target web site. This is to make sure your test execution environment is correct.

If no Chrome browser launched ...

The most likely reason is that the ChromeDriver does not match the Chrome browser on your machine. Chrome browser, by default, self-updates every two months or so. As a result, the chromedriver needs to be updated accordingly.

To find out the exact reason, click '**Test output**' tab in TestWise as below:



```
Run options: include {:locations=>{"./spec/login_spec.rb"=>[20]}}
F
Failures:
  1) Login Test Case Name
     Failure/Error: @driver = $browser = Selenium::WebDriver.for(browser_type, browser_options)
     Selenium::WebDriver::Error::SessionNotCreatedError:
       session not created: This version of ChromeDriver only supports Chrome version 79
```

The text "*session not created: This version of ChromeDriver only supports Chrome version 79*" suggests the ChromeDriver v79 (under TestWise Ruby Edition) does support the newer Chrome (v81 in this case) on the machine.

The workaround is simple:

1. Find out your Chrome version.
2. Download the ChromeDriver[4] matches the browser.
3. Put `chromedriver` executable in a PATH in TestWise's settings ('Execution' Tab → 'Execution Path').

   for TestWise Ruby edition, simply copy `chromedriver.exe` *to* `C:\agileway\TestWise6\interpreters\ruby\bin`

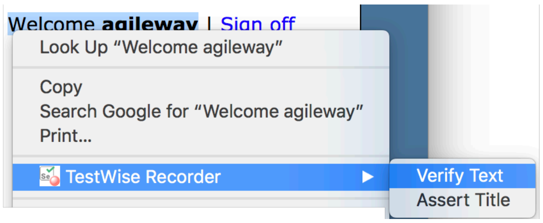
[4]https://chromedriver.chromium.org/downloads

Recording Test Steps (optional)

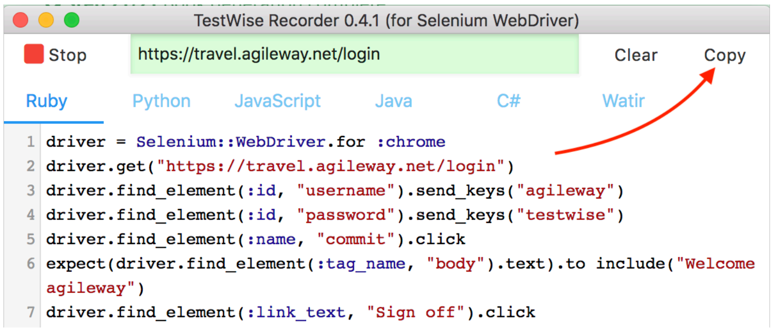
I Don't recommend this way

Open the site URL *https://travel.agileway.net* in Chrome and enable 'TestWise Recorder' extension. Perform the test steps below manually:

1. Enter username '*agileway*'
2. Enter password '*testwise*'
3. Click the '*Sign in*' button
4. To add verification for text '*Welcome agileway*', highlight the text in the browser, right-click and select "Verify Text" in the context menu.
5. Click the "Sign off" link


Welcome agileway | Sign off
Look Up "Welcome agileway"
Copy
Search Google for "Welcome agileway"
Print...
TestWise Recorder
Verify Text
Assert Title

Test steps are recorded along the way. Once done, inside the TestWise Recorder window, click the 'Copy' button on top to copy recorded test steps to clipboard.


TestWise Recorder 0.4.1 (for Selenium WebDriver)
Stop
https://travel.agileway.net/login
Clear
Copy
Ruby
Python
JavaScript
Java
C#
Watir

```ruby
driver = Selenium::WebDriver.for :chrome
driver.get("https://travel.agileway.net/login")
driver.find_element(:id, "username").send_keys("agileway")
driver.find_element(:id, "password").send_keys("testwise")
driver.find_element(:name, "commit").click
expect(driver.find_element(:tag_name, "body").text).to include("Welcome
agileway")
driver.find_element(:link_text, "Sign off").click
```

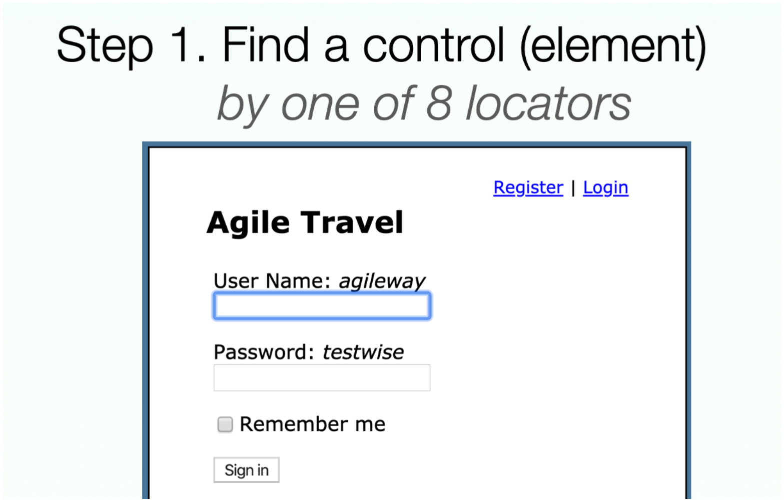
2.4 Selenium Syntax in minutes

Putting aside what you might have heard about Selenium previously, I am telling you that Selenium WebDriver is the easiest-to-learn test automation framework. In doubt? You will find out yourself in minutes.

A web page contains many page elements, such as links, text fields and buttons. Selenium syntax follows a **simple and intuitive pattern**:

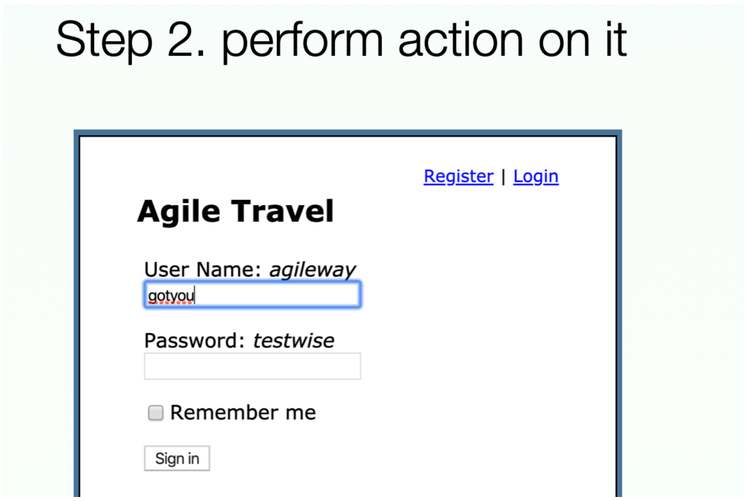
locate a web control and drive it

Step 1: Find a web control (also known as element)



For example, `driver.find_element(:id, "username")` for the "User Name" text box.

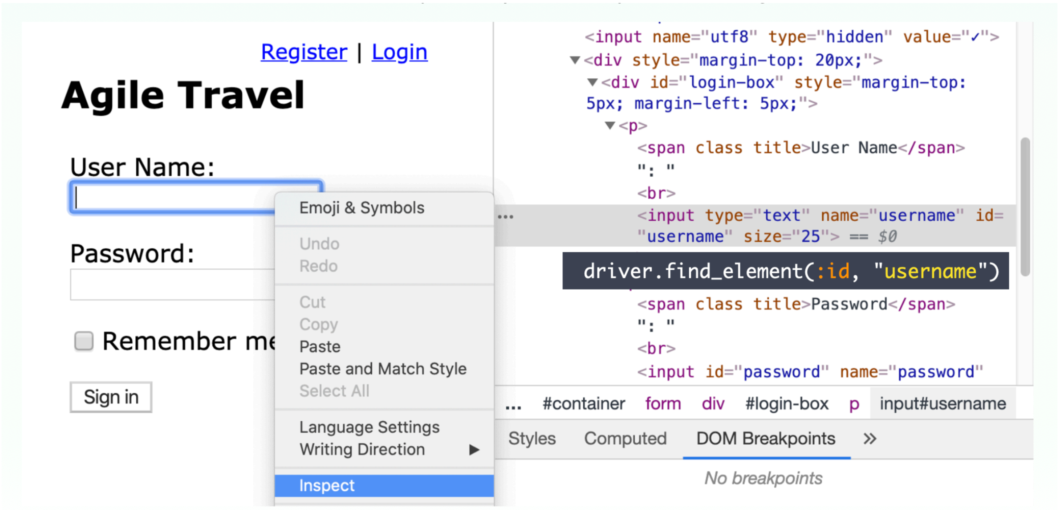
Step 2: Perform an action on it



Typing text is an action we can perform on the text box, in Selenium, it is `send_keys`. This complete step `driver.find_element(:id, "username").send_keys("agileway")` enters a user name.

Some might wonder, how about a link such as 'Register' link on this page? The selenium step will be `driver.find_element(:link, "Register").click`.

Astute readers will realize that Step 2 is easy, the effort is on Step 1: how to find a web element? It is not hard either, and you don't need other tools. Just simply right-click the target element in a web page and select 'Inspect'.



```
<input name="utf8" type="hidden" value="✓">
▼<div style="margin-top: 20px;">
  ▼<div id="login-box" style="margin-top:
  5px; margin-left: 5px;">
    ▼<p>
       <span class title>User Name</span>
       ": "
       <br>
       <input type="text" name="username" id=
       "username" size="25"> == $0
  driver.find_element(:id, "username")
       <span class title>Password</span>
       ": "
       <br>
       <input id="password" name="password"
```

HTML is behind a web page. We can see the HTML fragment with this element highlighted. Then choose one of Selenium's nine locators.

Locator	Example
ID	`find_element(:id, "user")`
Name	`find_element(:name, "username")`
Link Text	`find_element(:link_text, "Login")`
Partial Link Text	`find_element(:partial_link_text, "Next")`
XPath	`find_element(:xpath, "//div[@id="login"]/input")`
Tag Name	`find_element(:tag_name, "body")`
Class Name	`find_element(:class_name, "table")`
CSS	`find_element(:css, "#login > input[type="text"]")`
Relative (v4)	`find_element(relative: { tag_name: "img", right: elem })`

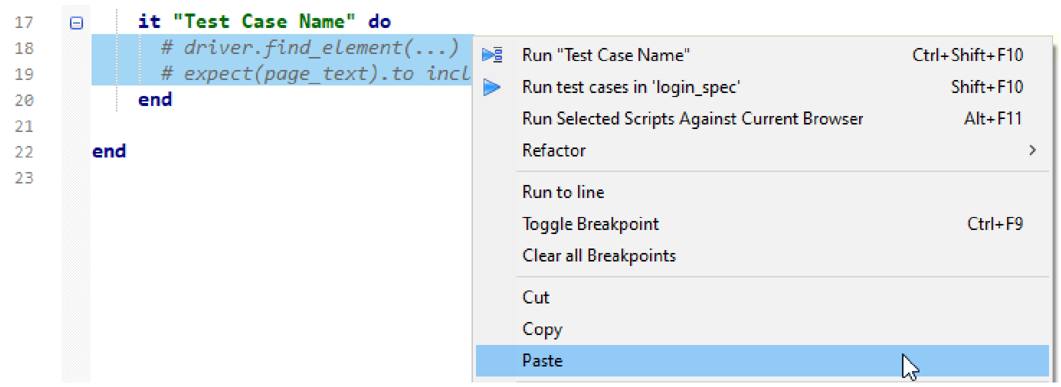
The easy choice, obviously, is to use `ID`. You may use any locator for a page element. Beginners might feel a little overwhelmed, but don't worry. Your start with ID, and learning to use other locators gradually.

2.5 Create a test case

Now you should have some test steps created by a recorder or manual scripting.

Recorder-generated

Switch to the TestWise IDE (the *login_spec.rb* editor tab shall be still active), paste recorded test scripts into the test case.



```
  it "Test Case Name" do
    # driver.find_element(...)
    # expect(page_text).to incl
  end

end
```

Run "Test Case Name" Ctrl+Shift+F10
Run test cases in 'login_spec' Shift+F10
Run Selected Scripts Against Current Browser Alt+F11
Refactor
Run to line
Toggle Breakpoint Ctrl+F9
Clear all Breakpoints
Cut
Copy
Paste

The test case is created.

While we are here, update the test suite name (the string in `describe "..."`) to "*User Authentication*" and the test case's name (the string in `it "..." do`) to "*User can log in with valid user name and password*".

The first two copied steps:

```
driver = Selenium::WebDriver.for :chrome;
driver.get("http://travel.agileway.net/login")
```

start Chrome browser and navigate to our target server. You can delete them if using TestWise, as they are already included in `before(:all)` block (created by TestWise in a more generic format).

Handcraft Scripting

There are a few test steps according to our login test design. Let's convert them to selenium steps one by one.

1. **Enter username "agileway"**.

   I will use the `ID` locator, an easy choice of the element's ID is present.

   ```
   driver.find_element(:id, "username").send_keys("agileway")
   ```

2. **Enter password "testwise"**.

   ```
   driver.find_element(:id, "password").send_keys("testwise")
   ```

3. **Click the button "Sign in"**.

   There is no `ID` for the 'Sign in' button, I use the `NAME` locator instead.

   ```
   driver.find_element(:name, "commit").click
   ```

   The above works, but not a good option (the `commit` name is not meaningful). For readers who are familiar with Xpath, the below one is better.

   ```
   driver.find_element(:xpath, "//input[@value='Sign in']").click
   ```

4. **Verify**: **"Welcome agileway" appears**.

   ```
   expect(driver.find_element(:tag_name, "body").text).to include("Welcome
   agileway")
   ```

   Here I used the `TAG_NAME` locator to get a web page body text for assertions. Don't worry about `expect` syntax, which I will cover later. In the meantime, just to get a feel of checking in automated test scripts.

5. **Click the "Sign off" link**.

   I used the `LINK_TEXT` locator here.

   ```
   driver.find_element(:link, "Sign off").click
   ```

Now, review the selenium test steps against your design. It makes sense, isn't it? One test script step for each user operation.

In the above five test steps, we used five of Selenium's eight locators (the new relative locator, introduced in v4, is rarely used anyway). As a matter of fact, I used only these five locators for over 95% of the selenium test scripts I wrote. See, Selenium is quite easy to learn, right?

Beginners commonly mistyped some test steps, that's normal. There are solutions, such as using Snippets in TestWise (Chapter 4). In the meantime, try to type the selenium steps correctly. If frustrated, copy-n-paste from the book source is fine too.

You may run the test case any time during writing a new test script, in fact, I recommend so. TestWise has several execution modes, the one I showed earlier is called: "**Individual Test Execution**". In this mode, the browser will be kept open after the test case execution completes. This is a very helpful feature so that we can continue inspect the page in the browser.

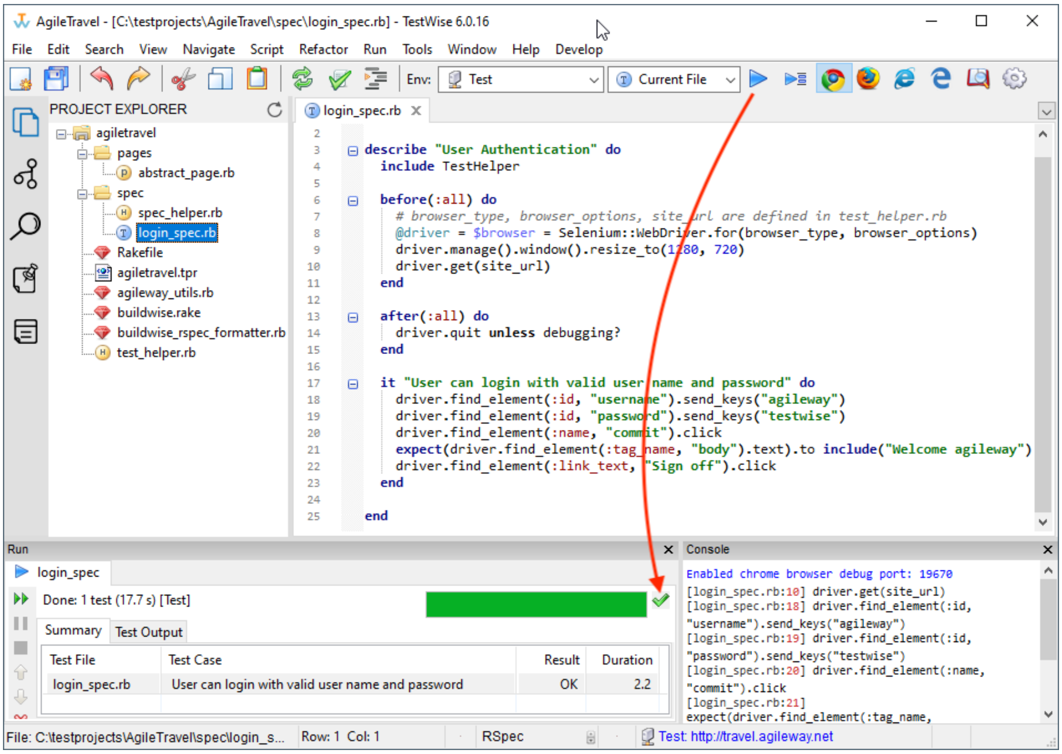
Full Test Script

The test scripts in the TestWise shall be like the below:

```
load File.dirname(__FILE__) + '/../test_helper.rb'

describe "User Authentication" do
  include TestHelper

  before(:all) do
    # browser_type, browser_options, site_url are defined in test_helper.rb
    @driver = $driver = Selenium::WebDriver.for(browser_type, browser_options)
    driver.manage().window().resize_to(1280, 720)
    driver.get(site_url)
  end

  after(:all) do
    @driver.quit unless debugging?
  end

  it "User can login with valid user name and password" do
    driver.find_element(:id, "username").send_keys("agileway")
    driver.find_element(:id, "password").send_keys("testwise")
    driver.find_element(:name, "commit").click
    expect(driver.find_element(:tag_name, "body").text).to include("Welcome agileway")
    driver.find_element(:link, "Sign off").click
  end
end
```

2.6 Run the full test in a Chrome browser

Select the Chrome browser icon (default) and press ▶ on the toolbar (highlighted in the screenshot below) to run the test case, and you can watch the test execution in a Chrome window.


AgileTravel - [C:\testprojects\AgileTravel\spec\login_spec.rb] - TestWise 6.0.16
File Edit Search View Navigate Script Refactor Run Tools Window Help Develop
Env: Test
Current File
PROJECT EXPLORER
agiletravel
pages
abstract_page.rb
spec
spec_helper.rb
login_spec.rb
Rakefile
agiletravel.tpr
agileway_utils.rb
buildwise.rake
buildwise_rspec_formatter.rb
test_helper.rb
login_spec.rb
Run
login_spec
Summary
Test Output
Console
File: C:\testprojects\AgileTravel\spec\login_s...
Row: 1 Col: 1
RSpec
Test: http://travel.agileway.net

```
describe "User Authentication" do
  include TestHelper

  before(:all) do
    # browser_type, browser_options, site_url are defined in test_helper.rb
    @driver = $browser = Selenium::WebDriver.for(browser_type, browser_options)
    driver.manage().window().resize_to(1280, 720)
    driver.get(site_url)
  end

  after(:all) do
    driver.quit unless debugging?
  end

  it "User can login with valid user name and password" do
    driver.find_element(:id, "username").send_keys("agileway")
    driver.find_element(:id, "password").send_keys("testwise")
    driver.find_element(:name, "commit").click
    expect(driver.find_element(:tag_name, "body").text).to include("Welcome agileway")
    driver.find_element(:link_text, "Sign off").click
  end

end
```

Done: 1 test (17.7 s) [Test]

Test File	Test Case	Result	Duration
login_spec.rb	User can login with valid user name and password	OK	2.2

```
Enabled chrome browser debug port: 19670
[login_spec.rb:10] driver.get(site_url)
[login_spec.rb:18] driver.find_element(:id,
"username").send_keys("agileway")
[login_spec.rb:19] driver.find_element(:id,
"password").send_keys("testwise")
[login_spec.rb:20] driver.find_element(:name,
"commit").click
[login_spec.rb:21]
expect(driver.find_element(:tag_name,
```

The green tick means the test passed.

You might notice something different from previous test executions: the browser was closed after the test execution. Yes, this is another TestWise execution mode: "**Test Script Execution**", which executes the test script file (including one or many test cases in it), the browser will close after the test execution completes.

WebDriver is a W3C standard. All major browsers vendors such as Google, Apple, Microsoft and Mozilla support WebDriver. For example, you just need to install GeckoDriver[5] for Firefox.

Set browser and target URL specifically

Some readers might ask *"I don't see the target server URL and Chrome browser being set in the test script"*.

```
@driver = Selenium::WebDriver.for(browser_type)
@driver.get.to(site_url)
```

The `browser_type` and `site_url` are defined in `test_helper.rb`, which you can easily modify. More importantly, with IDE support, you can run tests against another target server (in *Project settings*) and browser quickly in IDE. Feel free to change the target browser to Edge or Safari (provided that the browser and its driver are installed correctly) and run the test again.

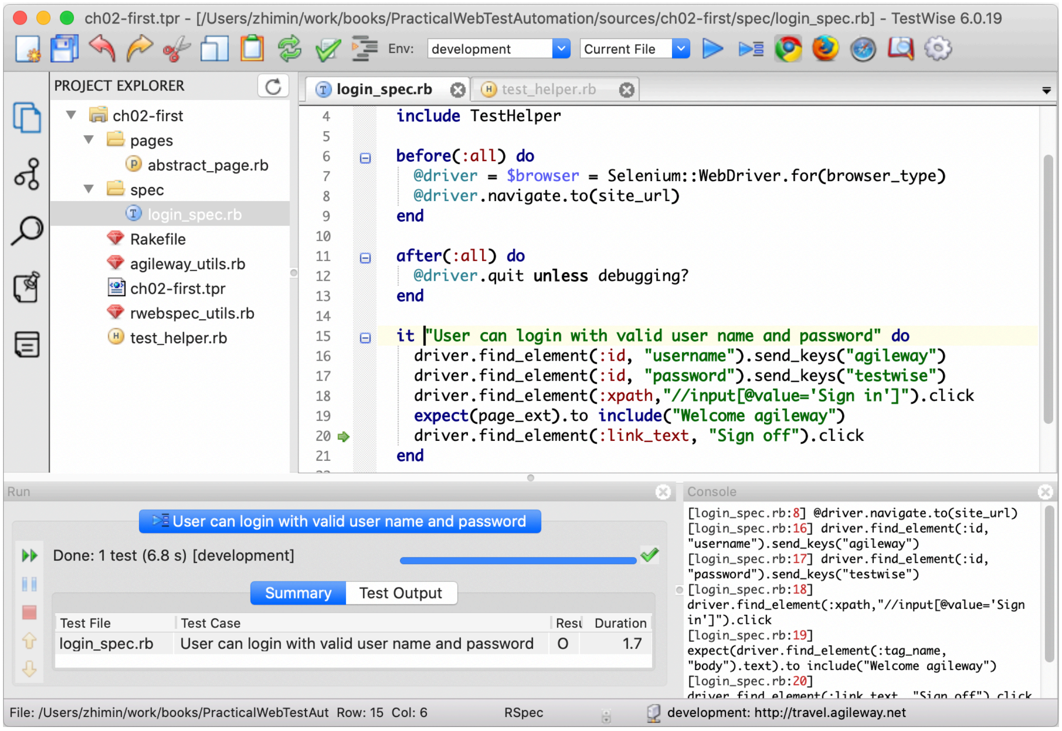
If you want to set the browser type and server URL specifically in each individual test script, you can.

```
@driver = Selenium::WebDriver.for(:edge)
```

2.7 Running on macOS

The test script works on all desktop platforms. Here is what it looks like when this test is run in TestWise on macOS.

[5] https://github.com/mozilla/geckodriver/releases


ch02-first.tpr - [/Users/zhimin/work/books/PracticalWebTestAutomation/sources/ch02-first/spec/login_spec.rb] - TestWise 6.0.19

Env: development | Current File

PROJECT EXPLORER
- ch02-first
  - pages
    - abstract_page.rb
  - spec
    - login_spec.rb
  - Rakefile
  - agileway_utils.rb
  - ch02-first.tpr
  - rwebspec_utils.rb
  - test_helper.rb

login_spec.rb | test_helper.rb

```ruby
include TestHelper

before(:all) do
  @driver = $browser = Selenium::WebDriver.for(browser_type)
  @driver.navigate.to(site_url)
end

after(:all) do
  @driver.quit unless debugging?
end

it "User can login with valid user name and password" do
  driver.find_element(:id, "username").send_keys("agileway")
  driver.find_element(:id, "password").send_keys("testwise")
  driver.find_element(:xpath,"//input[@value='Sign in']").click
  expect(page_ext).to include("Welcome agileway")
  driver.find_element(:link_text, "Sign off").click
end
```

Run

User can login with valid user name and password

Done: 1 test (6.8 s) [development]

Summary | Test Output

Test File	Test Case	Resu	Duration
login_spec.rb	User can login with valid user name and password	O	1.7

Console

```
[login_spec.rb:8] @driver.navigate.to(site_url)
[login_spec.rb:16] driver.find_element(:id,
"username").send_keys("agileway")
[login_spec.rb:17] driver.find_element(:id,
"password").send_keys("testwise")
[login_spec.rb:18]
driver.find_element(:xpath,"//input[@value='Sign
in']").click
[login_spec.rb:19]
expect(driver.find_element(:tag_name,
"body").text).to include("Welcome agileway")
[login_spec.rb:20]
driver.find_element(:link_text, "Sign off").click
```

File: /Users/zhimin/work/books/PracticalWebTestAut Row: 15 Col: 6 RSpec development: http://travel.agileway.net

2.8 When a test failed…

We just saw a successful automated test execution. Naturally, you will ask what will happen when a test fails? As a matter of fact, during the development of an automated test script, we are more likely to get errors or failures before we get it right. It is up to the technical testers to analyze the cause: is it a real application error or incorrect test scripts?

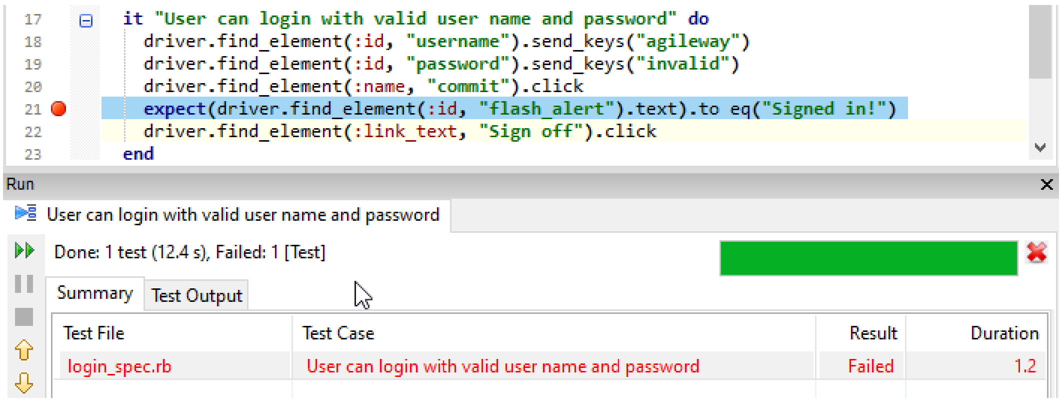
Next, we will make a simple change to the above test script to make it fail:

```ruby
driver.find_element(:name, "password").send_keys("invalid")  # will fail
```

And I want to refine the assertion: checking the specific alert message, rather than verify a piece of text appearing (anywhere) on the page.

```ruby
expect(driver.find_element(:id, "flash_alert").text).to eq("Signed in!")
```

Click ▶ to run the test. As expected, the test failed.



```
it "User can login with valid user name and password" do
  driver.find_element(:id, "username").send_keys("agileway")
  driver.find_element(:id, "password").send_keys("invalid")
  driver.find_element(:name, "commit").click
  expect(driver.find_element(:id, "flash_alert").text).to eq("Signed in!")
  driver.find_element(:link_text, "Sign off").click
end
```

Run

User can login with valid user name and password

Done: 1 test (12.4 s), Failed: 1 [Test]

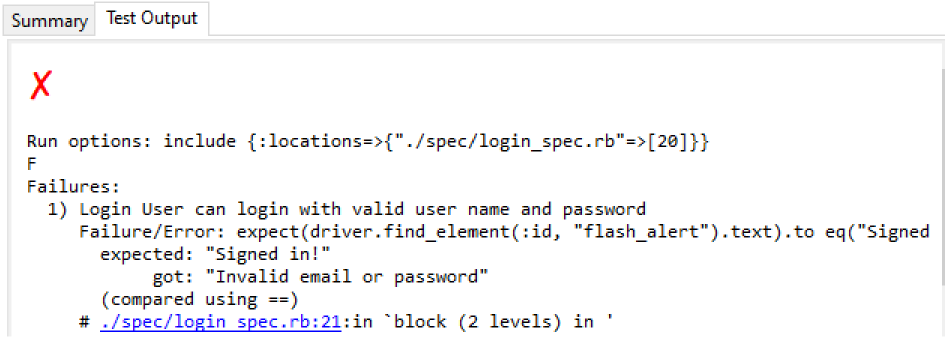
Summary | Test Output

Test File	Test Case	Result	Duration
login_spec.rb	User can login with valid user name and password	Failed	1.2

In TestWise, the test execution is marked as "Failed" and ● is shown on line 21 of the test script indicating where the failure is.

We, as humans, knew the reason for this failure: a wrong password was provided. From the test script's "point of view", it failed due to this assertion was not met: the alert message on the page is not "Signed in!".

If you want to find more details about the cause for test failure, check the text output of test execution including error trace under "**Test Output**" tab.



Summary | Test Output

```
X

Run options: include {:locations=>{"./spec/login_spec.rb"=>[20]}}
F
Failures:
  1) Login User can login with valid user name and password
     Failure/Error: expect(driver.find_element(:id, "flash_alert").text).to eq("Signed
       expected: "Signed in!"
            got: "Invalid email or password"
       (compared using ==)
     # ./spec/login_spec.rb:21:in `block (2 levels) in '
```

In this case, the cause of failure is quite obvious by examining the output.

```
expected: "Signed in!"
     got: "Invalid email or password"`
```

2.9 Wrap up

Let’s review what we have done in this chapter. Besides the test design, we

- Installed TestWise IDE
- Installed TestWise Recorder (optional)
- Created a test project in TestWise IDE
- Recorded test scripts using TestWise Recorder in a Chrome browser (optional)
- Created test script from pasted test steps from the recorder, or
- Script the selenium test steps manually
- Ran test case in a browser (pass and failed)

Hopefully, you were able to do all that within 10 minutes! You can view the screencast for this exercise online at the book’s website at http://zhimin.com/books/pwta[6].

[6]http://zhimin.com/books/pwta

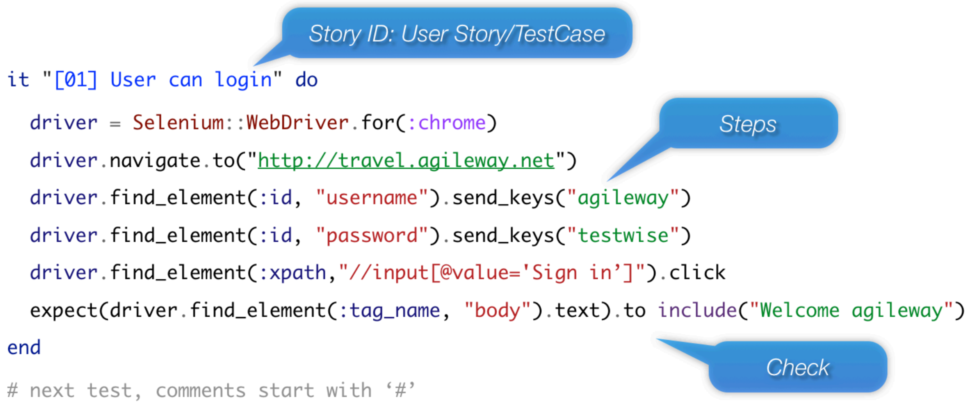
3. How Automated Testing works

In the previous chapter, we created an automated functional test running in a web browser, Chrome. This was done by simulating a user interacting with the browser: typing texts and clicking buttons.

Before we move on, let us examine our test targets - web applications (or websites). Simply speaking, a web site consists of many web pages. Behind each web page, there is an HTML (HyperText Markup Language) file. Browsers download the HTML files and render them.

HTML defines a set of standard web controls (aka elements) we are familiar with, such as text boxes, hyperlinks, buttons, checkboxes, etc. For web application testing, we interact with these controls as well as the texts that get marked up in the HTML such as labels and headings.

Now let us review the test script we created in the last exercise:



```
it "[01] User can login" do
  driver = Selenium::WebDriver.for(:chrome)
  driver.navigate.to("http://travel.agileway.net")
  driver.find_element(:id, "username").send_keys("agileway")
  driver.find_element(:id, "password").send_keys("testwise")
  driver.find_element(:xpath,"//input[@value='Sign in’]").click
  expect(driver.find_element(:tag_name, "body").text).to include("Welcome agileway")
end
# next test, comments start with ‘#’
```

Within a test case, test steps can be classified into the following two categories:

- **Operation** (also called step)

  Performing some kind of keyboard or mouse action on a web page. The above example test has three operations:

```
driver.find_element(:id, "username").send_keys("agileway")
driver.find_element(:id, "password").send_keys("testwise")
driver.find_element(:xpath, "//input[@value='Sign in']").click
```

- **Check** (also called assertion)

  Verifying the web page meets the requirement.

  ```
  the_page_text = driver.find_element(:tag_name, "body").text
  expect(the_page_text).to include("Welcome agileway")
  ```

3.1 Web test drivers

Web test drivers enable web controls to be driven by test scripts with a certain syntax, for testing purposes. All web test drivers covered in this book are free and open-source.

Selenium WebDriver

Selenium was originally created in 2004 by Jason Huggins, who was later joined by his other ThoughtWorks colleagues. Selenium supports all major browsers and tests can be written in many programming languages and run on Windows, Linux and Macintosh platforms.

Selenium WebDriver History

Selenium 2 is merged with another test framework WebDriver led by Simon Stewart at Google (that's why it is called 'selenium-webdriver').

- Selenium 2, released in July 2011
- Selenium 3, released in October 2016
- Selenium 4, released in October 2021

As of August 2022, the current stable version is Selenium 4.4.

Because WebDriver is a W3C standard (the official term is 'recommendation[1]'), just like HTML and XML, your test scripts most likely will work with future Selenium WebDriver versions.

Here is an example test in Selenium WebDriver:

[1]https://sfconservancy.org/news/2018/may/31/seleniumW3C/

```
require "selenium-webdriver"
driver = Selenium::WebDriver.for(:chrome)  # or :ie, :firefox, :safari
driver.navigate.to "http://www.google.com"
driver.find_element(:name, "q").send_keys "WebDriver IDE"
driver.find_element(:name, "btnG").click  #"btnG" is the 'Search' button
```

3.2 Automated testing rhythm

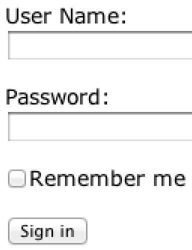
Regardless of which test framework you use, the '**testing rhythm**' is the same:

1. Identify a web control
2. Perform an operation on the control
3. Go to step 1 until reach a checkpoint
4. Check
5. Go to step 1 until the test ends

Identify web controls

To drive controls on a web page, we need to identify them first.

Let's look at this sample web page:



Its HTML source (you can view the HTML source of a web page by right clicking in the web page and selecting "View Page Source"):

```
User name: <input type="text" name="username" size="20"/>
Password: <input type="password" id="pwd_box" name="password" size="20"/>
<input type="submit" name="commit" value="Sign in"/>
```

Though the username and password appear the same (text box) on the browser, they are quite different in the source. Some attributes in HTML tags tell web browsers how to render it, such as `size="20"` in the user name text box. More importantly, application developers use attributes such as `"name"` (not exclusively) to determine the user's input associated with which control.

We can identify web controls by using these attributes for testing. Here is one way to identify them in Selenium:

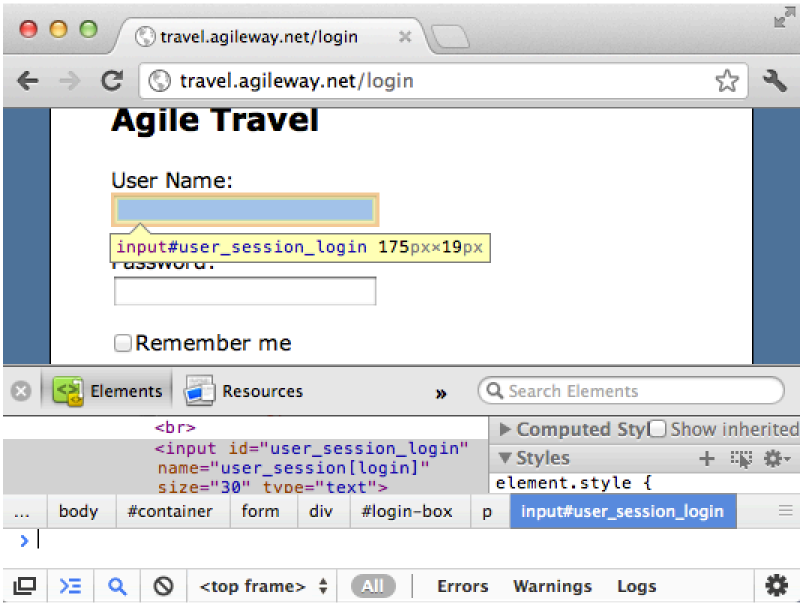
```
driver.find_element(:name, "username")
driver.find_element(:id, "pwd_box")
driver.find_element(:xpath, "//input[@value='Sign in']")
```

As you can see, these three test steps use three different attributes for three controls.

Obviously the easiest way to identify web controls is to use a recorder (a tool records user's operation and generate test scripts), if you have one installed. However, in my opinion, it is essential for technical testers to master and be comfortable to do it manually. The reasons are:

- Some test frameworks don't have recorders or have outdated ones
- Recorders might not work for certain circumstances
- Lack of freedom on choosing preferred attribute (for identifying controls)

In modern browsers, it is actually quite easy to identify element attributes (in HTML source) manually: just right-click Right-click on any page element and select Inspect Element.



```
<br>
<input id="user_session_login"
 name="user_session[login]"
 size="30" type="text">
element.style {
```

Drive web controls

Once we have identified a web control, the next step is to perform a required operation with it, such as typing text in a text field, clicking for a button, clearing a checkbox, and so on. Though different test frameworks have different syntax, the idea is the same.

Here are some examples:

```
driver.find_element(:name, "user[name]").send_keys "bob"
driver.find_element(:id, "next_btn").click
```

Check

The purpose of testing is to verify that a piece of function serves its purpose. After 'driving' the application to a certain point, we do checks (maybe that's why it is called 'checkpoint' in some testing tools).

In the context of web testing, typical checks are:

- verify certain texts are present
- verify certain HTML fragment are present (different from the above, this is to check raw page source)
- verify page title
- verify a link is present
- verify a web control is present or hidden

One key feature of Test frameworks (more in the next section) is to provide syntax conventions to perform verifications as above. Here are some examples:

- xUnit (assertion style)

  ```
  assert browser.html.include?("Payment Successful!")
  assert browser.button(:text, "Choose Selenium").enabled?
  assert browser.title == "User Registration"
  ```

- RSpec

  ```
  expect(driver.page_source).to include("Payment Successful!")
  expect(browser.find_element(:link_text, "Continue").displayed?).to be_truthy
  expect(driver.title).to eq("User Registration")
  ```

3.3 Test frameworks

Web test drivers, such as Selenium WebDriver, drive browsers. However, to make effective use of them for testing, we need to put them in a test framework that defines test structures and provides assertions (performing checks in test scripts).

xUnit

xUnit (JUnit and its cousins) test frameworks are widely used for unit testing by programmers. xUnit can be used in functional test scripts too, but it is not my preference, as it is not as expressive as the ones below.

RSpec

RSpec is a popular Behaviour Driven Development (BDD) framework in Ruby.

More expressive

Comparing to xUnit test frameworks, RSpec tests are easier to read. For example, for the JUnit test below:

```
class UserAuthenticationTest {
  public void testCanLoginWithValidUsernameAndPassword {
    // ...
  }
  public void testAccessDeniedForInvalidPassword() {
    // ...
  }
}
```

Its RSpec version will be like this:

```
describe "User Authentication" do
  it "User can login with valid login and password" do
    #   ...
  end

  it "Access denied for invalid password" do
    #...
  end
end
```

Execution Hooks

Execution hooks are similar to `setUp()` and `tearDown()` functions in JUnit. Test steps inside a execution hook are run before or after test cases depending on the nature of the hook. The example below shows the order of execution in RSpec:

```
describe "Execution Order Demo" do

  before(:all) do
    puts "Calling before(:all)"
  end

  before(:each) do
    puts "  Calling before(:each)"
  end

  after(:each) do
    puts "  Calling after(:each)"
  end

  after(:all) do
    puts "Calling after(:all)"
  end

  it "First Test Case" do
    puts "    In First Test Case"
  end

  it "Second Test Case" do
    puts "    In Second Test Case"
  end

end
```

Output

```
Calling before(:all)
  Calling before(:each)
    In First Test Case
  Calling after(:each)
  Calling before(:each)
    In Second Test Case
  Calling after(:each)
Calling after(:all)
```

What is the use of execution hooks? Let's look at the test script below (*the test script is in RWebSpec, an extension of Selenium WebDriver. please examine the structure of test scripts rather than test statement syntax, for now*). There are three login related test cases in a single test script file.

```
describe "User Login" do
  include TestHelper  # defined functions such as open_browser, login_as

  it "Can login as Registered User" do
    open_browser
    login_as("james", "pass")
    expect(page_text).to include("Welcome James")
    logout
    close_browser
  end

  it "Can login as Guest" do
    open_browser
    login_as("guest", "changeme")
    expect(page_text).to include("Login OK")
    logout
    close_browser
  end

  it "Can login as Administrator" do
    open_browser
    login_as("admin", "secret")
    assert_link_present_with_text("Settings")
    logout
    close_browser
  end
```

```
end
```

By utilizing execution hooks, we can refine these test cases to:

```
describe "User Login" do
  include TestHelper

  before(:all) do
    open_browser
  end

  after(:each) do
    logout
  end

  after(:all) do
    close_browser
  end

  it "Can login as Registered User" do
    login_as("james", "pass")
    expect(page_text).to include("Welcome James")
  end

  it "Can login as Guest" do
    login_as("guest", "changeme")
    expect(page_text).to include("Login OK")
  end

  it "Can login as Administrator" do
    login_as("admin", "secret")
    assert_link_present_with_text("Settings")
  end

end
```

By utilizing RSpec's `before(:all)`, `after(:each)` and `after(:all)` hooks, this version is not only more concise, more importantly, every test case is now more focused (distinguished from each other). Using these hooks effectively will make test scripts more readable and easier to maintain. For readers who are new to RSpec, don't worry, I will cover it more in later chapters.

Cucumber

Cucumber, another relatively new BDD framework in Ruby, is gaining popularity rapidly. To avoid distractions, we will focus on test practices using Selenium-WebDriver + RSpec. There will be a dedicated chapter on Cucumber towards the end of this book.

3.4 Run tests from the command line

In Chapter 2, we created an automated test script using a recorder and ran the test from TestWise.

One advantage of open-source test frameworks, such as Selenium WebDriver, is freedom. You can edit the test scripts in any text editor and run them from command line.

You need to install Ruby first, then install RSpec and preferred web test driver and library (called Gem in Ruby). Basic steps are:

1. **Install Ruby interpreter**
   - Mac

     Ruby is already included in macOS. You may use *rbenv* or *rvm* to install a specific Ruby version.

   - Linux

     Use standard package tool to install, usually just one command such as `sudo apt install ruby-full` on Ubuntu.

   - Windows

     Download RubyInstaller for Windows[2] with Devkit, such as '*Ruby+Devkit 2.7.6-1 (x64)*'. The DevKit is required for compiling certain libraries (called Gem in Ruby).

     Run the installer, don't forget the DevKit setup part.

2. **Install libraries (gems)**

[2]https://rubyinstaller.org/

```
> gem install selenium-webdriver
> gem install rspec
> gem install ci_reporter
```

3. **Install Browser drivers** (such as ChromeDriver for Chrome)

   A browser driver is one executable file (easier to install, see below) provided by browser vendors, such as ChromeDriver[3] for Chrome. Selenium scripts call it to drive the browser.

   - go to ChromeDriver site[4]
   - Download the one for your target platform, unzip it and put **chromedriver** executable in your PATH.

   Make sure the driver version matches your Chrome browser's version. Drop the executable (*chromedriver.exe* on Windows) into a folder in your PATH, for example, `C:\Ruby27-x64\bin`.

To verify the installation, open a command window (terminal for Unix/Mac), execute command `chromedriver --version`, You shall see texts like below:

```
ChromeDriver 104.0.5112.79
```

To run tests in a different type of browser, install its matching driver, such as GeckoDriver for Firefox. Selenium site[5] has the details for all browser types.

Once the installation (takes about 1 minute) is complete, we can run a RSpec test from the command line. you need to have some knowledge of typing commands in console (called Command on Windows).

To run test cases in a test script file, enter the command

```
> rspec google_spec.rb
```

Run multiple test script files in one go:

```
> rspec first_spec.rb  second_spec.rb
```

Run individual test case in a test script file, supply a line number in the chosen test case range.

[3]https://chromedriver.chromium.org/downloads
[4]https://sites.google.com/a/chromium.org/chromedriver/downloads
[5]https://selenium.dev/downloads

```
> rspec google_spec.rb:30
```

To generate a test report (HTML) after test execution:

```
> rspec -fh google_spec.rb > test_report.html
```

The command syntax is the same for Mac OS X and Linux platforms.

4. TestWise - Functional Testing IDE

In Chapter 2, we wrote a simple automated test case using TestWise, a functional testing Integration Development Environment (IDE). Selenium WebDriver test scripts can be developed in any text-based editors or IDEs. You can safely skip this chapter if you had decided the tool. Readers, who want to be more productive with TestWise, might find this chapter useful.

4.1 Philosophy of TestWise

The Philosophy of TestWise:

- **"The Power of Text"**
- **"Convention over Configuration"**
- **Simplicity**

The Power of Text *(inspired from the classic book Pragmatic Programmers)*

Unlike some testing tools, the main window of TestWise is a text-based editor, with various testing functions such as test execution, test refactoring, test navigation, etc. The benefits of using plain text (test scripts):

- Use of Source Control system to track revision and compare differences
- Powerful text editing, such as Snippets
- Search and replace, even across multiple files in project scope
- Refactoring (we will cover this in a later chapter)
- Easy view or edit using any text editors without dependency on the proprietary tool

Convention over Configuration *(inspired from popular Ruby on Rails framework)*

The principle of "Convention over Configuration" is gaining more acceptance with the success of Ruby on Rails framework. It makes sense for writing automated tests as well. In the context of testing, with conventions in place, when a tester opens a new test project, she/he should feel comfortable and can get to work straightaway.

TestWise defines simple conventions for the test project structure, test file naming and page classes, as you will see later in this chapter. This helps communication among team members or seeking help externally when necessary.

Simplicity

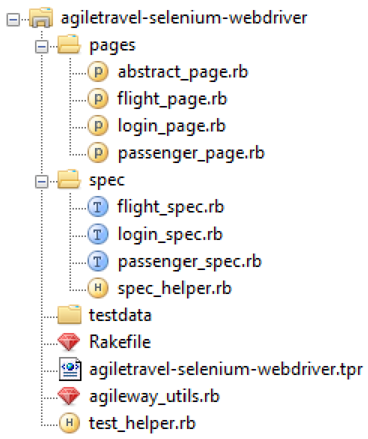
TestWise is designed from the ground up to suit testers, without compromises often found in testing tools that are based on programming IDEs (which are designed for programmers). Every feature in TestWise has one single purpose: a better testing experience.

To make new-to-automation testers more willing to adopt, TestWise is designed to be easy to install, launch quickly and get you started in minutes.

> **Next-Generation Functional Testing Tool**
>
> In October 2007, The Agile Alliance held a Functional Testing Tools Visioning Workshop to envision the next-generation of functional testing tools: "We are lacking integrated development environments that facilitate things like: refactoring test elements, command completion, incremental syntax validation (based on the domain specific test language), keyboard navigation into the supporting framework code, debugging, etc." [AAFTT07]
>
> TestWise was designed and implemented before the workshop, but shares the same vision.

4.2 TestWise project structure

The project structure in TestWise is simple.



```
agiletravel-selenium-webdriver
    pages
        abstract_page.rb
        flight_page.rb
        login_page.rb
        passenger_page.rb
    spec
        flight_spec.rb
        login_spec.rb
        passenger_spec.rb
        spec_helper.rb
    testdata
    Rakefile
    agiletravel-selenium-webdriver.tpr
    agileway_utils.rb
    test_helper.rb
```

There are several file types distinguished by icons in TestWise:

- **Test script files** (xxx_spec.rb)
  One test script file may contain one or more test cases (the extension '.rb' means it is a Ruby script file).

- **Page class files** (xxx_page.rb under `pages` folder)
  A Page Class is a reusable Ruby class representing a web page, we will cover it in detail in the next chapter.

- **Test Helper** (test_helper.rb)
  Common reusable functions are defined in Test Helper. It is included at the beginning of all test script files and the functions are available for all tests.

- **Project file** (xxx.tpr)
  Store project settings. To open a TestWise project, look for a xxx.tpr file.

- **Rakefile**
  Configuration file for Rake build language (equivalent to build.xml for Ant), which can be used to execute all or a custom suite of test cases.

- **Test data** (under /testdata folder, optional)
  The place to put your test data.

4.3 Test execution

Test execution, obviously, is the most important feature for testing tools. TestWise offers several ways to run tests.

Run test cases in a test script file (F10)

A test script file may contain one or more test cases that commonly form a logic group.

Run individual test case (Shift+F10)

When developing or debugging (trying to find out what went wrong) a new test case, you can just run this single test case and leave the web browser at the state when an error occurred for analyse. And yes, this is the most frequently used method for executing tests.

Run All Tests in folder

Also you can run all tests under a folder. Invoke via the context menu from right-clicking a test folder.

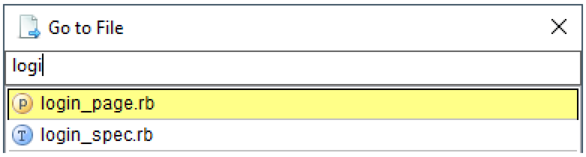
However, I discourage running tests this way. It is not practical to do so when you have many tests, let's say, over 100. Instead, we shall run a suite of tests in a Continous Testing process, so that we may continue to develop/fix/debug tests in TestWise while the test execution is happening on another machine. We will cover this in Chapter 11.

4.4 Keyboard navigation

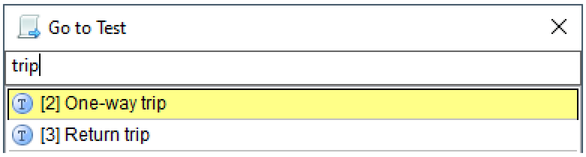
One criterion identified by Agile Alliance work for Next-Gen Functional Testing tools is "keyboard navigation into the supporting framework code". Those who are used to operating with a mouse all the time might find 'keyboard navigation' is just a matter of personal preference, and wonder how it is made into the list?

For any projects that are doing serious automated testing, there will be a large number of test scripts. When the number is big, being able to find the test case quickly, keyboard navigation becomes more than just a convenience.

Go to Test Script File (Ctrl+T)



Go to Test Case (Ctrl+Shift+T)

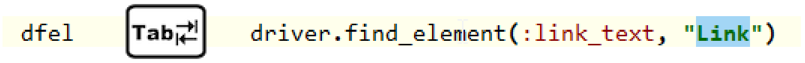


Rocky's mouse

Once I worked with a tester nicknamed Rocky who was in his fifties. Despite many doubts, he fell in love with automated testing quickly. He developed RSI (Repetitive Strain Injury, a potentially disabling illness caused by prolonged repetitive hand movements) with his mouse hand. Certainly years of the using computer mice had contributed to that. When we worked together on test cases, I moved the mouse to the far right side and sometimes even put a piece of paper between him and the mouse. Changing a habit is never easy, but Rocky was doing admirably well. Weeks later, Rocky used the keyboard more than the mouse and felt more productive as a result. Months later after I left the project, I met one of his colleagues, who told me: he saw Rocky once snapped the mouse on his desk, and said to himself: *"Zhimin said not to use it"*.

4.5 Snippets

Snippets in TestWise are small bits of text that expand into full test script statements. The use of snippets helps to create test steps more effectively when crafted manually. For example, type '`dfel`' then press Tab key in a script editor, TestWise will expand it into the test statement below (clicking a hyperlink):



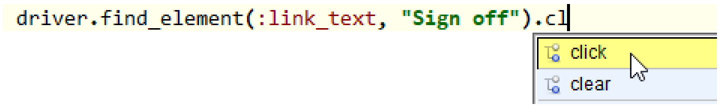
```
dfel
```

```
driver.find_element(:link_text, "Link")
```

There are two ways to insert a snippet:

- Enter snippet abbreviation and press Tab key, or
- Press 'Ctrl+J' and select from the list, or type to narrow down the selection.

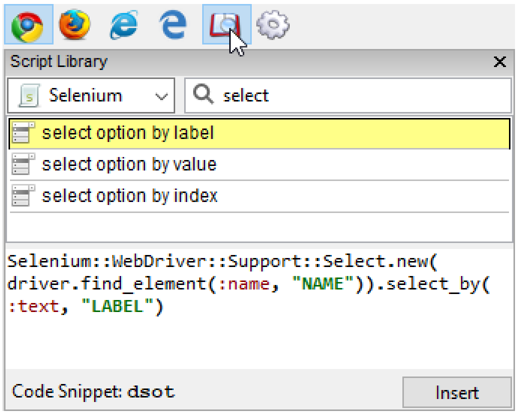
After a snippet is expanded, you may type over the highlighted text and press Tab to move to the next one if there is any. For example, type "Sign off" then press Tab key, the cursor will move to the end of the line. Type `.` and select `click` to complete this test statement.



```
driver.find_element(:link_text, "Sign off").cl
```

4.6 Script library

For testers who are new to the test framework and do not know the script syntax, may have many 'how-to' questions such as: What is the test script syntax for clicking a button?, How to assert the checkbox is checked?, etc. TestWise's built-in script library can provide the answers.



```
Selenium::WebDriver::Support::Select.new(
driver.find_element(:name, "NAME")).select_by(
:text, "LABEL")
```

Code Snippet: `dsot`

4.7 Test refactoring

Test Refactoring is a process of refining test scripts to make it easier to read, and more importantly, easier to maintain. One unique feature of TestWise is its refactoring support, performing test refactoring efficiently and reliably.

We will cover this important topic in later chapters.

4.8 Wrap up

We have quickly introduced some features of TestWise to help you develop test scripts more efficiently. For more details, please check TestWise online documentation and screencasts.

5. Case Study

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.1 Test site

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.2 Preparation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.3 Create a test project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.4 Test Suite: Sign in

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Positive Case: User can sign in OK

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Test Case Design

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Typing the test steps in TestWise Editor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Negative Case: User failed to sign in due to invalid password

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Run all test cases in the login_spec.rb

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.5 Test Suite: Select Flights

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Case 1: One-way trip

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Case 2: Return trip

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Technique: use execution hooks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Technique: check dynamic UI

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.6 Enter passenger details

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.7 Book confirmation after payment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Technique: Testing AJAX

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Technique: Displaying value from specific HTML element in console

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.8 Run all tests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

5.9 Wrap up

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

6. Maintainable Functional Test Design

6.1 Record/Playback leads to unmaintainable test scripts

Record, Refine, Run

6.2 Success criteria

Intuitive to read

Easy to update

6.3 Maintainable automated test design

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Reusable function

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Page Object

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

6.4 Maintain with ease

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

6.5 Case Study: refine test scripts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

DRY with Reusable Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Parameterizing functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

DRY with Page Objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

6.6 Wrap Up

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

7. Test Automation Characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

7.1 Specific

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

7.2 Clean

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

7.3 Independent

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

7.4 Frequent

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Frequency as the needs

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Frequency as the outcome

7.5 Focused

7.6 Programmable

7.7 Creative

7.8 Sustainable

7.9 Wrap up

8. Functional Test Refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

8.1 Code refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

8.2 Functional test refactoring

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

8.3 Tool support

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

8.4 Case study

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Extract “sign in” function

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Extract "sign off" function

Extract FlightPage

Extract PassengerPage

Extract PaymentPage

Move function to test helper

Move to execution hooks

Full test scripts

8.5 Summary

9. Review

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.1 Syntax errors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

How to avoid?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.2 Set up source control

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Git Installation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Set up Git for local working folder

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Set up Git for a shared folder on the network drive

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Frequently used Git commands after set up

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.3 GUI Object Map

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.4 Custom libraries

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.5 Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.6 Cross-browser functional testing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.7 Data-Driven Test

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

9.8 What is the best learning method?

10. Collaboration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

10.1 Pre-requisite

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Version Control Server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Same Testing Tool

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

10.2 Scenario 1: "It worked on my machine"

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Benefits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

10.3 Scenario 2: Synergy

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Benefits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

10.4 Scenario 3: Acceptance Test-Driven Development

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Benefits

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

10.5 Wrap up

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11. Continuous Integration with Functional Tests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.1 Long feedback loop

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.2 Continuous Integration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.3 Continuous Integration and Testing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.4 CI build steps

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.5 Functional UI testing build step with Rake

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Run selected tests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Run all tests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.6 Set up a Continuous Testing server: BuildWise

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Objective

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Prerequisite

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Install BuildWise server

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.7 Create a Build Project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.8 Trigger test execution manually

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.9 Feedback while test execution in progress

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.10 Build finished

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.11 Exercise: Set up CT for your own project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

11.12 Review

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

12. Test Reporting

12.1 Reporting automated test results

12.2 Defect tracking

12.3 Requirement traceability

Traceability matrix

Traceability with BuildWise

Prerequisite

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Generate Traceability Matrix

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

13. WebDriver Backed variants

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

13.1 Watir

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Invoke Selenium WebDriver directly in Watir

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

13.2 RWebSpec

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Invoke Selenium WebDriver underneath in RWebSpec

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

13.3 Capybara

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Invoke Selenium WebDriver underneath in Capybara

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

13.4 Test design with Watir, RWebSpec and Capybara

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Watir with RSpec

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Test script (Watir)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Reusable functions in test_helper (Watir)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Page classes (Watir)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

RWebSpec

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Test script (RWebSpec)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Reusable functions in test_helper (RWebSpec)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Page classes (RWebSpec)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Capybara with RSpec

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Test script (Capybara)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Reusable functions in test_helper (Capybara)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Page classes (Capybara)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

13.5 Review

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

14. Cucumber

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

14.1 Cucumber framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Create Selenium-Cucumber Test Project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Raw Selenium-WebDriver in Cucumber

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Use functions and page objects in Cucumber

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Refactoring: Introduce Page Object

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Run Cucumber tests from the command line

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

14.2 Comparison: RSpec and Cucumber

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

14.3 RSpec and Cucumber co-exist

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15. Adopting Test Automation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.1 Seek executive sponsorship

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Benefit realization

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Software maintenance

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.2 Choose test framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.3 Select test tool

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Run as part of Build Process

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.4 Find a mentor

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.5 Manage expectation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.6 Solo test automation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.7 Common mistakes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Aiming too long

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Juggling with test frameworks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Overestimate test automation effort

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Underestimate test automation effort

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Managers/Developers think "maintaining automated tests slows down the development"

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

No Test Automation Mentors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Lack of Continuous Integration Process

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

15.8 Wrap up

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Appendix 1 Functional Test Refactoring Catalog

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Move Test Scripts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Extract Function

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Move Function to Helper

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Extract to Page Class

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Introduce Page Object

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Rename

Appendix 2 Case Study: Test Automation in ClinicWise project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Build Stats

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Over 612,000 test executions over 7 years

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

609 comprehensive automated UI test cases

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Build time

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Test Automation enables Agile

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Stage 1: Write automated tests on the first day

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Stage 2: Set up CI server within the first week

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Stage 3: Release to production early

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Stage 4: Release often (daily)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Stage 5: Set up parallel test execution in CI

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Questions and Answers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Resources

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/practical-web-test-automation.

Books

- **Practical Continuous Testing**[1] by Zhimin Zhan

  The second book of "Practical" series, focuses on how to effectively execute automated functional tests in a Continuous Testing server.

- **Practical Desktop App Test Automation with Appium**[2] by Zhimin Zhan

  Automating Desktop apps on Windows using Appium framework, using the same test design and practices in this book.

- **Selenium WebDriver Recipes in Ruby**[3] by Zhimin Zhan

  The problem-solving guide to Selenium WebDriver with over 150 ready to run recipe test scripts.

- **API Testing Recipes in Ruby**[4] by Zhimin Zhan

  The problem-solving guide to testing API such as SOAP and REST web services.

- **Learn Ruby Programming by Examples**[5] by Zhimin Zhan and Courtney Zhan

  Master Ruby programming to empower you to write test scripts.

[1]https://leanpub.com/practical-continuous-testing
[2]https://leanpub.com/practical-desktop-app-test-automation-with-appium
[3]https://leanpub.com/selenium-recipes-in-ruby
[4]https://leanpub.com/api-testing-recipes-in-ruby
[5]https://leanpub.com/learn-ruby-programming-by-examples-en

Web Sites

- **Selenium Ruby API**[6]

  The API has searchable interface, The *SearchContext* and *Element* class are particularly important:

  - SearchContext[7]
  - Element[8]
- **Watir API**[9]
- **Selenium Home**[10]
- **RSpec**[11]
- **Cucumber**[12]
- **My Blog on Medium**[13]

  With 200+ blog articles in Test Automation, Continuous Testing, Programming, ..., etc. Many of these articles are featured in various software testing newsletters.

Tools

- **TestWise IDE**[14]

  AgileWay's next-generation functional testing IDE supports Selenium, Watir with RSpec and Cucumber. TestWise Community Edition is free.
- **Apatana Studio**[15]

  Free Eclipse based Web development IDE, supporting Ruby and RSpec.
- **BuildWise**[16]

  AgileWay's free and open-source continuous build server, purposely designed for running automated UI tests with quick feedback.

[6]https://www.rubydoc.info/gems/selenium-webdriver/Selenium/WebDriver
[7]https://www.rubydoc.info/gems/selenium-webdriver/Selenium/WebDriver/SearchContext
[8]https://www.rubydoc.info/gems/selenium-webdriver/Selenium/WebDriver/Element
[9]https://www.rubydoc.info/gems/watir/
[10]https://www.selenium.dev
[11]http://rspec.info
[12]http://cukes.info
[13]https://zhiminzhan.medium.com/
[14]https://agileway.com.au/testwise
[15]http://aptana.com
[16]https://agileway.com.au/buildwise

References

[Crispin & Gregory 09] Lisa Crispin and Janet Gregory. 2009. *Agile Testing*. Addison-Wesley Progressional.

[Shore & Warden 08] James Shore and Shane Warden. *The Art of Agile Development.* 2008. O'Reilly Media Inc.

[Hunt & Thomas 00] Andrew Hunt and David Thomas. 2000. *The Pragmatic Programmer: From Journeyman to Master.* Addison-Wesley Progressional.

[Fowler et al. 99] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts. "Refactoring: Improving the Design of Existing Code", Addison-Wesley Progressional.

[Fowler 00] Martin Fowler, "Continuous Integration (original version)" (posted Sep 10, 2010) http://martinfowler.com/articles/originalContinuousIntegration.html

[Crispin 07] Lisa Crispin, "To Track or Not to Track". *http://lisacrispin.com/downloads/TrackOrNot.pdf*

[Crispin 08] Lisa Crispin, "The Team's Pulse: CI/Build Process" (posted on Aug 23, 2010) http://lisacrispin.com/wordpress/2010/08/23/the-teams-pulse-cibuild-process/

[NEWS 11b] "Queensland Health payroll will cost up to $220 million to fix, acting director-general admits". Jul 14, 2011. *http://www.couriermail.com.au/news/queensland/queensland-health-payroll-will-cost-up-to-220-million-to-fix-acting-director-general-admits/story-e6freoof-1226094095536*

[NEWS 10a] "Pay system not properly tested: report", Jun 29, 2010. *http://www.computerworld.com.au/article/351475/pay_system_properly_tested_report/#closeme*

[Calzolari et al, 98] F. Calzolari, P. Tonella and G. Antoniol. 1998. "Dynamic Model for Maintenance and Testing Effort". Software Maintenance, 1998. Proceedings.

[Wells 09] Don Wells, "The Values of Extreme Programming" *http://www.extremeprogramming.org/values.html*

[NEWS 11c] "What would Larry Page do? Leadership lessons from Google's doyen". April 18, 2011. http://management.fortune.cnn.com/2011/04/18/what-would-larry-page-do-leadership-lessons-from-google's-doyen/

[Cockburn 04] Alistair Cockburn. 2004.
http://groups.yahoo.com/group/scrumdevelopment/message/2977?threaded=1&p=25

[Feathers 10] Michael Feathers. "UI Test Automation Tools are Snake Oil", Object Mentor Blog (posted on Jan 4 2010). *http://blog.objectmentor.com/articles/2010/01/04/ui-test-automation-tools-are-snake-oil*

[IDT07] Bernie Gauf and Elfriede Dustin. 2007. "The Case for Automated Software Testing". *http://journal.thedacs.com/issue/43/90*

[AAFTT09] AAFTT Workshop 2009. *http://cds43.wordpress.com/2009/10/06/aaftt-workshop-2009-chicago/*

[AAFTT07] Agile Alliance Functional Testing Tools Visioning Workshop. 2007.
http://www.infoq.com/news/2007/10/next-gen-functional-testing

[Myers, Glenford 04]. The Art of Software Testing. Wiley. ISBN 978-0471469124.

[FSF] The Free Software Definition, http://www.gnu.org/philosophy/free-sw.html