28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Web Test Automation Framework with Open Source Tools powered by Google WebDriver

KapiL Bhalla, Nikhil Bhandari
Engineer, QuickBooks Online
Intuit, Inc.
Kapil_Bhalla@intuit.com, Nikhil_Bhandari@intuit.com

Abstract

Building test automation harness on open source libraries often causes more pain than pleasure. Factors like application complexity (variety in web elements used on a web page, nested frames, iframes, etc), project scope (number of web pages), test automation team size, learning curve of automation libraries, etc… lead to high maintenance and less adaptable solutions. In such cases principles of abstraction and design patterns come to rescue and help us attain better maintainable, adaptable and thus sustainable solutions. This paper will demonstrate how application these principles of abstraction and design patterns can aid us in building flexible test automation harnesses.

When we looked at QuickBooks Online for first time for automating its testing we knew that we were facing a daunting task. Some of the challenges we were confronted with were supporting automation for more than 100 web pages, carrying out testing on modal windows, handling AJAX controls, testing Java Script error handling, coping up with constantly changing product and User Interface with changes going into the web application once every 6weeks, managing contributions from geographically spread teams in US and India. Our exercise evolved with time into an operational test automation harness.

This paper will provide some context on various factors which make tailoring a test automation harness challenging and measures which will make your solution work. We will also propose a Blue Print for test automation harnesses.

Biography

KapiL is a Quality Engineer in QuickBooks Online and responsible for test automation. He has been with Intuit since Jan 2008. He received a Masters of Computer Application degree from National Institute of Technology, Karnataka India and a B.Sc. degree in Computer Science from University of Delhi, India.

Nikhil holds an Engineering degree along with 10 years of experience. He is currently working with Intuit as a Staff Software Engineer - QA and has previously worked with Oracle, McAfee & Satyam Computers in Bangalore, India. He has been speaker at STARWEST Conference 2008 (USA), Free-Test Conference 2009, 2010 (Norway), Step-In Forum Evening Talk (India).

1. Introduction

This paper shares learning from journey of building user interface test automation harness for over a decade old web product with constantly changing user interface and geographically separated teams testing the web application.

The requirements that we faced were making test automation harness simple for quality engineering team and demonstrating to the management that in long run changes to the web application can be accommodated in test automation harness at a low cost. Making the test automation harness maintainable, flexible and work across different types of user scenarios in the web application were the prime challenge. This required handling variety of web elements like radio buttons, select dropdowns, check boxes, text boxes, frames, nested frames, etc… and simulating user behavior using mouse actions and keyboard actions. While most of the interaction with the web elements was solved by Google's web driver we were left with the bigger requirements making it easy for test automation engineers to use the test automation harness and to make it maintainable. Let us see in detail how we solved for these requirements.

2. Background

The target application in our case was an accounting web application. Requirements included automating testing across – multiple browsers like - Internet Explorer, Firefox and Safari on windows and Mac operating systems. The product spanned over 100 web pages, and multiple web technologies like HTML, Java Script, Flex, CSS, etc… That have gained acceptance over the last decade. Additional challenges included enabling multiple members to contribute to automation testing without bumping into each other's domain specialization, and to be able to achieve code/ workflow/ test reuse wherever possible.

The dynamism in requirements came along when some of the backend services used were changed in initial months, presenting multiple web pages doing the same redundant things (example: login) in the application. Multiple login pages demanded test automation harness to provide support for both login pages at run time this was made possible by using factory pattern.

Choosing Google Web Driver as the backend library for the test automation harness required the automation team to spend time on learning and understanding the libraries. To minimize the learning curve we decided to hide web driver libraries and provide simple APIs for test automation engineers to use.

3. Understanding the requirements and Meeting them

Apart from the default requirements of executing automated tests across various browsers on multiple platforms we were confronting few other requirements. These requirements are:

1. Easy to Use test automation harness / Libraries
2. Easy to learn test automation harness / Libraries
3. Reusable Code/ Workflows/ Tests
4. Maintainable test Automation Code
5. Support for multiple contributors for the test automation code
6. Flexible, loose tool coupling

Let us dive deeper into these requirements and see how they were met.

3.1 Meeting Requirement: Easy to Use test automation harness

What could be one of the easiest usability cases that a test automation framework can provide today? You may be right with your answer *"Record and Play"*. It is true to a large extent that recording is simple way of generating code/ script and there is no doubt in that. There is usually some expected effort involved in refactoring these recorded scripts into some kind of 2-level / 3-level architecture to ensure maintainability. If refactoring is not done after record and play and your application is of reasonable size and is evolving on UI front then maintainability goes for a toss. On the other hand test automation harness / frameworks demand refactoring test automation code upfront- and the gains of abstraction are harnessed later in the life cycle of test automation code by means of reduced maintenance cost.

The target that we set for our self was exposing Java APIs for automation of possible actions a user can carry out on the web pages of our application. This is enabled by modeling the web pages in Page Object classes and the possible set of actions on the page as methods of the Page Object class (Standard model for UI automation). Let us look at this by an example,

```
// This is a sample Page Object representing a standard login page.
Class LoginPage{
        …
        private username = null;
        private password = null;
        private login = null;
        …
        public void setUsername (String username){
                username.sendKeys (username);
        }
        public void setPassword (String password){
                password.sendKeys (password);
        }
        Public void clickLogin (){
                login.click();
        }
        …
}
```

Some Benefits of this approach include limited impact zone for UI changes and clear separation between test code and web page model. On the downside modeling web application in page object initially consumes time.

Modeling web pages in page objects provides level -1 abstraction of the web product that is being tested. A clear benefit for test automation contributors is test case/ workflow will look similar to a <OBJECT>.<METHOD>(<PARAMETERS>) composition and assertions if any. You will shortly see how we build more on this abstraction. For example: a test workflow for successful login will look similar to,

```
public void testLogin(){
        LoginPage loginPage = new LoginPage();
        loginPage.setUsername("KapiL");
        loginPage.setPassword("Bhalla");
        loginPage.clickLogin();
        // code to assert that we landed on correct page post login
        …
}
```

With the specifics of the automation libraries encapsulated into methods of web page objects a test automation engineer gets to use simple java methods to simulate user actions on the web page. This makes the test automation harness relatively easy to use.

3.2 Meeting Requirement: Easy to Learn test automation harness

With the page objects in place, test automation engineer is largely saved from learning libraries of automation tool being used. This reduces the coding effort, learning effort and required skill expected from test automation engineer. By generating Java Docs for all web page objects in the test automation harness easy to read documentation can be provided to the test automation engineers, this will further aid a test automation engineer in learning the test automation harness.

3.3 Meeting Requirement: Reusable Code/ Workflows/ Tests

Fall back in your chair and imagine the login page of your favorite application (let us assume that to be *gmail)*. Given the task of automating the following 3 test cases:

1. Successful login
2. Wrong password attempted, and
3. Password left empty while logging in

What would do? Would you like to reuse the following 3 lines to code?

```
loginPage.setUsername("<your-user-name>");
loginPage.setPassword("<password-content>");
loginPage.clickLogin();
```

Or will you prefer a parameterized method called loginAs(username, password) which executes the above mentioned 3 lines of code? We will refer to methods like loginAs(…) as workflows and they help us grouping together common code that needs to be executed several times, this is workflow reuse.

Now consider that you want to login into gmail and test functionality of some cool features of gmail and while doing that you want to test login functionality every time. Given this requirement you may choose to put all assertions and validations along with the workflows that you grouped in methods or you may directly reuse your parameterized testSuccessfulLogin(…), this kind of reuse is referred to as test case reuse. Doing so not only helps us reduce the coding effort while automating but it also ensures that more frequently used components of your application are validated in proportion of their use.

Reconcile

Taking few moments to reconcile what we have till now shall be of use,

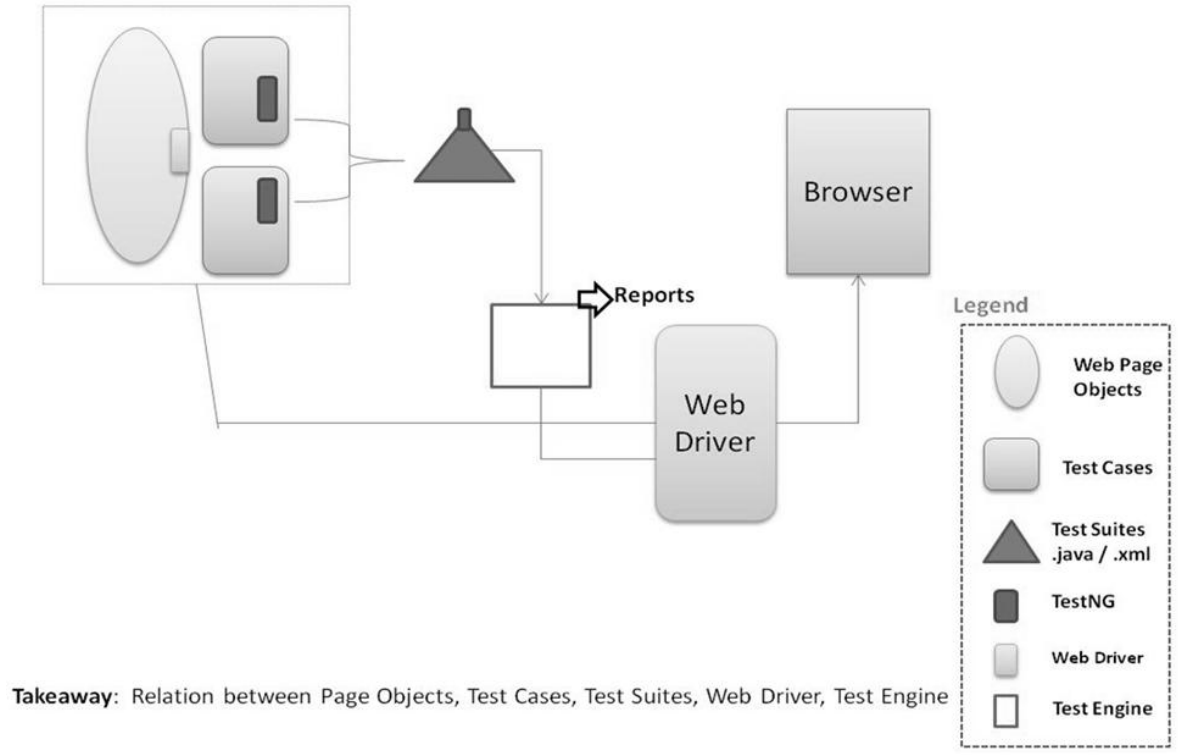


Diagram1: Represents architectural relation between page objects – workflows – test code

The above architecture diagram summarizes how web pages of your web application are modeled by page objects, how page objects are used by the workflows, and how test code use methods in page object and/ or workflows and/ or other test code.

3.4 Meeting Requirement: Maintainable test automation harness

If done well this shall be one of the major cost savings factors in long term. Historically speaking 'species adapting fastest to change has been most successful in surviving evolution', given that the only constant in evolving software is 'change' one of the biggest challenges for us was how open we were to changes being made in UI of our web application.

The change can be a simple change of attribute of an element on the web page (say an ID) to introduction of a parallel web page offering exactly same functionality as the one before (say a new version of Login page being tested for Beta experience). Let us list down some of the possible changes that are common to UI of a Web Application:

1. Changing attributes of an existing element on the web page
2. Existing elements being deleted from the web page
3. New elements being added on the Web Page
4. Introduction of a new page replacing the old page
5. Introduction of a new page along with the old page

The list of possible changes only grows during life span of a web application; this makes handling change not only important but critical. Diving deeper will reveal how we handled change. Architecture shown in

diagram 1 reduces lots of rework by enabling interaction with elements on a web page in object oriented manner, but there still exist few things which can be improved upon. On a closer inspection we found:

1. API calls within page object that bind us to the tool we are using (in our case WebDriver). Example:
    a. driver.findElement(By.id("<AN_ELEMENT_ID>"));
    b. element.sendKeys("<KEYS_TO_TYPE_INTO_THE_ELEMENT>");

2. Tight coupling of web element attributes to locate the web element on the web page. Example: IDs being used to locate the elements on the web page. "<AN_ELEMENT_ID>" in the above example.

This demands abstracting out the tool specific code from the page objects and abstracting out the attributes used to locate the web elements on the web page we will see how this was solved.

Abstracting out the tool/ library (WebDriver) from the page objects, demanded creation of a utility that will find elements on the given web page once provided with the attributes of the web element. All elements being interacted inside methods of page objects of the test automation harness are required to use this utility to locate the web element on the page.

Abstracting out the attributes of web elements from page objects confronts us with 2 choices:

1. Pass the attributes at runtime when test code is using the methods of the page object.
2. Load and make available elements on the web page when the page object is instantiated or upon use of an element.

This opens up another choice between Lazy loading and Eager loading of attribute specific data into the page object. With the classic tradeoffs of each we leave this choice to the inspired engineers from this paper. We choose option 2 and persisted attributes of the web elements in configurable 'Web Element Repository Files'.

Abstracting out the attributes of web elements from page objects offered tremendous flexibility while making the test code ready to accept changes in application UI, freedom from recompilation of page objects in case of change in attributes lead to reduced cost in handling one kind of change.

These two abstractions - test tool abstraction and web element attribute persistence - brings us to the revised architecture of the test automation harness,

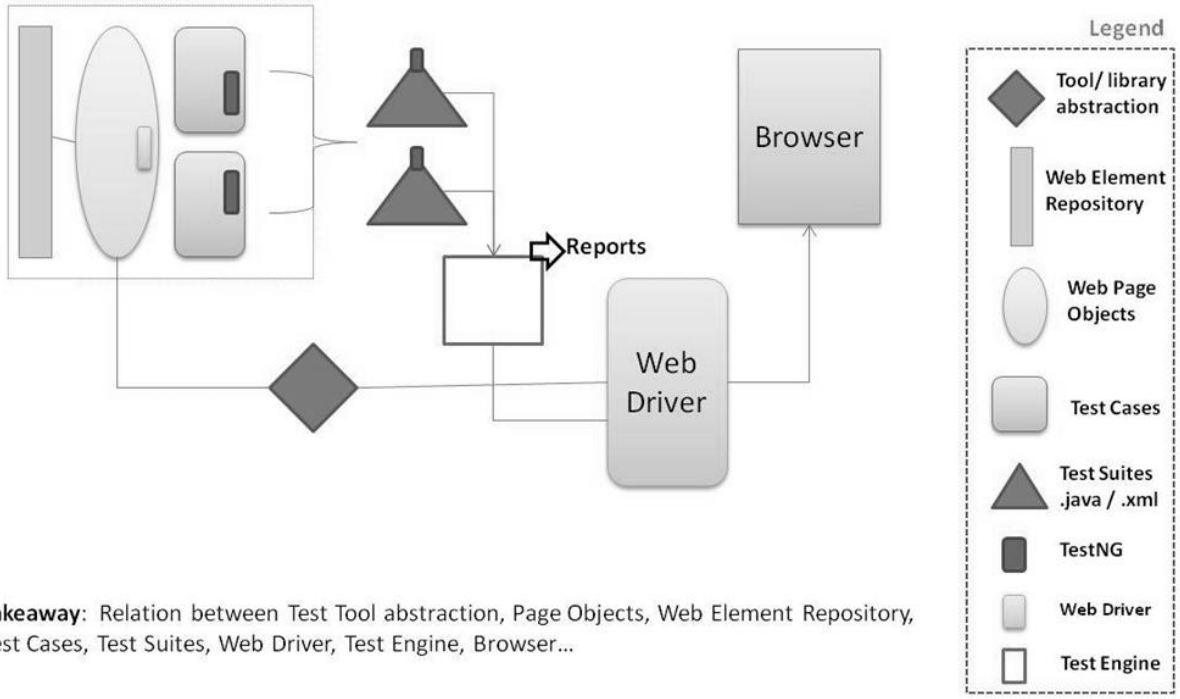


Diagram2 – Adding test tool abstraction and web element attribute persistence to test automation harness

3.4.1 Changing attributes of an existing element on the web page

With attributes of elements on the web page being persisted in configuration files/ database it is easy to make changes to them. The required changes to the attributes of elements like change in ID, finding the element using NAME, etc can be made by updating the respective element field / record in the persistence layer. Now changes to attributes of elements are not expensive and don't require compilation of page object or test automation code.

3.4.2 Existing elements being deleted from the web page

If an element is deleted from the web page then the corresponding entries in persistence layer needs to be deleted and methods using the element in the page object need to be updated. This kind of change can also impact automated test code written using methods of the corresponding web page object where the element was showing up. This is an expensive change and needs to be handled at all levels.

If the deleted element is replaced by a similar element then the web element mapping between the persistence layer and test automation harness can be reused and the impact can be less.

3.4.3 New elements being added on the Web Page

On addition of new elements on a web page we will need to add entries in the persistence layer and update the web page object to make available the set of possible actions on the web page using the new web element. This is also an expensive change and is likely to be made at all levels in the test automation harness.

3.4.4 Introduction of a new page replacing the old page

On addition of a new page replacing an old web page we need to provide support for the new page at page object level. If the page offers same functionality which was being provided by its predecessor then we may use the same page object, in this case change will be required at web element persistence layer and in body of methods of the page object.

3.4.5 Introduction of a new page along with the old page

When we need to support multiple pages offering the same functionality in the web application and we are not sure on which web page will appear on runtime in the workflow that is being automated, factory pattern comes in handy. In this case we have support for both web pages in the test automation harness at all levels, based on which page appears in our workflow we detect the page and get object of the respective page from the page factory in the test automation harness. This is relatively less likely to happen and factory pattern is one elegant way of handling the requirement of supporting multiple pages providing same functionality in parallel.

3.5 Meeting Requirement: Support for multiple contributors

Enabling multiple engineers to contribute in test automation is a well known requirement that we get when we set ourselves with the task of automating testing for a monster sized application. But with this requirement comes responsibility of managing all cumulative code that engineers write. Giving full freedom to modify or add code in any part of automation can be recipe for code maintenance nightmare, and too limited access can lead to parallel test automation for overlapping areas in the web application.
It is too early to comment on ROI of our implementation but sharing what we chose to do is precisely the reason this paper is being written.

We have chosen to arrange our test automation effort in alignment with the team structure of the development/ quality engineering team. The development/ quality engineering team is aligned according to areas/ features of the web application, with this team structure engineers working on particular areas of the application hone their skills to become domain experts. Packaging the test automation harness code in alignment with areas/ features of the web application empowered contributors/ domain experts with much needed independence while automating the tests and abstraction to others from test automation work being done on a specific feature of the web application. Some of the application areas and packages (java) in test automation project are:

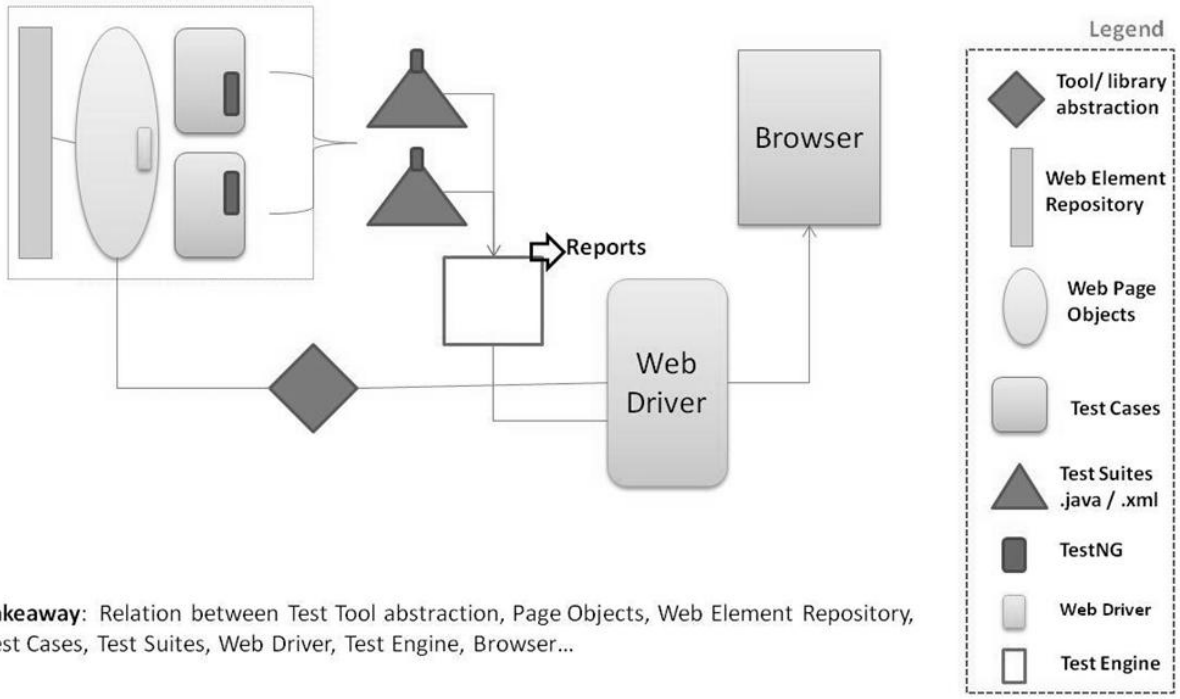
1. Login / Logout
2. Home Page
3. Administration Pages
4. Application Preferences, etc…

3.6 Meeting Requirement: Flexible, loose tool coupling

Separation of automation code into Page Objects and test automation code makes test automation code free from tool specific code. To further reduce the coupling of tool/ libraries we build a core class which carries out interactions with the elements on the web page, this is exposed by several methods offering basic interaction possible on different elements on a web page. With the core class ready all methods in page objects will refer to methods in it, this encapsulates tool specific activities largely to one core class, this makes the page objects free from tool/ library specific code.

This opens an opportunity of moving to a different tool/ library in case required with savings of test code, page objects, and Web Element repository. It also creates an opportunity of using multiple tools in core classes beneath Page Objects to get the automation job done.

4. Blueprint of the test automation harness



Takeaway: Relation between Test Tool abstraction, Page Objects, Web Element Repository, Test Cases, Test Suites, Web Driver, Test Engine, Browser…

Summary

The challenge of building test automation harness was not an easy one. What mattered most for us was standing up to the challenge of using a free / open source library to achieve our goals in low cost manner. Today we successfully execute automated top 10 user workflows and Build Acceptance Tests everyday and save more than 6 engineering hours every week on execution both test automation suite.

We are still on the journey of making improvements on the test automation harness and increasing the Return on Investment, the test automation harness is integrated with fitnesse (http://fitnesse.org/) acceptance testing framework and being used by a team of 18 engineers for carrying data driven testing on already automated workflows. We have come a long way in implementing an automation test harness for a very complex and dynamic web application from having absolutely nothing to begin with – along the process in addition to the explicit ROI the team has gained absolute experience and is now in a position to accept any test automation challenge with a “can do” confidence.

References

http://en.wikipedia.org/wiki/Factory_pattern
http://code.google.com/p/selenium - Google WebDriver now comes bundled with Seleniun 2.0a5 bundle
http://code.google.com/p/selenium/wiki/DesignPatterns
http://fitnesse.org – Acceptance Testing Framework