Meeting Challenges of Agile Development using Selenium

Khemlal Sinha

Khemlal.sinha@intel.com

Abstract

UI-based automation has its own advantages. We realized this while trying to keep up with a product that releases quarterly, has a very short test cycle, supports multiple platforms, and requires continuous end-to-end validation. While test automation was the obvious solution, the bigger challenge was to identify a framework and tool that would adapt quickly and provide cross platform/browser support.

We came up with a solution to automate our web based manageability product with an open source tool – Selenium. It had the capability to automate web based applications and works on cross browser platforms and was easily integrated with continuous integration environments. As we built on this framework, it increased our confidence in the product for dealing with multiple dependencies on a short test cycle.

One may argue that UI based automation is not always scalable. We want to present a case where UI based automation has not only helped us scale our product release but also covered different complexities that other tools could not provide. This paper covers:

- The challenges of a fast paced web based product release that is supported on multiple platforms.
- The importance of identifying the correct automation tool to address short term and long term challenges.
- Selenium's extensibility and advantages over other tools.
- How Selenium and TestNG were used to enable continuous integration.

Biography

Khemlal Sinha is working as Sr. Quality Assurance Engineer in Intel Security group. He has done his Engineering in Computer Science .He has 8+ years of experience Quality Assurance and Automation. He has worked on various automation tools, scripting languages.

Copies may not be made or distributed for commercial use

1.0 Introduction

Today, most organizations follow agile development models to release products in fast-paced markets. Adopting agile development has many advantages, but at the same time it comes with challenges. The most important challenge is to release a high-quality product in a short release cycle. To achieve the quality goals, we need to make sure that:

- The products are tested on multiple supported platforms and browsers.
- All the end-to-end feature testing is performed frequently.

Our team works on McAfee Foundation Service (MFS) which is the core platform product for other Intel Security products. Many point products are dependent on our release. Also, our team is spread across time zones working on the same code base and checking-in code every day. We wanted to make sure that developer changes do not break the product functionality. Our priority was to make sure that we released a high quality product to the market in a short duration.

1.1 Agile Challenges

We had several challenges while releasing the product after adopting an agile model.

We followed an agile model and we had two week sprints. Developers wanted to make sure checked-in code did not break the build. To achieve this goal and raise confidence in the build we wanted to have some kind of UI automation in place to run every day, or on each check-in.

Stakeholders wanted to have a solid regression test suite to check the product on frequent basis and ensure feature functionality is not broken.

Writing a unit test (testing a class in isolation of the others) is sometimes costly and time consuming. In this case, we wanted to have options for automating user stories acceptance tests.

We wanted to have better code coverage to make sure that our tests validated the code base.

Since our product supports a variety of browsers and platforms, we wanted to test it on different platforms and browsers. This would allow us to validate the product's performance and behavior across platforms and browsers. During some of our past tests, we found differences in the product's behavior from browser to browser.

1.2 Problems with manual testing

Agile sprints give very little time to manually run regression test the functionality of the product.

Manual testing is cost efficient when the test case only needs to run once or twice. However, in the agile process, all the features require testing multiple times, and frequently. This approach hasn't been very cost effective in terms of resource and time.

Manual testing is sometime not reliable. If the QA person gets changed, test execution may not be accurate. To execute the test cases the first time using manual testing will be useful. However, manual testing may not catch regression defects under frequently changing requirements.

Using manual testing, testing on different machine with different OS platform combination is not possible, concurrently. To execute such a task, different testers are required.

To overcome these kinds of challenges, we wanted to have a have a solid automation tool in place which was portable and can be easily integrated with a continuous environment (CI) and can be run very frequently.

2.0 Justification for choosing UI automation

One may argue that you have unit and integration test as part of automation, why do we need UI automation? Unit tests work better where we want to test a unit or method of the developed code but there were few instances where we found UI automation looked better option for our product testing. Here are a few situations where Selenium does a better job than unit or integration testing.

2.1 Test requires workflow validation

Working on the web based enterprise product where we have to test workflow, unit and integration testing may not be a feasible solution. We found UI automation does better job. A simple example is creating a task which requires navigating through five different pages to create the task. Checking one by one all the pages may not be a feasible solution using unit and integration test.

2.2 Validating Drag & Drop

We have a few pages in our product where we have to drag and drop a Dashboard to see the details. Drag and drop cannot be tested using unit and integration test. Here we found UI automation is a better option for us.

2.3 Validating Client-side JavaScript

We had several instances where the JavaScript UI was broken. There were no tests written to exercise JavaScript. For example, the MFS login page has a Logon button which should not be enabled until the user name and password fields have been entered. This functionality was broken and button was getting enabled after the user name was entered. You cannot test this functionality in unit or integration test. Here, UI automation was a better to test java script functionality.

2.4 Testing Tooltips

Tooltips are a very important feature in our product and used extensively. Testing was not possible through unit or integration test. Here we found UI automation is a better option for us.

2.5 Usability testing

From a usability point of view where we wanted to mimic actual workflow testing, we found UI automation was a better option for us.

2.6 Testing Cloud Applications

When working on cloud projects where they have many components involved and lots of redirection, testing is quite complex. Automating through unit or integration tests was time consuming for us. Here we found that UI automation worked better job for us.

For example, the cloud application supports authentication through third party applications and then after successful authentication, it uses many internal components and finally to a successful login page. We tried it to automate through integration tests, but the cost was very high and was time consuming. We were able to automate it through Selenium very quickly.

3.0 Which automation tool to use

It was difficult for us to choose a cost effective automation tool to fit our requirements. We looked into tools like QTP, Test Complete and Selenium and we found that Selenium has many advantage over others tools.

Our product was developed in Java programming language and we wanted an automation tool which was cost effective and used the same programming language. Since the product supports multiple browsers and OS platforms, we wanted to test across browsers and platforms. To achieve this goal we wanted to use an automation tool that supported these requirements.

It has support for all of the popular browsers like IE, Firefox, chrome etc. It also supports several Operating Systems and that makes it a tool of choice for cross browser/ cross platform certification. An example is provided here to show Selenium ability to execute same test suite on multiple browser using TestNG framework.

An example in *Figure 1* shows how TestNG with Selenium was configured to execute tests on multiple browsers.

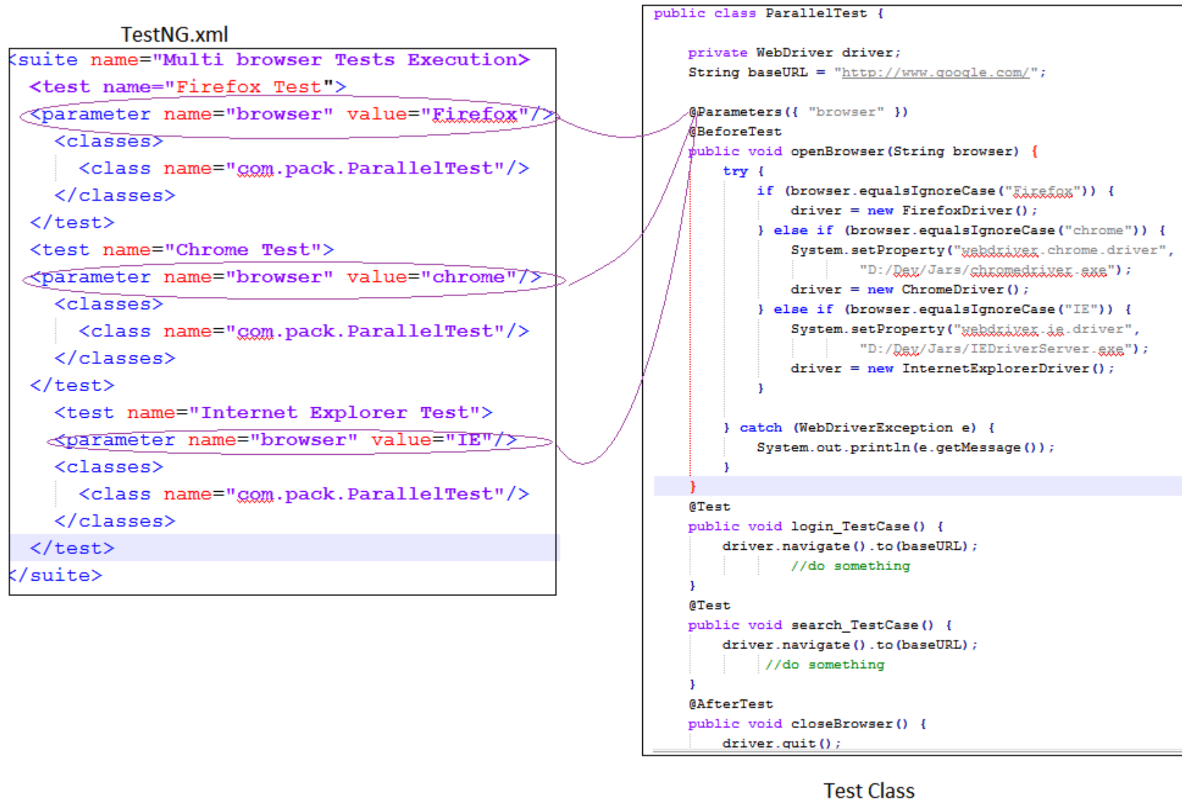

Figure1-Configuring Selenium test to run on multiple browsers

Selenium test cases can be easily grouped and can be executed based on the requirements. TestNG has a unique feature for grouping tests methods which does not exist in JUnit framework. It allows you to execute methods belongs to that particular group. In TestNG, you can declare one method belong to one or more groups, even a certain set of groups can be included or excluded in groups. Group can be used in various scenarios to organize tests methods in better way.

An example would be if you have 300 methods in a class and you want to execute 25 methods as a part of Build verification test (BVT) and rest as a part of regression. You can assign a group called “BVT” to those 25 methods and you can assign group “regression” to rest 275 methods.

The example in *Figure 2* has four test methods, method testingFeatureMethod1 and testingFeatureMethod4 have a group called bvt, and testingFeatureMethod2 and testingFeatureMethod4 are included in group regression. We can specify the group we want to execute in TestNG.xml and it will take care of identifying and executing method in test class.

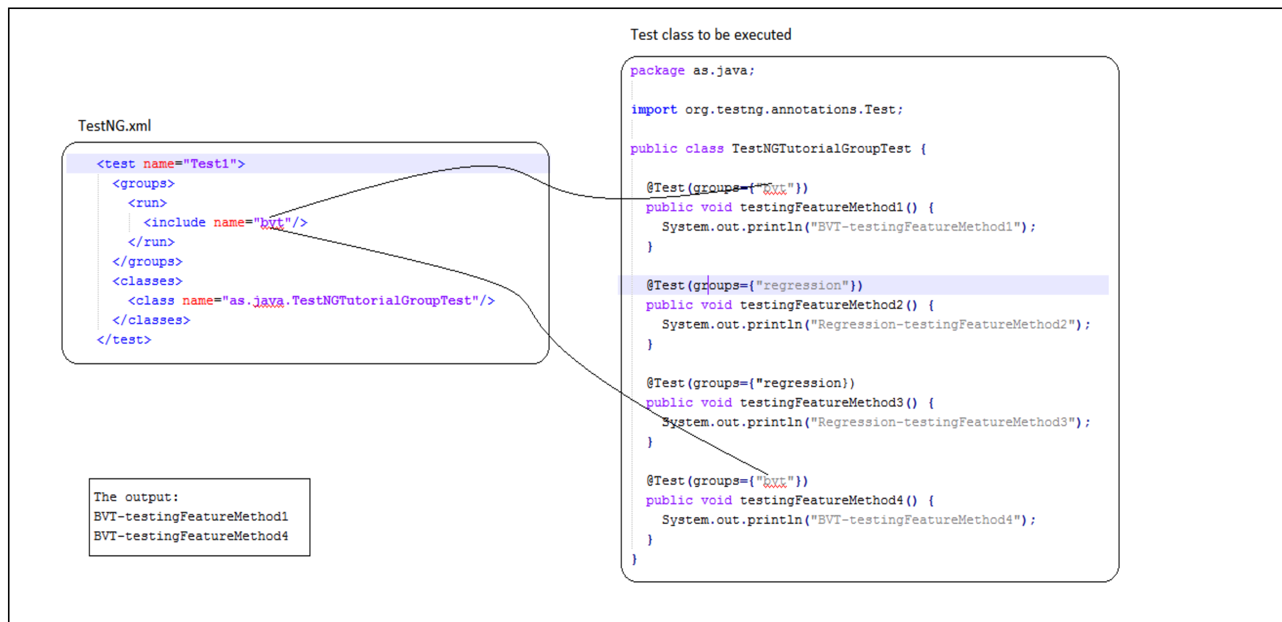

Figure2-Configuring Selenium test to run on group basis

Selenium was first written in Java but it also supports .Net, Ruby, Perl, PHP and Python. This is a big plus when you want to build your framework in a language that has the highest adoption in your organization, unlike tools like QTP which force you to use VBScript.

Selenium has a robust set of tools like Selenium IDE, WebDriver, Selenium Grid, and Selenium Server that supports rapid development of test automation for web-based applications.

Selenium operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior.

You can instantiate several concurrent tests with Selenium which will be helpful to execute your test-suite fast and save execution time.

TestNG provides an ability to run test methods, test classes and tests in parallel. By using parallel execution, we can reduce the execution time as tests are started and executed simultaneously in different threads.

Let us look at the basic example for parallel execution of test methods using TestNG.xml. We have defined thread count as two in TestNG.xml. When TestNG starts test execution, it will assign one thread to each test method in the test class and two tests methods will be executed in parallel.

Copies may not be made or distributed for commercial use

```java
package com.parallel;
import org.testng.annotations.Test;

public class TestParallelOne {

    @Test
    public void testCaseOne() {
        //Printing Id of the thread on using which test method got executed
        System.out.println("Test Case One with Thread Id:- "
                + Thread.currentThread().getId());
    }

    @Test
    public void testCaseTwo() {
        ////Printing Id of the thread on using which test method got executed
        System.out.println("Test Case two with Thread Id:- "
                + Thread.currentThread().getId());
    }
```

Figure 3 – A class with two test methods will be executed via TestNG

Figure 4 shows the simple TestNG.xml file. It defines two attributes, 'parallel' and 'thread-count', at the suite level. Since test methods are executed in parallel, we have provided 'methods'. The 'thread-count' attribute is used to pass the number of maximum threads to be created.

```xml
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Parallel test suite" parallel="methods" thread-count="2">
  <test name="Regression 1">
    <classes>
      <class name="com.parallel.TestParallelOne"/>
    </classes>
  </test>
</suite>
```

Figure 4 – TestNG.xml configured to execute test parallels by providing thread counts

The output looks like *Figure 5.*

```
<terminated> testng.xml [TestNG] C:\Program Files\Java\jre7\bin\javaw.exe
[TestNG] Running:
  D:\Dev\TestNGParallel\testng.xml

Test Case One with Thread Id:- 9
Test Case two with Thread Id:- 10

===============================================
Parallel test suite
Total tests run: 2, Failures: 0, Skips: 0
===============================================
```

Figure 5 – Result of TestNG tests execution

This shows two methods executed using different threads. It can be easily integrated with a Continuous Integration environment such as Teamcity and can be configured to run on each check-in or on nightly build.

Selenium has another benefit. Unlike other commercial automation tools where you have to pay for license costs, Selenium is open-source and freely available.

4.0 Choosing a UI automation model

Starting a UI Automation in Selenium WebDriver is not a tough task. It requires finding the elements in the page and performing action on it.

Consider this simple script mentioned in *Figure 6* to login into a website

Figure 6 – Selenium script to login to a website without using Page Object Model

As you can observe, all we are doing is finding elements and setting values for those elements.

This is a small script. Script maintenance looks easy. But with time, the test suite will grow. As you add more and more lines to your code, things become tough to manage.

The chief problem with script maintenance is that if ten different scripts are using the same page element, any change in that element means changing all ten scripts. This is time consuming and error prone.

A better approach to script maintenance is to create a separate class file which would find web elements, fill them or verify them. This class can be reused in all the scripts using that element. In the future, if there are changes in the web element, we need to change one class file and not ten different scripts.

This approach is called the Page Object Model (POM). It helps make code more readable, maintainable, and reusable.

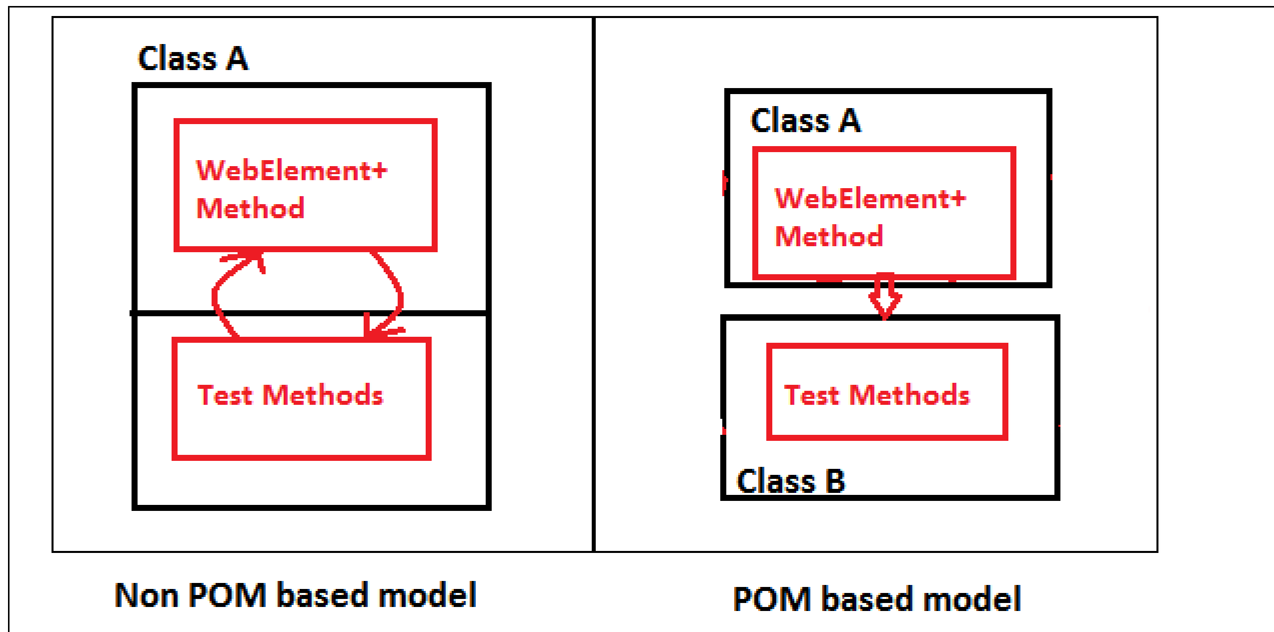

Figure 7 – Non Page Object Model (POM) vs. Page Object Model (POM)

4.1 What is POM?

- The Page Object Model is a design pattern to create an Object Repository for web UI elements.
- Under this model, for each web page in the application there should be corresponding page class.
- This Page class will find the WebElements of that web page and contains Page methods which perform operations on those WebElements.

An example is shown in Figure 8 where a page had been created separately which contains element factory and method to perform action on the element. A test method had been written separately perform actual test which internally calls reusable method of the page. These pages can be used again in any other test method. When elements change, we need not to go to all the test methods; we need to change only in the page where that element is being used.

ServerTaskWizardDescriptionPage

```java
public ServerTaskWizardDescriptionPage(T parent, WebDriver driver) {
    super(driver);
    this.parent = parent;
    this.next = new ServerTaskWizardActionsPage(this, driver);
}

public ServerTaskWizardDescriptionPage<T> name(String name) {
    ui.type(NAME, name);
    return this;
}

public ServerTaskWizardDescriptionPage<T> notes(String notes) {
    ui.type(NOTES, notes);
    return this;
}
```

ServerTaskWizardDescriptionPage.java contains the methods to finding out object and performing action like setting values , clicking button etc. It contains reusable methods.

ServerTaskTest

```java
public void testAdvanceScheduleServerTask() throws Exception {
    String strTask;
    String cronJob = setCronAdvanceSchedule();
    String SCHEDULE_TYPE="Advanced";
    strTask="ST_AS_Task1";
    console.serverTaskLogSection().purge(0 ,"Days");

    String cronTask = getFullTaskName(strTask);
    console.serverTasksSection()
            .newServerTask()
            .name(cronTask)
            .next()
```

ServerTaskTest contains actual test method and inherites the method of ServerTaskWizardDescriptionPage to perfom actions on the page objects

Figure 8 – Shows sample selenium script written using Page Object model

4.2 Advantages of POM

- The Page Object Model says operation and flow in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
- A second benefit is the object repository is independent of test cases, so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG or JUnit for functional testing.
- Code becomes less and optimized because of the reusable page methods in the POM classes.
- Methods get more realistic names, which can be easily mapped with the operation happening in the UI.

5.0 Creating a framework for UI automation

A test automation framework is an integrated system that sets the rules of automation of a specific product. This system integrates the function libraries, test data sources, object details and various

reusable modules. These components act as small building blocks that need to be assembled to represent a business process. The framework provides the basis of test automation and simplifies the automation effort.

We are working on enterprise product, we wanted to have a framework to manage our automated test, customizable, providing results in better format, low maintenance, testing product on multiple browser, running it parallel to save the time. And the most important thing is this should allow us to integrate with the continuous integration environment.

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

Writing a test is typically three steps:

1) Write the business logic of your test and insert TestNG annotations in your code.
2) Add the information about your test (e.g. the class name, the groups you wish to run, etc...) in a TestNG.xml file or in build.xml.
3) Run TestNG.

There are many advantages available in TestNG. For example:

1) Annotations are easier to understand
2) Test cases can be grouped more easily
3) Parallel testing is possible
4) Multiple browser testing is possible.

As an example, we have created a build.xml file which contains many targets for cleaning, compiling and executing the tests. We can define a target for TestNG task.

1) Build.xml will have ant tasks for compiling test classes, associating required libraries, properties etc. It has an ANT task for TestNG which is responsible for invoking TestNG.xml file. You can invoke build.xml from command line by providing command line parameter. An example is shown here

ant -buildfile build.xml <target name to be executed on build.xml>

c:\>ant -buildfile C:\trunk\dev\src\MobileSSPSeleniumTest\build.xml TestNG-execution

2) TestNG.xml will have details about package, class or method to be executed. It will start executing methods available in class.

Figure 9 shows the flow between build.xml, TestNG and Test Class

Build.xml

```
<target name="testng-execution" depends="create,compile">
    <mkdir dir="${testng-report-dir}" />
    <testng outputdir="${testng-report-dir}" classpathref="classpath" useDefaultListeners="true">
        <xmlfileset dir="${basedir}" includes="testng.xml" />
    </testng>
</target>
```

TestNG.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Login Tests Suite">

    <listeners>
        <listener class-name="org.uncommons.reportng.HTMLReporter"/>
        <listener class-name="org.testng.reporters.EmailableReporter"/>
        <listener class-name="org.testng.reporters.XMLReporter"/>
        <listener class-name="org.testng.reporters.FailedReporter"/>
        <listener class-name="org.testng.reporters.SuiteHTMLReporter"/>
    </listeners>

    <test name="Mobile SSP successful login test">
        <classes>
            <class name="com.mcafee.mobilessptests.LoginTest"/>
        </classes>
    </test>
</suite>
```

LoginTest.java

```
package com.mcafee.mobilessptests;

import org.testng.annotations.Test;

public class LoginTest {


    @Test
    public void LoginTestMobileUser()throws Exception{

        //Test Code
    }




}
```

Figure 9 shows the flow between build.xml, TestNG and Test Class.

We can configure and execute the Selenium test from Teamcity by providing the build.xml file location and target name to be executed. This can be executed on a nightly basis or on each check-in based on your configuration

Detailed workflow of tests execution from Teamcity environment

1. Provide ANT target to be executed on Teamcity build step
2. Once Teamcity execute this target based on your trigger details ; it internally calls TestNG.xml which has details of Test Suite to be executed.
3. It opens up the browser and start executing the tests

Figure 10-Shows configuring selenium tests execution from continuous integration environment.

Figure 11 shows the architecture of our automation framework. It starts with calling ant target on build.xml which internally calls TestNG. TestNG invokes the test suites, which are classes or methods. Test Script imports the Selenium build in library and creates drivers for different browsers and starts the actual test on browser. The framework contains build.properties which contains environment details like the application URL, application login credentials, thread count, and browser details. Once the execution finishes, TestNG has a listener which provides the results in HTML format.

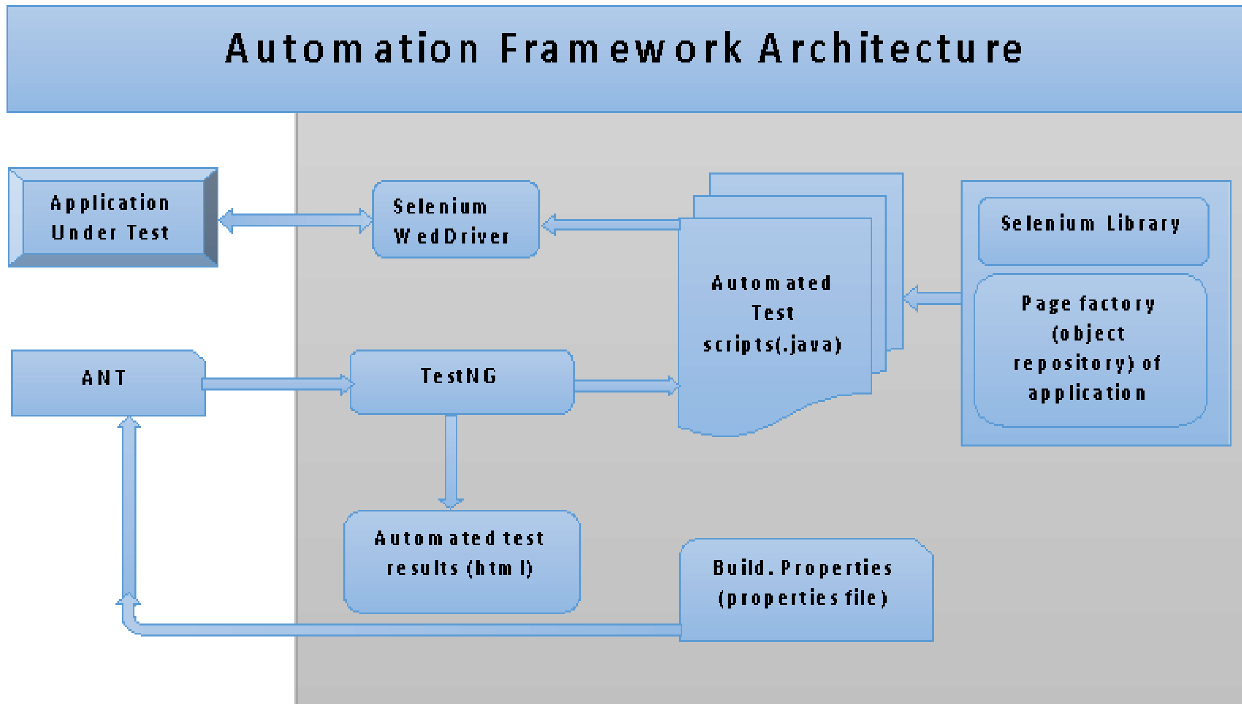

Figure 11 – Architecture diagram of automation framework

6.0 Continuous Integration

Selenium test suite can be integrated with the Teamcity environment. Teamcity has provisions to execute ANT targets as a part of build step. We can include TestNG ANT tasks in the build.xml ANT targets and execute classes and methods.

These are the components used to execute tests in a continuous integration environment

TestNG – We have used TestNG to define test suite, groups and tests to be executed. We can also define number of thread for concurrent testing.

ANT – Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. We have used it to compile test classes, associating Selenium library, page repository library and calling TestNG.

PowerShell- McAfee Foundation Service (the application under test) supports silent installation from command line. To automate this process we executed same command through PowerShell on remote machine. We have used PowerShell to create test environment that includes creating/reverting VM and installing application on the VM.

WebDriver- WebDriver invokes browser and perform various actions, assertion on the browser element. It can be IE driver, Firefox, chrome and any other driver.

Properties file- Property files are used to define the global properties to be used by application, browser property.

The team wanted to run the entire test suite on each check-in and on nightly builds as well. Selenium test suite can be easily integrated in Teamcity and can be run as a part of nightly build. *Figure 12* shows how Teamcity was used to install MFS and invoke Selenium tests to test it.

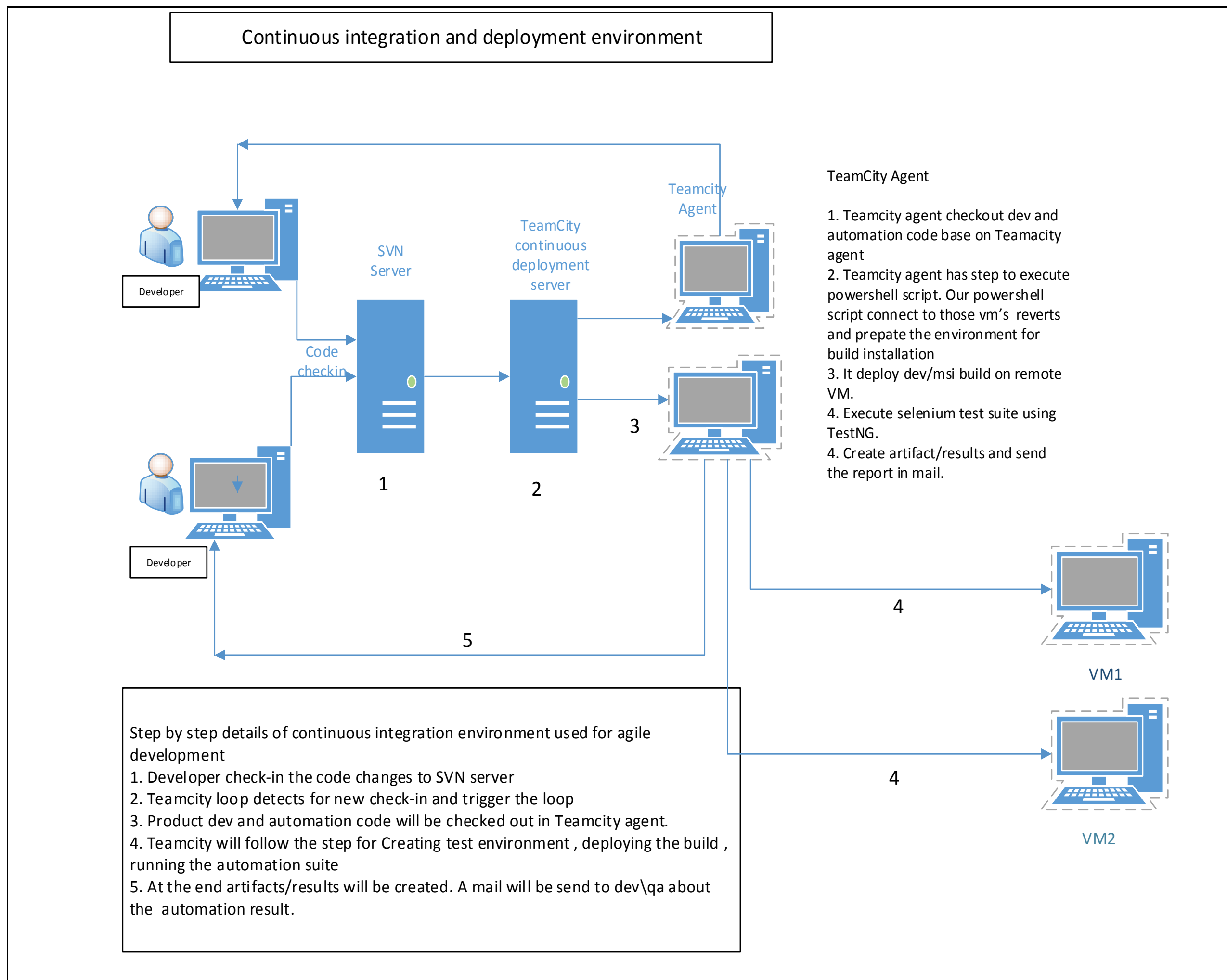

Figure 12 – Shows complete E2E flow starting from code check-in to selenium test execution and result generation

Step1 When a team member checks in code, Teamcity has been configured in such a way that it invokes the Selenium loop.

Step2 It checks out the application build and automation scripts to the Teamcity agent folder

Step3 It starts executing build steps mentioned in Teamcity and as a part of the build it starts executing build.xml.

Step4 Build steps can include automated environment creation tasks such as, reverting the virtual machine (VM) to a clean state, deploying MFS to the VM and starting TestNG.xml to execute tests.

Step5 Once the test is completed, it generates the artifacts and results in Teamcity.

7.0 Conclusion

This paper shows how TestNG and Selenium can be used for your automation need in agile development environment. It also explains you to configure TestNG for executing tests in parallel, with different browser and grouped based on your requirement. This paper also provides details about how easily selenium test suite and TestNG can be integrated with Teamcity continuous integration environment to execute tests on a nightly basis. Developer and team will easily know if someone had checked-in the code and broken the functionality by looking at the Teamcity automation result.

When teams are working across the globe, code check-in happens on a frequent basis, having an automated regression suite and running it on frequent basis discovers the defects earlier. This helps to improve the quality of the product.

8.0 References

Books

[1] Avasarala, S. (2014). *Selenium WebDriver practical guide interactively automate web applications using Selenium WebDriver*. Birmingham, UK: Packt Pub.

[2] Manoj Mahalingam S. *Learning continuous integration with TeamCity master the principles and practices behind continuous integration by setting it up for different technology stacks using TeamCity*. (2014). Birmingham, UK: Packt Pub.

Blogs

[1] "TestNG." *TestNG*. Web. 6 Aug. 2015.

[2] "Browser Automation." *Selenium WebDriver — Selenium Documentation*. Web. 6 Aug. 2015.

[3] "Selenium Easy." *Selenium Easy*. Web. 6 Aug. 2015

Copies may not be made or distributed for commercial use