IJCSMS International Journal of Computer Science & Management Studies, Vol. 12, Issue 03, Sept 2012
ISSN (Online): 2231 –5268
www.ijcsms.com

How Automated Testing Tools Are Showing Its Impact In The Field Of Software Testing

Deepti Gaur[1], Dr. Rajender Singh Chhillar [2]

[1]M.tech Student, Department Of Computer Science and Application,

M.D University, Rohtak-124001, Haryana, India

deeptigaur50@gmail.com

[2]Professor, Department Of Computer Science and Application,

M.D University, Rohtak-124001, Haryana, India

Chhillar02@gmail.com

Abstract

As, we know that Software testing is a very vast field in Software development life cycle. In this paper, we describe that how automated testing tools are very much convenient and easy to use which also makes testing faster and more effective in less time. Actually the world of technology revolves at fast pace today and among all Testing tools only automated testing tools makes Software testing more significant and effective.

Keywords: *Selenium RC, Egg plant, Eiffel studio, Regression testing.*

1. Introduction

Testing:-

A Process of evaluating a particular product to determine whether the product contain any defects.

Software Testing:-

Software testing is a process of evaluating a system by manual or automatic means and verify that it satisfied specified requirements or identify difference between expected and actualresult.
Testing process comes from software development life cycle.

The software process actually consists four parts:-
(a) Plan- Device a plan. Define your objective and determine the strategy and supporting methods required to achieve that objective.
(b) DO- Execute the plan. Create the conditions and perform the necessary training to execute the plan.
(c)CHECK : Check the results. Check to determine whether work is progressing according to the plan and whether the results are obtained.
(d)ACTION : Take the necessary and appropriate action if checkup reveals that the work is not being performed according to plan or not as anticipated.
Actually Testing have different strategies such as –Manual testing and Automated testing , Due to which testing is done by manual tools and Automated tools usually. Testing tools a lot of them are Automated because it makes the process of software testing faster and convenient. The world of technology revolves at a fast pace today, with more due extension the things are advanced. This is very hard of software engineers to use the manual tools, because testing take a lot of time, efforts and resources. Automate software testing tools functions as Virtual Testers. In software testing, test automation is the use of special software (separate from the software being tested) to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.[1]Commonly, test automation involves automating a manual process already in place that uses a formalized testing process. Test automation is the process of writing a computer program to do testing that would otherwise need to be done manually. Once tests have been automated, they can be run quickly and repeatedly. This is often the most cost effective method for software products that have a long maintenance life, because even minor patches over the lifetime of the application can cause features to break which were working at an earlier point in time.

1.1 Meaning Of Automated Testing:

Automatic testing speeds up the process of testing. One reason automatic testing is important is it ensures that software is reliable, especially when updates are made. To assist in this concept, a software testing workbench is used, which is an integrated set of tools to support the testing process [12]. Another way to ensure software reliability is using test redundancy detection. This reduces test maintenance costs and also ensures the integrity of test suites [11]. It has been shown that once test cases are written, it is a good idea to update the test cases when changes are made to the software. If test suites being updated (test maintenance) are not conducted carefully, the integrity of the test suites being used will be questioned [11]. Steps in a test suite can be almost completely automated using existing tools like JUnit and MuJava and two new tools to generate compatible test cases [10]. Research has found that an experiment was done where the suitability of the tool for generating high-quality test cases was checked [10]. The tool used two groups of students with the same academic level, some of whom used the java tool, while others wrote test cases manually. In the experiment, the students using the tool obtained much better results than the students who did not [10].
Automated testing, in which Quality Assurance teams use software tools to run detailed, repetitive, and data-intensive tests automatically, helps teams improve software quality and make the most of their always-limited testing resources.

Automated testing helps teams test faster, allows them to test substantially more code, improves test accuracy, and frees up QA engineers so they can focus on tests that require manual attention and their unique human skills.

Use these best practices to ensure that your testing is successful and you get the maximum return on investment (ROI).

1. Decide what Test Cases to Automate
2. Test Early and Test Often
3. Select the Right Automated Testing Tool
4. Divide your Automated Testing Efforts
5. Create Good, Quality Test Data
6. Create Automated Tests that are Resistant to Changes in the UI.

1. Decide What Test Cases to Automate

It is impossible to automate all testing, the first step to successful automation is to determine what test cases should be automated first. The benefit of automated testing is correlated with how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are those that are run frequently and require large amounts of data to perform the same action. You can get the most benefit out of your automated testing efforts by automating:

- Repetitive tests that run for multiple builds
- Tests that are highly subject to human error
- Tests that require multiple data sets
- Frequently-used functionality that introduces high risk conditions
- Tests that are impossible to perform manually
- Tests that run on several different hardware or software platforms and configurations
- Tests that take a lot of effort and time when doing manual testing

Success in test automation requires careful planning and design work. Start out by creating an automation plan. This plan allows you to identify the initial set of tests to automate, and serve as a guide for future tests. First, you should define your goal for automated testing and determine which types of tests to automate. There are a few different types of testing, and each has its place in the testing process. For instance, unit testing is used to test a small part of the intended application. Load testing is performed when you need to know how a web service responds under a heavy workload. To test a certain piece of the application's UI, you would use functional or GUI testing.

After determining your goal and which types of tests to automate, you should decide what actions your automated tests will perform. Don't just create test steps that test various aspects of the application's behavior at one time. Large, complex automated tests are difficult to edit and debug. It is best to divide your tests into several logical, smaller tests. This structure makes your test environment more coherent and manageable and allows you to share test code, test data and processes. You will get more opportunities to update your automated tests just by adding small tests that address new functionality. Test the functionality of your application as you add it, rather than waiting until the whole feature is implemented.

When creating tests, try to keep them small and focused on one objective. For example, use separate tests for read-only versus read/write tests. This separation allows you to use these individual tests repeatedly without including them in every automated test.

Once you create several simple automated tests, you can group your tests into one, larger automated test. You can organize automated tests by the application's functional area, major/minor division in the application, common functions or a base set of test data. If an automated test refers to other tests, you may need to create a test tree, where you can run tests in a specific order.

2. Test Early and Test Often

To get the most out of your automated testing, testing should be started as early as possible in the development cycle and run as often as needed. The earlier testers get involved in the life cycle of the project the better, and the more you test, the more bugs you find. You can implement automated unit testing on day one and then you can gradually build your automated test suite. Bugs detected early are a lot cheaper to fix than those discovered later in production or deployment.

3. Select the Right Automated Testing Tool

Selecting an automated testing tool is essential for test automation. There are a lot of automated testing tools on the market, and it is important to choose the tool that best suits your overall requirements.

Consider these key points when selecting an automated testing tool:

- Support for your platforms and technology. Are you testing .Net, C# or WPF applications and on what operating systems?
- Flexibility for testers of all skill levels. Can your QA department write automated test scripts or is there a need for keyword testing?
- Feature-rich but also easy to create automated tests. Does the automated testing tool support record-and-playback test creation as well as manual creation of automated tests? Does it include features for implementing checkpoints to verify values, databases, or key functionality of your application?
- Create automated tests that are reusable, maintainable and resistant to changes in the applications UI. Will your automated tests break if your UI changes?

4. Divide Your Automated Testing Efforts

Usually, the creation of different tests is based on the skill level of the QA engineers. It is important to identify the level of experience and skills for each of your team members and divide your automated testing efforts accordingly. For instance, writing automated test scripts requires expert knowledge of scripting languages. Thus, in order to perform that task, you should have QA engineers that know the script language provided by the automated testing tool.

Some team members may not be versed in writing automated test scripts. These QA engineers may be better at writing test cases. It is better when an automated testing tool has a way to create automated tests that does not require an in-depth knowledge of scripting languages, like Test Completes "keyword tests" capability. A keyword test (also known as keyword-driven testing) is a simple series of keywords with a specified action. With keyword tests, you can simulate keystrokes, click buttons, select menu items, call object methods and properties, and do a lot more. Keyword tests are often seen as an alternative to automated test scripts. Unlike scripts, they can be easily used by technical and non-technical users and allow users of all levels to create robust and powerful automated tests.

You should also collaborate on your automated testing project with other QA engineers in your department. Testing performed by a team is more effective for finding defects and the right automated testing tool should allow you to share your projects with several testers.

5. Create Good, Quality Test Data

Good test data is extremely useful for data-driven testing. The data that should be entered into input fields during an automated test is usually stored in an external file. This data might be read from a database or any other data source like text or XML files, Excel sheets, and database tables. A good automated testing tool actually understands the contents of the data files and iterates over the contents in the automated test. Using external data makes your automated tests reusable and easier to maintain. To add different testing scenarios, the data files can be easily extended with new data without needing to edit the actual automated test.
Creating test data for your automated tests is boring, but you should invest time and effort into creating data that is well structured. With good test data available, writing automated tests becomes a lot easier. The earlier you create good-quality data, the easier it is to extend existing automated tests along with the application's development.

6. Create Automated Tests that are Resistant to Changes in the UI

Automated tests created with scripts or keyword tests are dependent on the application under test. The user interface of the application may change between builds, especially in the early stages. These changes may affect the test results, or your automated tests may no longer work with future versions of the application.
The problem is that automated testing tools use a series of properties to identify and locate an object. Sometimes a testing tool relies on location coordinates to find the object. For instance, if the control caption or its location has changed, the automated test will no longer be able to find the object when it runs and will fail. To run the automated test successfully, you may need to replace old names with new ones in the entire project, before running the test against the new version of the application. However, if you provide unique names for your controls, it makes your automated tests resistant to these UI changes and ensures that your automated tests work without having to make changes to the test itself. This best practice also prevents the automated testing tool from relying on location coordinates to find the control, which is less stable and breaks easily.

There are two general approaches to test automation: Code-driven testing. The public (usually) interfaces to classes, modules or libraries are tested with a variety of input arguments to validate that the results that are returned are correct.
Graphical user interface A testing framework generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.
Test automation tools can be expensive, and are usually employed in combination with manual testing. Test automation can be made cost-effective in the long term, especially when used repeatedly in regression testing.
In Automation Testing the test Engineer or Software Quality Assurance person should have coding knowledge as they have to write down the test cases in form of code which when run and give output according to checkpoint inserted in it. Checkpoint is the point which is inserted to check any scenario.
One way to generate test cases automatically is model-based testing through use of a model of the system for test case generation but research continues into a variety of alternative methodologies for doing so In some cases, the model-based approach enables non-technical users to create automated business test cases in plain English so that no programming of any kind is needed in order to configure them for multiple operating systems, browsers, and smart devices.[2]
What to automate, when to automate, or even whether one really needs automation are crucial decisions which the testing (or development) team must make. Selecting the correct features of the product for automation largely determines the success of the automation. Automating unstable features or features that are undergoing changes should be avoided. [3]

1.2 Automation framework and a testing tool

Tools are specifically designed to target some particular test environment. Such as: Windows automation tool, web automation tool etc. It serves as driving agent for an automation process. However, automation framework is not a tool to perform some specific task, but is an infrastructure that provides the solution where different tools can plug itself and do their job in a unified manner. Hence providing a common platform to the automation engineer doing their job. There are various types of frameworks. They are categorized on the basis of the automation component they leverage. These are:

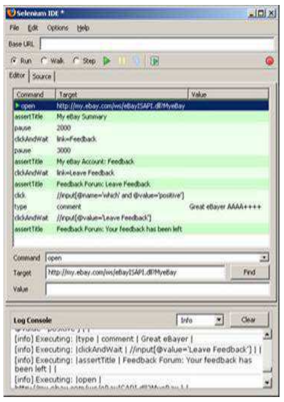
1. Data-driven testing
2. Modularity-driven testing
3. Keyword-driven testing
4. Hybrid testing
5. Model-based testing

Tool name	Produced by	Latest version
Selenium	Open source	2.20
Egg plant	Test plant	2012
Silk test	Micro focus	2011
Eiffel studio	Eiffel software	7.1 June 2012
Rational Robert	IBM Rational	2003

Selenium:-Selenium is an open source, robust set of tools that supports rapid development of test automation for web-based applications. This tool is primarily developed in Java Script and browser technologies and hence supports all the major browsers on all the platforms. Most of the time, we will not need to change our scripts for them to work on other platforms. Selenium provides a record/playback tool for authoring tests without learning a test scripting language.

Selenium Components There are three variants of Selenium, which can be used in isolation or in combination to create complete automation suite for the web applications. Each one has a specific role in aiding the development of web application test automation. Selenium IDE Selenium Core Selenium Rc.

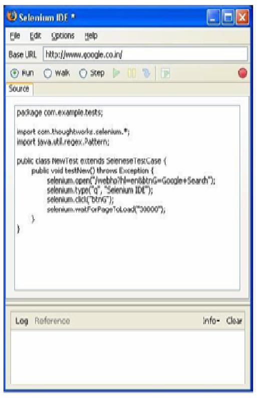
How to Use Selenium:-Select "Selenium IDE" from the Tools menu in Firefox. By default when the IDE comes up it has recording turned on. Go to a web site that you want to record, click on the record button and begin the browsing task(s). we will notice that as we click and type in the browser the IDE is recording everything we are doing. When we are done, simply end recording and click on the green arrow to play back the script you just created. Don't forget to save your script before closing the IDE . Below are the screen shots of working of Selenium IDE and the script generated by IDE.



Command	Target	Value
open	http://my.ebay.com/ws/eBayISAPI.dll?MyeBay	
assertTitle	My eBay Summary	
pause	2000	
clickAndWait	link=Feedback	
pause	3000	
assertTitle	My eBay Account: Feedback	
clickAndWait	link=Leave Feedback	
assertTitle	Feedback Forum: Leave Feedback	
click	//input[@name='which' and @value='positive']	
type	comment	Great eBayer AAAA++++
clickAndWait	//input[@value='Leave Feedback']	
assertTitle	Feedback Forum: Your feedback has been left	

Command: open
Target: http://my.ebay.com/ws/eBayISAPI.dll?MyeBay

```
[info] Executing: |type | comment | Great eBayer |
[info] Executing: |clickAndWait | //input[@value='Leave Feedback'] | |
[info] Executing: |assertTitle | Feedback Forum: Your feedback has been left | |
[info] Executing: |open |
```



Base URL: http://www.google.co.in/

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class NewTest extends SeleneseTestCase {
        public void testNew() throws Exception {
                selenium.open("/webhp?hl=en&btnG=Google+Search");
                selenium.type("q", "Selenium IDE");
                selenium.click("btnG");
                selenium.waitForPageToLoad("30000");
        }
}
```

Selenium Core:- Selenium Core is a test tool for web applications. Selenium Core tests run directly in a browser, just as real users do. They run in Internet Explorer, Mozilla and Firefox on Windows, Linux and Macintosh. It is a simpler form of Selenium, and suitable for non-developers. Browser compatibility testing: To test the application if it works correctly on different browsers and operating systems. The same script can run on any Selenium platform. System functional testing: Create regression tests to verify application functionality and user acceptance. **Selenium Remote Control :-** Selenium-RC is the solution for tests that need more than simple browser actions and linear execution. We can use Selenium-RC whenever our test requires logic which is not supported by Selenium-IDE. Selenium-RC uses the full power of programming languages to create more complex tests like reading and writing files, querying a database, and emailing test results.

Java example using Selenium

public void setUp() throws Exception

{

selenium = new DefaultSelenium(

"localhost", 4444, "*chrome",

"http://www.google.com");

selenium.start();

}

public void tearDown() throws Exception

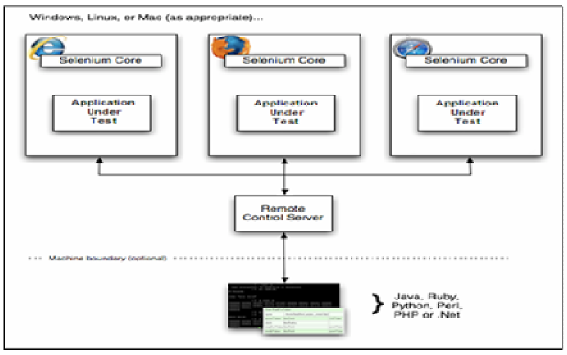
{

selenium.stop();

}

Features

- We can use Java syntax to write test script
- We can read files to get test data



References

[1] olawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press. p. 74. ISBN 0-470-04212-5.

[2] "Proceedings from the 5th International Conference on Software Testing and Validation (ICST). Software Competence Center Hagenberg. "Test Design: Lessons Learned and Practical Implications.". Brian Marick. "When Should a Test Be Automated?". StickyMinds.com. Retrieved 2009-08-20.

[3] F Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[4] Sannella, M. J. 1994 Constraint Satisfaction and Debugging for Interactive User Interfaces. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[5] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3 (Mar. 2003), 1289-1305.

[6] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[7] Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Journal of Systems and Software, 2005, in press.

[8] Spector, A. Z. 1989. Achieving application requirements. In Distributed Systems, S. Mullender

[9] Macario Polo, Sergio Tendero, Mario Piattini, Integrating techniques and tools for testing automation, EBSCO host database, ISSN# 09600833, Wiley Publishers, March 2007.

[10] N. Koochakzadeh, V. Garousi, A Tester-Assisted Methodology for Test Redundancy Detection, Advances in Software Engineering, Hindawi Publishing Corporation, V 2010, Article ID 932686, 2009, URL: http://www.hindawi.com/journals/ase/2010/932686/

[11] Ian Sommerville, Software Engineering, Eighth Addition, 2007.