28TH ANNUAL
PACIFIC NW
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010

ACHIEVING QUALITY IN A COMPLEX ENVIRONMENT

Conference Paper Excerpt from the

CONFERENCE PROCEEDINGS

Streamlining test automation through white-box testing driven approach

Sushil Karwa & Sasmita Panda
McAfee, Inc. Bangalore, India
sushil.karwa@gmail.com, sasmita@gmail.com

Abstract

Software test automation has been perceived as something that can cater to our Regression or Smoke tests and if time permits, to Functional tests. How often have you found yourself in a situation where you want “something extra” that could complement your black box test automation efforts? In this paper, we propose to bring innovation to this age old approach of test automation which mostly relies on black box testing approaches. This paper does not insist on replacing black box automation, rather it proposes to complement it with white box test automation. We will talk about various white box automation ideas that can be implemented across the different stages of software development life cycle to improve the overall software quality.

This paper explores how white box test automation can fill in the gaps in your overall test automation strategy. We will also discuss how a test automation strategy that includes both white box and black box automation techniques can streamline the achievement of more effective and efficient testing. Finally, we will share our lessons learned while implementing white box test automation.

About the Authors

Sushil Karwa *is a Sr. QA Project Lead at McAfee. He has been associated with McAfee since last 7 years. At McAfee, he is responsible for managing the end to end Quality goals for the Security Management Server product. His passion towards improving product quality through white box testing saw him initiating and implementing the process of white box test automation for various product teams at McAfee. He also leads certain specific initiatives w.r.t security testing efforts at McAfee.*

Sushil has a MS in Quality Management from BITS Pilani University, India. He is a Certified Ethical Hacker and Security Analyst from EC-Council.

Sasmita Panda *is a Sr. Software QA Engineer at McAfee and works with Host Intrusion Prevention System product team. Her testing and quality assurance experience includes a focus on white box testing automation for a client-server based application. Her current role also involves identification of different techniques for measuring code coverage. In her present role, she also performs security testing with a focus to uncover security loopholes in the product at various phases of SDLC.*

Sasmita has a Master in Computer Application from Madras University, India. In past she had also worked as software developer and is very passionate about improving software quality and promoting quality achievement throughout the organization.

1. Introduction

In today's complex software environment test automation has not only become a strategic necessity but also critically vital to improve the overall product quality. The QA fraternity has started to view test automation as one of the most desired methodology for testing due to its quite evident benefits – cost saving, time saving and reliability.

Although there has to be some deterministic set of rules or guidelines on when and how to use automation, at times this is more of a decision that is influenced by the use of "common sense". If we know that a test is not going to be run repetitively, this really shouldn't be a part of the automation test suite. But what if these repetitive tests are too expensive to automate through black box automation? Do we drop them out? Or do we give it a thought that it might just be an "API call" away if it were automated using white box test approach?

In this paper we're going to talk about using various white box test automation techniques that can be used along with black box test automation.

This paper will consist of following:

- A typical test automation strategy
- Most commonly followed approach for test automation
- White box test automation and how it offers more value to your overall test automation
- Challenges faced while implementing white box test automation

2. Are we balanced?

Testing is all about trade-offs. Testing as an activity has evolved over a period of time. Today we are spoiled with choices when it comes to picking up different testing techniques. Lot of QA personnel today face this million dollar question – which testing types to use and which to avoid for your project? With so many testing types available, where we focus our time can be a non-trivial question to answer. Automation has proven its worth over several years and has answered a few of these questions. But even today, many of us find ourselves in a tricky situation to decide how much manual vs. automated testing is good enough.

Some would say we have a goal to automate n% of our manual tests. But what's the point of automating even 100% of your tests if those tests are not covering your product from all angles. We still might be unbalanced when it comes to test automation. The reason: many of us do not do any automated white box testing. Even if many of us appreciate white box testing because of the value it offers, it may still be missing from many of our testing toolboxes. We will discuss in this paper how automated white box testing can balance our overall test automation efforts.

3. Typical Test Automation Strategy

Test automation has always been a daunting and demanding task. So how do we see quick returns from test automation? Before getting into details of white box test automation, let's have a quick look of what an ideal test automation strategy looks like. There are various aspects about test automation one should balance in order to come up with an effective test automation strategy. Discussed below are few of these aspects.

3.1. Identifying your test automation goal

The first thing before starting test automation is to identify your test automation goals. Your test automation strategy would evolve depending on what you want to achieve out of test automation. Some of the typical automation goals include

- Efficiency – Regression tests, reducing test time
    - Ex: Automating a routine task like looping through various input types, automating smoke testing.
- Increased Test Coverage – Enhancing the test coverage by automating tests
    - Ex: API testing
- Multi-platform coverage – Testing the product on multiple platforms in an automated way.
    - Ex: Automating installer testing.

3.2. Test Automation Prioritization during Product life cycle

Once the test automation goal is set, we need to prioritize the tests that need to be automated. Prioritization of tests can depend on certain driving factors like:

- Repeatability – Tests that will be run repeatedly
- Complexity of test scenarios – workflows, real world scenarios, etc.
- Nature of the tests – monotonous, uninteresting, etc.
- Sometimes we prioritize new test cases over automated regression ones.

3.3 Setting realistic expectations

Test automation cannot and should not be treated as one of the main sources for finding "new" bugs. Yes, test automation will discover defects for you but not in large numbers.

Automating everything is another unrealistic expectation from test automation. We should not be looking at test automation to replace manual testing completely.

3.4. Defining measurable goals

Goals for test automation should be specific, realistic and achievable. Automating n% of the test cases for a particular feature area in your application shouldn't be the only means of measuring the automation goals.

Automating the Build Verification Tests (BVT) test suite is something that is quite achievable and realistic. Automating the complex and mundane tests could be another specific goal.

4. Test Automation – most commonly followed approach

While the obvious benefits of test automation are widely known, test automation is usually considered to be just black box test automation. Test automation has always promised to cut costs in terms of execution time and needed resources. If applied correctly, test automation benefits can include

- Reduced cost of testing – especially when you automate the repeatable tests
- Accelerated time to market – automating tasks that are truly on critical path to improving product delivery timeline can improve software delivery time
- Reduced cost of quality – test automation support the timeline decrease of the software testing lifecycle, facilitate defect identification, and thus help improve quality

Considering the benefits offered by test automation, it has been one of the most sought after types of testing in the software development cycle. Even before we have any User Interface to test, we could still write some automated black box tests to either remotely connect to any application and execute the tests or execute the functionality from a command line provided the application supports it. During the testing cycles we come across lots of repetitive and mundane tests in addition to other tests that are prone to errors if run manually. Automating these types of tests can save much manual test execution time which can be used for other productive tasks. Building up the automated test suite not only helps increasing the confidence in the product but also enhances the test coverage.

Typically in most of the organizations, black box testing is the most commonly used approach for test automation. It's a good way to start, but it has certain limitations. Some of them are listed below:

- Possibility of redundantly testing the same functionality
- Chances of uncovering new defects are less
- Lack of internal knowledge of the code, does not help in coming up with new functional tests
- Cannot test all APIs – inability to create special hooks in the code
- Limited testing for error conditions and paths
- The time to implement automated test cases

So what is the alternative? How can we overcome these limitations and still be able to increase our overall test automation? Because white box testing deals with the internal knowledge of the code, it can offer a lot of value and streamline our overall test automation efforts. White box test automation in a real sense can fill in those gaps in your test automation strategy. Automating white box tests not only complements the black box test automation but also helps uncovering altogether different categories of defects that are hard to find through black box testing. Let's review what the different white box testing techniques are and how they can be automated.

5. White box test automation techniques- value additions

5.1. API Testing

An Application Programming Interface (API) is a collection of software functions and procedures. API tests essentially takes into consideration the individual methods and functions that make up a software system. An API test will not necessarily test the overall system as a whole, but focuses on the individual building blocks of the system and ensures that they work correctly. But generally API tests are more extensive than the unit tests. They take into consideration most likely usage scenarios that the API will be used for. This typically involves interactions between several different blocks or modules within the software application.

In our project, we started white box test automation by developing API tests. We focussed on testing all the public APIs in our code and targeted for 100% test coverage on them. We followed some general guidelines while developing our automated API tests. Some of them were:

- Public API tests should only use the classes and interfaces that are part of public API

- Ensure to write tests for all "if" and "else" statements, and "try" and "catch" blocks.

- Use assertEquals() instead of assertTrue(), since it gives better feedback over what was expected when a failure occurs.

APIs are the building blocks of your software product and improving the quality of the software largely depends on how these APIs are tested. Testing all the APIs through black box testing types is not always feasible. For instance, certain error conditions are difficult to force while testing through black box approach alone. Combining certain testing techniques like equivalence classes, boundary analysis and forced error testing with API testing proves to be very effective.

Since API testing directly targets the code level, it is one of the important types of white box testing. In order to test a typical usage scenario of an API, testers end up writing test code or mini programs. Automating API testing by writing white box tests helps you to increase the confidence in your test coverage. Here are some of the areas that need to be considered while automating API testing.

5.1.1 Test the APIs by setting up the initial conditions required to invoke them:

- Some API requires a certain set of activities to be done before it can be called.
- These initial conditions, if set before invoking the API, can influence the behavior of that API.

- Example: A non-static member function API requires an object to be created before it could be called.

5.1.2 Test an API based on its declaration:

- Create your tests based on the nature of the API.
- For example: An API can be a standalone API i.e. not a member function or it could be static / non static member function. Depending on its declaration, the object has to be created to invoke the API.

5.1.3 Test an API based on its invocation:

- An API can be called directly.
- API can be invoked by certain events, for instance mouse click, mouse movement, etc.
- API is invoked when an exception occurs.

5.1.4 Test the outcome of the execution of API:

- API when invoked may have several outcomes.
- Some returns certain data or status.
- Some just does not return anything.
- Some modify certain resources.

5.2. Code Coverage

Code Coverage is a metric that can be used to determine how much code has been executed by your test suite. It helps in identifying the paths in your code that are not getting executed. Since black box tests are designed without any knowledge of internal implementation, there would always be some part of the systems that gets rarely tested or completely missed out.

In the following code snippet, achieving 100% statement coverage through black box testing may not be easy. In order to execute the code on line number 234 the “if” condition should be true and setting the value of type to null through black box testing might be little tedious. By writing a simple white box test, this type value can be easily set to “null”. This would evaluate the condition to true and thus the code on line 234 gets executed.

```
231    public EnterceptPolicy CreatePolicy(PolicyObject po, PolicyType type, Locale locale) {
232        // if the type is null, this is a non-starter
233        if (type == null) {
234            throw new NullPointerException("type must be non-null");
235        }
```

The best way of measuring the code coverage on a regular basis is to automate the process of running the code coverage tools and generating the reports. The following steps ensures that the code coverage results get generated automatically so that one can analyze the reports and add new tests to increase the overall test coverage:

In our project we automated the code coverage process through both black box and white box testing.

- Developed scripts to instrument the build delivered to black box team using EMMA code coverage tool.

- Ran black box tests (manual and automated).

- Generated code coverage report.

- Configured EMMA with our automated white box tests in IDE.

- Executed our white box tests and generated code coverage report.

- Automatically merged both the reports using EMMA.

- Analyzed the report and came up with new test scenarios to increase coverage

- Start developing white box tests early in the lifecycle.

- Use an appropriate code coverage tool and integrate it with your white box tests.

- Run the automated white box test suite on a regular basis (probably on every build)

- Automatically generate the code coverage report after the execution of the automated white box test suite. By automating this process you make sure that code coverage report always gets generated which can be analyzed later. Either write some batch scripts or add new targets in your build file to automatically generate the code coverage report upon execution of white box tests.

- Analyze the coverage report and create new test scenarios. Develop additional white box and black box tests for these test scenarios

- Based on your coverage goal, strive for adding new tests to cover the untested code and meet the coverage objective.

5.3 Code Analysis

Code analysis is an automated way of analyzing the components and resources of the application under test to identify situations that are likely to produce errors. Code analysis tools scan the code and look for potential problems based on some pre-defined rule sets. For instance the “unused code” rule set would contain a collection of rules that find the unused code in your application. Similarly the other most common rule sets would report errors like buffer overrun, uninitialized memory, null pointer dereferences and memory and resource leaks.

Some of the most prominent benefits of Code Analysis are:

- Finding probable bugs in the code.
- Locating the “dead” code.
- Detecting performance issues.
- Improving code structure and maintainability.
- Helps in conforming to coding guidelines and standards.
- Automated tools provide mitigation recommendations, reducing the research time.
- It allows a quicker turn around for fixes.

There are tools available in the market that can be integrated with your development environment. One can create automated targets for running code analysis tools on their product code and include them in the build file. So whenever the code is built, the code analysis target will also get automatically executed and the report gets generated. Some IDEs like Visual Studio come with an option to automatically perform the code analysis and generate the report every time the code is built.

After the report is generated the findings need to be analyzed manually and defects should be raised appropriately.

In our project we used a code analysis tool called PMD whose plugin is available for several IDEs.

- PMD supports creating targets in Java based build tool like Apache Ant.

- We created ANT targets to automatically run code analysis on individual modules of our code

- These ANT targets used to get executed every time we ran our white box tests.

- Code analysis reports were generated for every run automatically.

- These reports were then manually examined and defects were raised appropriately.

- We also automated the process of detecting any API violations in our code.

In addition to the pre-defined rule sets provided by the code analysis tools, one may want to add their custom rules to fulfil specific need. For instance public API tests should only use the classes and interfaces that are part of the public API with exceptions of some third party libraries that are used in the project. If this is not the case you should treat it as a case of API violation in your code. Creating a customized rule set to detect such problem and feeding it into the code analysis tool would automatically discover this problem in the source code.

5.4 Build Verification Testing

Build Verification testing (BVT, also known as smoke testing) is used to make a decision whether a build released to QA is in an acceptable state so that detailed functional testing can be carried out on that build. The BVT test suite is comprised of test cases that test the basic functionality of the product. If any of the tests fails from the test suite, the BVT fails. Minimizing human intervention in test execution is the key here.

BVT is executed every time a build is released to QA. Considering that it needs to be executed repetitively during the product life cycle, it is one of the best candidates for test automation. BVT can be a blend of both black box and white box tests. Black box tests can focus on the UI (User Interface) part and make sure that the UI conforms to the product usability specifications. White box tests can test the functionality by directly calling the relevant APIs.

Automating BVT requires setting up a unit test framework that can be used to make direct calls to the code. One can create a BVT level white box test suite that gets executed automatically whenever a build is released to QA. Based on the result of BVT white box test execution, the build can be straightaway rejected without any manual intervention required, thus saving lot of testing time during the project.

5.5 Security Testing

Security testing uncovers any security vulnerabilities in the application that can be exploited by malicious user. The main objective of security testing is to ensure that the application under tests remains robust even in the face of a malicious attack.

The functional specification might outline a secure design, the developers might be attentive and write secure code, but it's the testing process that determines whether the software is secure in the real world. Through white box testing one can validate the implemented security functionality and uncover the vulnerabilities that can be exploited.

Security testing can be very time consuming and may sometime go beyond the scope of overall testing cycle. One should follow a risk-based approach while performing security testing. Risks, once identified, should be ranked based on its severity, business impact and probability of occurrence and

prioritized accordingly. Testing efforts should be made to focus on high risk areas. In order to have optimum coverage within stipulated time, one should use both black box and white box testing techniques to carry out security testing on their product.

In our project we automated security testing by developing framework using Ruby and WATIR.

We created automated tests to pass in malicious scripts as inputs to our application.

We tested for vulnerabilities like Cross Site Scripting, SQL Injection, URL Manipulation, Information disclosure, etc.

We also developed automated tests to verify the implementation of security functionality in our product like encryption and data storage.

Around 15% of our security defects were found by our automated white box tests.

These automated white box tests were used for regression testing as well.

Automating security tests through white box not only saves time but also provides new avenues for performing security testing in other unexplored areas. One of the examples of how security tests can be automated is discussed below. Let's say your application interacts with the database and we know that improper input data validation can lead to SQL injection attacks. A white box test could be to access the object that holds the input data and provide a maliciously crafted SQL string to delete a table in the database. A security test would be to ensure that the table does not get deleted when this data is stored in the database. Similarly one can think of automating security tests to uncover coding or functionality errors to name a few as mentioned below.

- Improper input validation compromising security.
- Broken authentication
- Insecure cryptographic storage.
- Information disclosure vulnerability
- URL Tampering
- File manipulation
- Manipulating Sessions

6. Challenges

There are certain challenges one may face while implementing white box test automation. Some of these are discussed below.

Skill set requirement: In order to perform white box testing, a specific skill set is required which include basic understanding of a programming language. Other skills include familiarity in designing framework, working with tools, understanding various white box testing techniques, etc.

Expecting too much in the beginning: Understanding the code, developing the basic white box test automation framework, getting familiar with the tools used, ability to find defects in the vast sea of code are some of the reasons why it takes time to see any visible results at the beginning.

Not defining the scope of the automation: Without defining the scope of white box test automation upfront, one cannot achieve the desired results. Automating 100% should not be the goal. By automating we are not going to eliminate the manual testing completely.

Dependency on development team: There will be some kind of dependency on the development team to understand the workflow of the code and the way it's been implemented. Sometimes the design document may not provide you the information you are looking for.

Maintaining the energy: Maintaining the motivation and enthusiasm to do white box testing can be a big challenge. This should be carried out with self-interest rather than being forced by the management.

7. Lessons learned

We feel like we learned the following lessons from our experience of implementing white box test automation in our project.

- It's very important to use right tool and technology that suites your need while performing any of the white box testing techniques.
- Interaction with developers helped us to come up with good ideas for our white box test automation.
- Starting white box test automation from the early stages of the product development cycle enabled us to uncover defects in the code that could be fixed in cost effective way.
- Keep searching for new ways to improve the process and technologies you use to make things more efficient, productive and simpler.

8. Conclusion

We found white box testing can complement black box testing to increase overall test effectiveness. White box test automation can be a very powerful mechanism to test your application. It helps in uncovering programming and implementation errors very early in the life cycle. Employing the right white box testing techniques to effectively test your application can help increasing your confidence in your product significantly. By complementing the two forms of testing, more areas can be covered and new tests can be developed to increase the overall test coverage. As the experts say, the earlier a defect is found, cheaper it is to fix it. This is exactly where white box testing plays a big role to save money and automating it saves the testing time.

References

[1] Henk Coetzee, Volume 7, Number 2, March/April 2006, "Software Test Automation in the SDLC" - http://www.testfocus.co.za/

[2] Jussi Kasurinen, Ossi Taipale, and Kari Smolander, Volume 2010 (2010), Article ID 620836, "Software Test Automation in Practice: Empirical Observations" - http://www.hindawi.com/journals/ase/2010/620836.html

[3] James McCaffrey, "API Test Automation in .NET" - http://msdn.microsoft.com/en-us/magazine/cc163892.aspx

[4] Automated Software Testing Magazine, May 2009 Edition - www.automatedtestinginstitute.com

[5] List of rule sets used by code analysis tool PMD - http://pmd.sourceforge.net/rules/index.html