IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

Optimizing Whole Test Suite Generation

Dr.V.Sangeetha[1], T.Ramasundaram[2]

Assistant Professor, Department of Computer Science, PRUCAS, Pappireddipatti, Dharmapuri, Tamilnadu, India[1]

Assistant Professor, Department of computer Science, Sri Vijay Vidyalaya College of Arts & Science,

Nallampalli, Dharmapuri, Tamilnadu, India[2]

Abstract: Software testing is as old as the hills in the history of digital computers. The testing of software is an important means of assessing the software to determine its quality. Since testing typically consumes 40 - 50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering. Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. A common scenario in software testing is therefore that test data is generated, and a tester manually adds test oracles. As this is a difficult task, it is important to produce small yet representative test sets, and this representativeness is typically measured using code coverage. There is, however, a fundamental problem with the common approach of targeting one coverage goal at a time: Coverage goals are not independent, not equally difficult, and sometimes infeasible – the result of test generation is therefore dependent on the order of coverage goals and how many of them are feasible. To overcome this problem, whole test suites are evolved with the aim of covering all coverage goals at the same time, while keeping the total size as small as possible. However, the new paradigm works well; the test suites were not optimal. So, we propose a new tool with a novel paradigm to give optimality. This tool will have several advantages, as for example its effectiveness is not affected by the number of infeasible targets in the code. We are also described about the implementation of the tool using Eclipse IDE, java, Java Swing.

Keywords: Software Engineering, Software Testing, Infeasible goal, Testing Techniques, Automated Testing Tools.

I. INTRODUCTION

The quality of a software product can be checked or evaluated based on the testing procedures that the product or software undergone. Basically testing [15] is an ongoing activity that is related with any process to produce a quality or working product.

According to IEEE, Testing is the process of evaluating a system of system component by manual or automated means to verify that it satisfies required requirements [15]. So it is used to check the status of the working product after and during the software build. Software testing is also used to detect and identify the defects that the software may have. It is one of the vital parts of software development life cycle (SDLC).

Software testing can be done either by using automated or manual testing. By testing software through automated means is the best way to test the software. This testing of software is useful when repeated test scripts [15] are required or where the test scripts subroutine are generated.

The one of the most important advantage of automation testing is its execution speed. On the other hand, manual testing requires testing manually which needs more time, more chances of error and is no more useful.

Hence all issues of manual testing can be fixed using automation testing. This paper demonstrates the taxonomy of different types of testing techniques and different automated testing tools comprising of Functional, Management and Loading testing.

II. CHARACTERISTICS OF SOFTWARE TESTING

A. Validity:
A test is considered as valid when it measures what it is supposed to measure.

B. Reliability:
A test is considered reliable if it is taken again by the same students under the same circumstances and the score average is almost the constant, taking into consideration that the time between the test and the retest is of reasonable length.

C. Objectivity:
Objectivity means that if the test is marked by different people, the score will be the same. In other words, marking process should not be affected by the marking person's personality.

D. Comprehensiveness:
A good test should include items from different areas of material assigned for the test. e.g., (dialogue - composition - comprehension - grammar - vocabulary - orthography - dictation - handwriting)

E. Simplicity:
Simplicity means that the test should be written in a clear, correct and simple language, it is important to keep the method of testing as simple as possible while still testing the skill you intend to test. (Avoid ambiguous questions and ambiguous instructions).

F. Scorability:
Scorability means that each item in the test has its own

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

mark related to the distribution of marks.

III. OBJECTIVE OF TESTING

The objective of testing is to find problems and fix them to improve quality (Fig.1). Software testing typically represents 40% of a software development budget.
There are four main objectives of software testing:
Demonstration: It demonstrates functions under special conditions and shows that products are ready for integration or use.
Detection: It discovers defects, errors and deficiencies. It determines system capabilities and limitations, quality of components, work products and the system.
Prevention: It provides information to prevent or reduce the number of errors clarify system specifications and performance. Identify ways to avoid risk and problems in the future.
Improving Quality: By doing effective testing, we can minimize errors and hence improve the quality of software. [2]

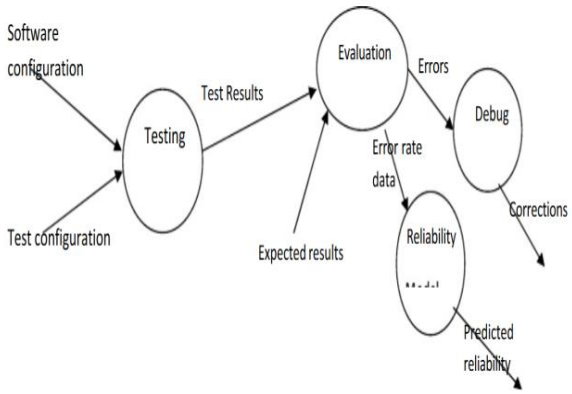

Fig.1 Test Information Flow

IV. DIFFERENT TESTING TECHNIQUES

A. Black Box Testing
Black Box Testing is based on the requirements specifications and there is no need to examining the code in black box testing. This is purely done based on customers view point only tester knows the set of inputs and predictable outputs. [6][3]

Advantages:
- Testers need not to have knowledge on specific programming language.
- Testing is done from user's point of view.
- It helps to expose any ambiguities or inconsistencies in the requirement specifications.[4]
- Programmer and tester both are independent of each other.

Disadvantages:
- Test cases are hard to design without clear specifications.
- Some parts of back end are not tested at all.
- Chances of having repetition of tests that are already done by programmer.

B. White Box Testing
White box testing mainly focuses on internal logic and structure of the code. White-box is done when the programmer has techniques full knowledge on the program structure. With this technique it is possible to test every branch and decision in the program.[2]

Advantages:
- It reveals error in hidden code by removing extra lines of code.
- Maximum coverage is attained during test scenario writing.[5]
- Developer carefully gives reasons about implementation.

Disadvantages:
- A skilled tester is needed to carry out this testing because knowledge of internal structure is required.
- Many paths will remain untested as it is very difficult to look into every nook and corner to find out hidden errors.

C. Grey Box Testing:
Gray-box testing attempts, and generally succeeds, to combine the benefits of both black-box and white-box testing. Gray-box testing takes the straight-forward approach of black-box testing, but also employs some limited knowledge of the inner workings of the application.
White box + Black box = Grey box, it is a technique to test the application with limited knowledge of the internal working of an application and also has the knowledge of fundamental aspects of the system. [5] Therefore, a tester can verify both the output of the user interface and also the process that leads to that user interface output. Gray-box testing can be applied to most testing phases; however it is mostly used in integration testing.

Advantages:
- It provides combined benefit of black box and white box testing techniques.
- In grey box testing, tester can design excellent test scenarios.
- Unbiased testing
- Create an intelligent test authoring.

Disadvantages:
- Test coverage is limited as the access to source code is not available.
- Many program paths remain untested. 3. The test cases can be redundant.[5]

TABLE I COMPARISON BETWEEN THREE FORMS OF TESTING TECHNIQUES

S. No	Black Box Testing	White Box Testing	Grey Box Testing
1	Analyses fundamental aspects only i.e.no knowledge of internal working.	Full knowledge of internal working.	Partial knowledge of internal working.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 1, January 2016

2	It is least exhaustive and time consuming.	Potentially most exhaustive and time consuming	It is somewhere in between the two.
3	Not suited for algorithm testing.	It is suited for algorithm testing (suited for all).	Not suited for algorithm testing.
4	Granularity is low.	Granularity is high.	Granularity is medium.
5	Performed by end users and also by tester and developers (user acceptance testing).	It is performed by developers and testers.	Performed by end users and also by tester and developers (user acceptance testing).

V. COVERAGE CRITERIA

A. Types of Coverage Criteria

Automatic unit testing is guided by a structural coverage criterion. There exist many coverage criteria in literature, each of them aims at covering different components of a CUT. Below is a list of coverage criteria for structural testing for Java programs.

1. Line Coverage: the goal is to execute all the lines (non-comments line) of the CUT. It is perhaps the most used criterion in practice.
2. Branch Coverage: the goal is to cover all the branches of the CUT. Covering a branch means executing the branch at least twice by taking once the true branch and twice the false branch.
3. Modified Condition Decision Coverage: [16] the goal is that every condition within a decision has taken on all possible outcomes at least once, and every condition has been shown to independently affect the decision's outcome (the condition of interest cannot be masked out by the other conditions in the decision). This criterion is stronger than branch coverage.
4. Mutation: the goal is to kill all the mutants. Killing a mutant means that the output of a test case on the program is different from the output of the test case on the mutant.
5. Weak Mutation: the goal is to weakly kill all the mutants. Weakly killing a mutant means that the internal state of the program immediately after execution of the mutated component must be incorrect [17]. This criterion is weaker than Mutation.
6. Method coverage: the goal is to call all the methods of the CUT by executing the test suite.
7. Top-level Method Coverage: the goal is to call all the methods of the CUT directly which means that a call to the method appears as a statement in a test case of the test suite. This criterion is stronger than Method Coverage.
8. No-Exception Top Level Method Coverage: the goal is to call all the methods from the test suite via direct invocations, but with parameters that lead to a normal execution of the methods (not generating exceptions). This criterion is stronger than Top-level Method Coverage.
9. Direct Branch Coverage: the goal is that each branch in a public method of the CUT to be covered by a direct call from a unit test, but makes no restriction on branches in private methods.
10. Output Coverage: the goal is to call all the methods with parameters that cover all the different types of output the method can return. E.g. if the method's type is Boolean the method should be called twice in order to return once a true value and once a false value.
11. Exception Coverage: the goal is to cover all the feasible undeclared exceptions (if exceptions are unintended or if thrown in the body of external methods called by the CUT).

Exception Coverage, Method Coverage, Top-level Method Coverage, No-exception Top-level Method Coverage have a fitness function which provides no guidance during the search [1]. Mutation criterion is considered the gold criterion in research literature [19]. This criterion is difficult to apply and computationally expensive and it is practically only used for predicting suite quality by researchers. The reasons why mutation testing cannot be used for testing real software are:

1. The number of mutants for a given system can be huge and it is very expensive to run the test against all the mutants.
2. Equivalent mutants which are mutants that only change the program's syntax, but not its semantics and thus are undetectable by any test

The criteria implemented in EvoSuite are [1]: Line Coverage, Branch Coverage, Direct Branch Coverage, Output Coverage, Weak Mutation, Exception Coverage, Top-level Method Coverage, No-exception Top-level Method Coverage. The most used criterion is branch coverage [18] but even though it is an established default criterion in the literature, it may produce weak test sets, and software engineering research has considered many other criteria. Another option is to combine different coverage criteria.

B. Combination of Coverage Criteria

The search is guided by the coverage criteria the resulting test suite should satisfy. The coverage criteria are translated to a mathematical formula which is the fitness function whose work is to evaluate the individuals during the search. It is possible to use more than one criterion to guide the search. In this case if the combined criteria are non-conflicting than the resulting fitness function is a linear combination of each fitness functions. The aim of this work is to study how search-based testing scale to combinations of multiple criteria does for unit testing in Java programs.

VI. SOFTWARE TESTING TOOLS

The tools summarized in this section are a select few that are used when performing automatic testing.

DOI 10.17148/IJARCCE.2016.5140

A. Ranorex

This is a cost-effective and comprehensive tool used for automatic testing. This is a better alternative to conventional testing tools because it tests applications from a user's perspective, using standard programming techniques and common languages such as C# and VB.net. It does not require you to learn a scripting language, because it is written in pure .net code. You can use any one of the three languages, C#, VB.net and Iron Python. It is used by hundreds of enterprise and commercial software companies around the world [9].

Ranorex is based on XPath, which is a very good way to find certain elements in a web based application. It is a pure .net API, which is very different from other tools which sit on an API [9].

Advantages [9]:

- Image-based recognition.
- Offers a flexible and standard test automation interface.
- Provides the ability to do test automation in your own environment.
- Allows testers with less programming knowledge to create professional test modules with Ranorex Recorder.
- User interface allows for managing test cases and configurations.
- Supports use of data variables

Disadvantage:

- Ranorex tool doesn't provide an option to export automation code to different environments like UFT (VBScript), Java.
- To add mobile device via Wi-Fi, it's necessary that the application under test is started on mobile device or simulator.

B. Rational Functional Tester (RFT)

This product was developed by IBM in 1999. It is an object-oriented automated testing tool. It includes functional and regression testing tools which capture the results of black box tests in a script format. With this tool, functional black box tests can be run as well as structural white box tests for memory leaks, code bottlenecks, or measuring code coverage.

The upgraded version of RFT "Baltic", or IBM Rational Release 7, was released in 2006. This platform automates much of the software development and delivery process and helps enterprises overcome geographic and organizational silos that hamper development projects. There are 12 products in this new software development platform [10].

Advantages [11]:

- Enables regression testing
- Frees up Quality Assurance departments from maintaining and executing basic tests, encouraging the creation of additional, thorough tests
- Reduces human error that can occur during activities such as test step execution and test result recording

Disadvantages:

- Requires a skilled tester.
- Cannot look into every bit of code to find out hidden errors.

RFT works with Java, Web based, Microsoft Visual Studio, .NET, terminal-based, SAP, Siebel and Web 2.0 applications [12]. This product uses a patented Object Code Insertion (OCI) technology where no source code is required. The technology looks at the application's executable files. The tools built into the software, including Purify Quantify and Pure Coverage, perform white box testing on third party code and controls.

Advantages:

- Provides run-time error and memory leak detection
- Records the exact amount of time the application spends in any given block of code for the purpose of finding inefficient code bottlenecks
- Pinpoints areas of application that have and have not been executed

Disadvantages:

- The test can be redundant if the software designer has already run a test case.
- Testing every possible input stream is unrealistic because it would take an inordinate amount of time.
- Difficult to design without clear specifications.

C. Janova

This tool is similar to others because it enables the user to automate software testing solutions but with this tool it is done in the cloud.

Advantage:

- The tool does not require scripts to be written; only English-based tools are used that streamlines the task of software implementation with efficient, easy to use tools.
- Low cost.
- There is no software to download and no infrastructural investment required [13].
- Since it is in the cloud, it is a very quick and easy setup which includes no install.
- The cloud based software has an easy to navigate home page

Disadvantage [14]:

- It took too much time to navigate through each test script and get past the failed tests.
- If the browser closed unexpectedly, the login session was automatically on hold for 15 minutes. This prevented the testing from progressing, due to this 15 minute timeout period in the tool.

D. QTP

QTP stands for Quick Test Professional, a product of Hewlett Packard (HP). This tool helps testers to perform an automated functional testing seamlessly without monitoring once script development is complete.

HP QTP uses Visual Basic Scripting (VBScript) for automating the applications. The Scripting Engine need

not be installed exclusively as it is available part of the Windows OS. The Current version of VBScript is 5.8 which is available as part of Win 7. VBScript is NOT an object oriented language but an object based language.

Advantages:
- It is easy even for a non-programmer to understand QTP and start adding test cases.
- Support for record and playback and ability to edit scripts after recording. Also different recording modes are provided in QTP viz. Normal, Analog & Low level.
- Support for different addins like Java, Oracle, SAP, .NET, Web Forms, Siebel, PeopleSoft, Web services, Main frame (Terminal Emulator) etc.
- Supports all popular Automation frameworks - Keyword driven testing approach, Data driven testing approach, Modular testing approach, Hybrid frameworks etc.
- QTP comes with an inbuilt IDE, which is simple and easy to use.
- Easy to maintain different types of suites viz. Smoke, Sanity, and Regression etc.
- Easy to maintain test iterations and data driving the tests through configurations.
- Test reporting with all necessary details for analysis is provided.

Disadvantages:
- Cost is high – License and maintenance.
- Cannot run multiple threads/instances – For example the Grid support available in Selenium, where we can run multiple instances of the application on different browsers at the same time.
- Slow in execution when compared to even open source tools like Selenium.
- You need to buy different addins – Java, Oracle, SAP, .Net, Seibel, Peoplesoft etc.

E. Squish GUI Tester

The Squish IDE, built on Eclipse, provides a feature-rich integrated development environment for GUI testing. Complete with test management, script debugging and an interactive application object spy.

Squish is the leading cross-platform/cross-technology GUI test automation tool for functional GUI regression tests. Many companies in all kinds of industries all over the world use Squish to drastically cut down the time spent on GUI testing software releases while increasing the quality of their applications.

Advantages:
- User has not to be logged in for testing on remote computers
- Utilization of common script languages
- Web testing available (though as a separate edition with additional cost)

Disadvantages:
- Developer-oriented solution
- Outdated appearing user interface, handling partially laborious
- No simple support of obfuscating
- No easy provision of screenshots in case of errors
- Limited integration in test management

F. TestComplete

TestComplete Platform has an open flexible architecture that makes creating, maintaining, and executing automated tests across desktop, web, mobile easy, speedy, and cost effective. Some of the powerful features of TestComplete Platform that demonstrate it's flexibly and ease of use.

Advantages:
- Support for multiple scripting languages
- Record robust automated tests without knowing scripting
- Write regression tests that don't fail when UI changes
- Perform Data Driven testing
- Create custom plugins and extensions

Disadvantages:
- Neither GridLookUpEdit nor XtraGrid are fully supported by TestComplete.TC records any object by coordinates. So when an object (ex: combo box) order changes or width changes, related tests fail
- Cannot create Data Driven Tests with coordinate points
- Some times TC was stuck in recognizing the cell values in a grids
- We have redesigned the application to use Virtual scrolling approach by keeping the same user interface. We were not able to run any of the existing tests as the positions of objects are slightly changed in new grids

G. eggPlant

The eggPlant range of test automation tools comprises a variety of tools to satisfy every testing need – from functional to performance, mobile to desktop, digital to legacy. eggPlant tools can function on their own, with test tools from other vendors, or together in a unified environment. Pick and choose which tools fit your needs and environment, and use them together to easily drive automated UI tests with accurate results.

Advantages:
- Guided Record Mode
- Instead of a regular record mode, capturing images generates script code and avoids extraneous mouse and keyboard movements being recorded
- Code Completion
- Easy integration with quality management software
- Eggplant can be used to augment systems such as HP Quality Center
- Execution-only mode
- eggPlant tests can be run unattended via command line.

Disadvantages:
- As EggPlant uses image matching technology, images captured in one operating system cannot work on other OS.
  Ex: Tests with Images captured in windows XP cannot be run on windows 2k8.We need to capture images

again in other operating system for tests to run successfully. Requires a dedicated resource to maintain the test scripts whenever we update Operating system or change the computers.

- Tests fail even if the resolution of system is changed while running tests. Image has to match 100% for any test to run.
- Expensive compared to other competitive tools in the market
- EggPlant is not very popular in market. So it is very difficult to find the resource having programing skills to work on Eggplant automation when compared to other competitive tools like Selenium Webdriver, QTP etc.

Due to above drawbacks, we failed one more time to continue our automation project with EggPlant tool.

VII. CONCLUSION

The testing of software is an important means of assessing the software to determine its quality. Since testing typically consumes 40 - 50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering.

Software testing can endow excellent results if done properly and effectively. Test cases are most important elements of software testing. If we design test cases in better manner and in proper way using test cases designing algorithm then we can produce a better software product. By doing this we will get the result as we want in the requirement specified in the software requirement specification (SRS). This paper mainly deals with testing techniques available for designing of test cases and also deals with some of existing testing tools. In the future, we can implement these test cases design techniques for real-time scenarios projects.

REFERENCES

[1]. "J. Miguel Rojas, J. Campos1, M. Vivanti, G. Fraser, A. Arcuri, "Combining Multiple Coverage Criteria in Search-Based Unit Test Generation" in Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 436-439, 2011

[2]. F. Saglietti, N. Oster, and F. Pinte, "White and grey-box verification and validation approaches for safety-and security-critical software systems," Information Security Technical Report, vol. 13, no. 1, pp. 10–16, 2008.

[3]. T. Murnane and K. Reed, "On the effectiveness of mutation analysis as a black box testing technique," in Software Engineering Conference, 2001. Proceedings. 2001 Australian, 2001, pp. 12 –20.

[4]. Nidhra, Srinivas, and Jagruthi Dondeti. "Black box and white box testing techniques- A Literature." International Journal of Embedded Systems & Applications 2.2 (2012).

[5]. Khan, Mohd Ehmer, and Farmeena Khan. "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques." International Journal of Advanced Computer Sciences and Applications 3, no. 6 (2012): 12-15

[6]. P. Mitra, S. Chatterjee, and N. Ali, "Graphical analysis of MC/DC using automated software testing,"in Electronics Computer Technology (ICECT), 2011 3rd International Conference on, 2011, vol. 3, pp. 145 – 149.

[7]. Sara Sprenkle, Holly Esquivel, Barbara Hazelwood, Lori Pollock, WebVizor:A Visualization Tool for Applying Automated Oracles and Analyzing Test Results of Web Applications, IEEE Computer Society, August 2008.

[8]. David Crowther, Peter Clarke, Examining Software Testing Tools, Dr. Dobb"s Journal:Software Tools for the Professional Programmer, ISSN# 1044789X, Academic Search Premier, June 2005, Vol. 30, Issue 6.

[9]. Ranorex website, 2012 Ranorex GmbH, URL: www.ranorex.com.

[10]. Darryl Taft, IBM Readies Rational Revamp, EBSCO host database, ISSN# 15306283, Academic Search Complete, June 2006.

[11]. IBM.com, IBM Rational Functional Tester, IBM Corporation, December 2008, URL: http://public.dhe.ibm.com/common/ssi/ecm/en/rad14072usen/RAD14072USEN.PDF.

[12]. IBM developerWorks, Bridging the Gap Between Black Box and White Box Testing, URL: http://www. ibm. com/developerworks/rational/library/1147. html.

[13]. Enhanced online News, Janova Launces Simple Yet Powerful Automated Software Testing Solutions Leveraging the Power of the Cloud, 2012 Business Wire, December 1, 2011URL: http://eon.businesswire.com/news/eon/20110412005635/en/softwaretesting/
software-development/software.

[14]. Nancy Bordelon, A Comparison of Automated Software Testing Tools, A Capstone Project Submitted to the University of North Carolina Wilmington in Partial Fulfillment of the Requirements for the Degree of Master of Science.

[15]. Godbole, N. (2004). Software quality assurance. Pangbourne, U.K.: Alpha Science International Ltd.

[16]. M. Whalen, G. Gay, D. You, M. P.E. Heimdahl, M. Staats "Observable Modified Condition/Decision Coverage", In Proceedings In Proceedings of International Conference in Software Engineering (ICSE), 2013.

[17]. P. Amann, J. Offut, "Introduction to Software Testing", 2008

[18]. K. Lakhotia, P. McMinnb, M. Harman, "An empirical investigation into branch coverage for C programs using CUTE and AUSTIN". Journal of Systems and Software, 2010

[19]. G. Fraser, A. Arcuri, "Achieving Scalable Mutationbased Generation of Whole Test Suites". Empirical Software Engineering 2014.

BIOGRAPHY

T. Ramasundaram Received M.Sc., degree in Computer Science from Govt.Arts College, Dharmapuri, Periyar University, Salem in 2005 and M.Phil., Degree in Computer Science from Periyar University, Salem, in 2009. Now he is a research scholar in Department of Computer Science, Periyar University, Salem. He is working as Assistant Professor in Department of Computer Science, Sri Vijay Vidyalaya College of Arts & Science, Nallampalli, Dharmapuri. His research interests are Software Engineering, Data Mining.