"Useful Activities and Challenges of Effectiveness Automated Software Testing"

Latharani T R

Research Scholar

Abstract – The design of an appropriate test suite for software testing is a challenging task. It requires a suitable tradeoff between effectiveness, e.g., a sufficient amount of test cases to satisfy the test goals of a given coverage criterion, and efficiency, e.g., a redundancy-reduced selection of test cases. In this paper we discussed activities and challenges of effectiveness automated software testing.

Keyword: Activities, Challenges, Automated Software Testing

----------♦----------

INTRODUCTION

Automation testing is used to rerun the test scenarios that were performed manually, quickly and repeatedly. Automation testing which is also known as test automation is when the tester writes scripts and uses software to test the software. This process involves automation of a manual process.

Test automation has often been touted as an important part of an organization's quality strategy.

Apart from regression testing, Automation testing is also used to test the application from load, performance and stress point of view. It increases the test coverage; improve accuracy, saves time and money in comparison to manual testing.

Test automation has always been an attractive alternative to expensive, time consuming and inconsistent manual testing. Key program factors include: the development paradigm, the quality objectives, and your deployment velocity. When, how and how much test automation to apply against a program is dependent on these factors - the return on investment must align with these factors; otherwise, the long-term success of the test automation effort will be in jeopardy and almost certainly fail. The most common key program factors are:

- Development paradigm (Agile, non-Agile, instrumented, non-instrumented)
- Quality objectives (defect escape velocity)
- Target deployment velocity (volume of new/enhanced functionality per release)

REVIEW OF LITERATURE:

Software permeates many aspects of our life; thus, improving software reliability is becoming critical to society. A recent report by National Institute of Standards and Technology found that software errors cost the U.S. economy about $60 billion each year [1]. Although much progress has been made in software verification and validation, software testing is still the most widely used method for improving software reliability. However, software testing is labor intensive, typically accounting for about half of the software development effort [2].

To reduce the laborious human effort in testing, developers can conduct automated software testing by using tools to automate some activities in software testing. Software testing activities typically include generating test inputs, creating expected outputs, running test inputs, and verifying actual outputs. Developers can use some existing frameworks or tools such as the Unit testing framework [3] to write unit-test inputs and their expected outputs. Then the Unit framework can automate running test inputs and verifying actual outputs against the expected outputs. To reduce the burden of manually creating test inputs, developers can use some existing test-input generation tools [4-6] to generate test inputs automatically. After developers modify a program, they can conduct regression testing by rerunning the existing test inputs in order to assure that no regression faults are introduced. Even when expected outputs are not created for the existing test inputs, the actual outputs produced by the new version can

be automatically compared with the ones produced by the old version in order to detect behavioral differences.

ACTIVITIES AND CHALLENGES OF AUTOMATED SOFTWARE TESTING:

Software testing activities consist of four main steps in testing a program: generating test inputs, generating expected outputs for test inputs, run test inputs, and verify actual outputs. To reduce the laborious human effort in these testing activities, developers can automate these activities to some extent by using testing tools. Our research focuses on developing techniques and tools for addressing challenges of automating three major testing activities: generating test inputs, generating expected outputs, and verifying actual outputs, particularly in the absence of specifications, because specifications often do not exist in practice. The activities and challenges of automated software testing are described below.

GENERATE (SUFFICIENT) TEST INPUTS:

Test-input generation often occurs when an implementation of the program under test is available. However, before a program implementation is available, test inputs can also be generated automatically during model-based test generation [19,20] or manually during test-driven development [6], a key practice of Extreme Programming [7]. Because generating test inputs manually is often labor intensive, developers can use test-generation tools [4-6] to generate test inputs automatically or use measurement tools [10-12] to help developers determine where to focus their efforts. Test inputs can be constructed based on the program's specifications, code structure, or both. For an object-oriented program such as a Java class, a test input typically consists of a sequence of method calls on the objects of the class.

CONCLUSION:

Activities ultimately result in some *action*, which is some set of pure computation. An important fact about the collaboration and activity diagrams is that they are most useful for constructing executable systems through forward and reverse engineering [13].

REFERENCES:

1. National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3, May 2002.

2. Boris Beizer. *Software Testing Techniques*. International Thomson Computer Press, 1990.

3. Erich Gamma and Kent Beck. JUnit, 2003. http://www.junit.org.

4. Parasoft Jtest manuals version 4.5. Online manual, April 2003. http://www. parasoft.com/.

5. Christoph Csallner and Yannis Smaragdakis. JCrasher: an automatic robustness tester for Java. *Software: Practice and Experience*, 34:1025–1050, 2004.

6. Agitar Agitatior 2.0, Novermber 2004. http://www.agitar.com/.

7. Kent Beck. *Extreme programming explained*. Addison-Wesley, 2000.

8. Quilt 0.6a, October 2003. http://quilt.sourceforge.net/.

9. Jcoverage 1.0.5, 2003. http://jcoverage.com/.

10. Susan B. Horwitz. Tool support for improving test coverage. In *Proc. 11th European Symposium on Programming*, pages 162–177, Grenoble, France, April 2002.

11. Alain Faivre and Jeremy Dick. Automating the generation and sequencing of test cases from model-based specifications. In Proc. 1st International Symposium of Formal Methods. Europe on Industrial-Strength Formal Methods, pages 268–284, London, UK, 1993.

12. Wolfram Schulte, Yuri Gurevich, Wolfgang Grieskamp, and Margus Veanes. In Proc. International Symposium on Software Testing and Analysis, pages 112 to 122 – 2002

13. G. Booch, J. Rumbaugh, and I. Jacobson, the Unified Modeling Language User Guide. Addison-Wesley, 2001.