Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Declarative Software Testing

Abhishek Sharma*
M.TECH, CSE Student(Amity University)
Noida, India

Prof (Dr) Ajay Rana
Program Director, CSE, Amity University
Noida, India

Abstract- ***To produce high quality software, both software developers and testing experts need continuous improvement and change in their work strategies and courses of action. Along these lines, far much work has been carried out in the successful methods for archiving the necessities. However essential viewpoint is to verify that whatever is documented in specifications really works accurately in the created software. Software testing is carried out to guarantee this aspect. Point of this paper is to concise a software product testing methodology utilizing the idea of Components that aides in distinguishing proof and choice of suitable test methods that could be utilized as data to acceptance testing and as starting steps to begin the genuine testing procedure of the system. This model concentrates on arranging the system test by nearly indicating them with system's requirements, prerequisites, and important functional use cases. By this it implies, that the primary emphasis of this model is on the importance of the functionality of the complete system composing of the critical use cases, requirements traceability matrix, test case prioritization and application acceptance criteria. An organized approach to outline test cases is made with the assistance of important use cases. Some work is done to trace the user's needs to system requirements and use cases and benefits of using use case modelling approach in structuring the relationships among test cases is analysed. As tests cases are unpredictable and are quite liable to changes in future, challenges imposed due to traceability among requirements, use cases and test cases are main subjects of this work along with the challenges faced by software testers to perform application acceptance testing. A green path plan is proposed to help testing experts characterize application's acceptance criteria and weight assignment approach is used to prioritize the test cases to determine the successful running of the application.***

Keywords: component testing, test artefact, specifications, acceptance criteria, prioritized testing.

I. INTRODUCTION

Software Testing is a methodology of checking that the functionality implemented is acting according to the requirement specified by the customers or users mentioned in the software requirement specification document. Software testing is carried out to dispose of numerous types of errors from the application created by the engineers. The slips that are not identified throughout the testing process, later when they fail in the real world situation are referred to as failures.

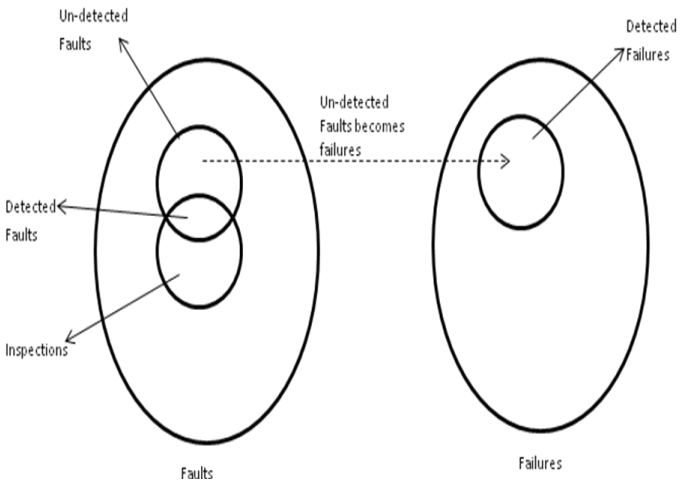

Fig 1: Faults mapping to Failures in the system

Testing has experienced extensive change of modernization throughout the decade and there is still an inclination to move it more upstream in the development process. Other than the mechanisms of verification, validation, inspection and reviews; testing is still an important and relied technology to identify errors in the software product and then referring those errors back to the development process for fixation [2]. Goal of effective testing is to reveal high severity errors as

early as possible. In reality this is not completely possible but planned efforts to tackle this issue can considerably reduce the severity of this issue [1].

In spite of the fact that better approaches for software inspection are present like code reviews, requirements analysis, change impact analysis, peer reviews etc; even now testing is the primary source of software quality assurance. It comprises of running a test as well as it covers test case designing, expected outcomes, test case modelling, real test data preparation and requirement verification.

Software testing might be expressed as the procedure of accepting and checking that a software program/application/product:

1. Meets the requirements that guided its design and development
2. Functions as expected
3. Might be executed with the same attributes
4. Fulfills the needs of stakeholders

Since testing is the stage which is vigorously depended by different development phases to ensure software quality, so this focuses all the more on the improvements in the testing phase of software development which is the process of executing.

II. INFERENCE

It is evident that for testing the complete system, we cannot rely on Unit Testing completely. There are some drawbacks in the methodology of Unit Testing. Unit testing cannot be expected to catch every error in the program. It is impossible to evaluate every execution path. Unit testing does not catch integration errors or broader system-level errors (such as functions performed across multiple units, or non-functional test areas such as performance).

Unit testing is only effective if it is used in conjunction with other software testing activities. It is unrealistic to test all possible input combinations for any piece of software. Like all forms of software testing, unit tests can only show the presence of errors; it cannot show the absence of errors. To obtain the intended benefits from unit-testing, a rigorous sense software to assure its correctness with respect to specification. Any deviation from specification is termed as defect or failure and is determined by compared vs. intended behavior of the system [2]. Deviations from specification identified during in-house inspection are termed as defects while deviations reported during or after deployment process are called failures. It is always difficult to claim with surety about the quality of the software and that it will never fail during its real time execution.

Testing is divided into two main areas black box testing and white box testing [3]. Black box testing is highly dependent on functional specifications while white box testing is dependent on implementation details like algorithms, data structures, path testing etc.

It is essential to keep careful records, not only of the tests that have been performed, but also of all changes that have been made to the source-code of this or any other unit in the software. So we need some other approach to decrease the effort and also which could ensure that the tested application is more reliable if it passes some number of test cases.

III. SOLUTION IS WEIGHTS

The most important decision one does while performing the testing is to find the answer to the question "What is a test criterion?", that is, the criterion that defines what constitutes an adequate test. To determine appropriate test criteria, has always been in focus. A great number of such criteria's have been proposed. Testing any functionality depends on various factors. The factors are sated below:

- Criticality of that unit
- Dependency of that unit with other units
- Other units that are dependent on that unit
- Occurrence of that unit
- Functionality of that unit
- End user's operation

The factors stated above changes with the functional of the unit/system that we want to test. The criteria to perform testing could be testing based on the weight of the Components. A component consists of a single block of code that can be invoked in the way that procedure, function or method is invoked. Component-based software engineering (CBSE) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.

All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes). Because of this principle, it is often said that components are modular and cohesive. With regard to system-wide co-ordination, components communicate with each other

via interfaces. When a component offers services to the rest of the system, it adopts a provided interface that specifies the services that other components can utilize, and how they can do so.

Components are substitutable [1], so that a component can replace another (at design time or run-time), if the successor component meets the requirements of the initial component (expressed via the interfaces). Consequently, components can be replaced with either an updated version or an alternative without breaking the system in which the component operates.

IV. CONCLUSION

We are able to identify the flaws and limitations of currently used unit/component testing approach. We are also informed about the concept of Weight of Components. We are able to learn that Software testing is done to ensure that whatever is written in the Requirement Specifications is met. We also learnt a software testing approach using the concept of Components that helps in identification and selection of suitable test paths that can be used as input to acceptance testing and as a pre-requisite to start actual testing of the system. We came to know about the traditional aspects of the IT industry regarding the software development.

We also focused on the conventional testing approaches that being used in the IT industry these days which follows the Software Development Cycle. We understood the role of developer and the tester both, for the testing of the software. These days in the IT industry, testing is not the only job of testers; developers are also involved in the testing process, generally called Unit Testing. Finally we discussed about various approaches regarding Component Weight testing; which states that every component has some cost associated with it and we should choose a component depending upon its criticality and cost.

In this paper we studied the current standard unit testing approach the industry follows and also where it fails. Also we devised an idea of component weights and better and comprehensive testing that can be done using Component weights. In future the Study can be expanded into further details about component weights and also may lead into development of a new Component based testing model, which would be better that the currently used approach of unit testing.

ACKNOWLEDGMENT

I take this chance to express my deep gratitude and regards to my guide Prof. (Dr.) Ajay Rana, Program Director - Amity School of Engineering and Technology for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life. I also express a deep sense of gratitude to my Mentor and Colleague Mr. Chirag Goel, Technical Lead, Xebia IT Architects, for his/her cordial support, valuable information, which helped me in completing this task through various stages of project development and design.

Lastly, I thank god, my parents, and friends for their constant support and motivation without which this assignment would not be feasible.

REFERENCES

[1] Xie, Tao.; "Towards a Framework for Differential Unit Testing of Object-Oriented Programs"; Wiley-IEEE Computer Society Press.

[2] Cem, Kaner ; Exploratory Testing ; Quality Assurance Institute Worldwide Annual Software Testing Conference 2006.

[3] Kolawa, Adam; Huizinga, Dorota. ; Automated Defect Prevention: Best Practices in Software Management; Wiley-IEEE Computer Society Press.

[4] Leitner, A., Ciupa, I., Oriol, M., Meyer, B., Fiva, A. ; "Contract Driven Development = Test Driven Development - Writing Test Cases" ; European Software Engineering Conference on the Foundations of Software Engineering 2007

[5] IEEE, "IEEE Standard Glossary of Software Engineering Terminology" (IEEE Std 610.12-1990), Los Alamitos, CA: IEEE Computer Society Press, 1990

[6] Vilkomir, A, Kapoor, K & Bowen, JP, "Tolerance of Control-flow testing Criteria", Proceedings 27th Annual International Computer Software and applications Conference, 3-6 November 2003, 182-187, or http://ro.uow.edu.au/infopapers/88

