6th International Conference on Recent Trends in Engineering, Science & Management
Shri Guru Teg Bahadur Khalsa College Anandpur Sahib, Punjab (ICRTESM-17)
8th January 2017 , www.conferenceworld.in ISBN: 978-93-86171-21-4

A STUDY OF SOFTWARE TESTING AND IT'S APPLICATION

Darpan Sood[1], Charu Gupta[2]

Shri Guru Teg Bahadur Khalsa college, Anandpur Sahib, Rupnagar (India)

ABSTRACT

Software testing has become an important tool in modern world to investigate the authenticity, working , reliability and sustainability of various software's that are developed these days to cater the demand .The developed system is evaluated to check the quality of the ensemble product. Software testing is one of the most commonly used and accepted technique that is used worldwide for the betterment of the product. In this paper we will discuss the software testing life cycle.

***Keywords*: - *Application, Development, Product, Software, Testing*.**

I. INTRODUCTION

Very early in the development of computers, people referred to the actual physical components - the tubes and relays, the resistors and wires, and chassis – as computer *hardware*. The word *software* was then coined to describe the non-hardware components of the computer, in particular the programs that were needed to make the computers perform their intended tasks. The word caught on rapidly, and was in quite general use by 1960. One speaks of software shops (i.e. organizations that produce software), software maintenance, and, more recently, software engineering. Although the word *software* can be used in connection with all kinds of programs, it is usually used to denote programs whose use is not limited to one particular job or application. Thus, one speaks of system software, of software systems, of mathematical software, of software for business applications, etc.

Growth in software engineering technology has led to production of software for highly complex situations occurring in industry, scientific research, defense and day to day life. The computer revolution is fueled by an ever more rapid technological advancement. Today, computer hardware and software permeates our modern society. Computers are embedded in wristwatches, telephones, home appliances, buildings, automobiles, and aircraft. Science and technology demand high-performance hardware and high-quality software for making improvements and breakthroughs. We can look at virtually any industry - automotive, avionics, oil, telecommunications, banking, semi-conductors, pharmaceuticals - all these industries are highly dependent on computers for their basic functioning. When the requirements for and dependencies on computers increase, the possibility of cries from computer failures also increase. The impact of these failures ranges from inconvenience (eg., malfunctions of home appliances) to economic damage (eg., interruption of banking systems) to loss of life (eg., failure of flight systems or medical software). Needless to say, the reliability of computer systems has a major concern for our society. Though high reliability of hardware part of these systems can be guaranteed, the same cannot be said for the software. Therefore a lot of importance is attached to the testing phase of the software development process, where around half the development resources are used [Musa

ISBN: 978-93-86171-21-4

et al., 1988]. Essentially testing is a process of executing a program with the explicit intention of finding faults and it is this phase, which is amendable to mathematical modeling.

It is always desirable to remove a substantial number of faults from the software. In fact the reliability of the software is directly proportional to the number of faults removed. Hence the problem of maximization of software reliability is identical to that of maximization of fault removal. At the same time testing resource are not unlimited, and they need to be judiciously used.

In focusing on error prevention for reliability, we need to identify and measure the quality attributes applicable at different life cycle phases. As discussed previously, we need to specifically focus on requirements, design, implementation, and test phases.

1.1. Testing Phase in Software Development Life Cycle

Software development process is often called Software Life Cycle, because it describes the life of a software product from its conception to its implementation. Every software development process model includes system requirements as input and a delivered product as output. Many life cycle models have been proposed, based on the tasks involved in developing and maintaining software, but they all consist of the following stages and faults can be introduced during any of these stages.

1. **Requirement**

   Analysis phase precedes all other activity for software development. During this phase the system service, constraints and goals are established. The user requirement are understood and specified. Though actual designing or coding is done, incorrect, missing or unclear requirements as specified by the user can lead to faults. In the Specification the translations of requirements into precise description of the externals of the software system are done. Hence defective software can result if user requirements are misinterpreted.

2. In the **Design** phase the requirements and specifications are transformed into a structure that is suitable for implementation in some programming language. Here, overall software architecture is defined, and the high level and detailed design work is performed. Misinterpretations are common when the system design is translated into lower-level descriptions for program design specifications.

3. During **Implementation and Unit testing** phase, the software system is created, which implements the design. Unit testing ensures that each unit meets its specifications. The programmers and designers of the development team may commit some logical errors that cannot be pointed out by the compiler. The passing of parameters between modules is not considered during unit testing and can hide a fault.

4. In **System testing**, the individual programs are integrated and tested to determine whether the implementation satisfies the requirement. Though this phase aims at removing all the faults included in the above phases, faults can be introduced into the software during the removal process. The theory developed in this thesis primarily addresses the testing phase.

5. During **Maintenance** in the operational phase faults are corrected, which were not discovered during the previous stages, and enhances the performance of the software system. Incorrect user documentation, poor human factors and changes in requirement can lead to failure reports by the users.

The efforts to improve the software development process are accompanied with parallel efforts aiming at ensuring high quality software systems. The software quality assurance consists of those procedures, techniques and tools

applied by professionals to ensure that a software product meets or exceed pre-specified standards during software development cycles.

1.2. Software Quality Characteristics

Quality is built into the software; it does not just happen to be there because the developers did a good job. Quality assurances practices are concurrent with all process activities. Quality is defined as "the degree of excellence of something". This implies a subjective factor; any project can be found lacking if measured against a vague notion of what high quality is. Software quality in the context of software engineering measures how well software is designed (*quality of design*), and how well the software conforms to that design (*quality of conformance*) [Musa et al., 1990], (quality of design) measures how valid the design and requirements are in creating a worthwhile product [Triantafyllos et al., 1995] whereas (quality of conformance) is concerned with implementation. Software quality is measured via a set of attributes that are characteristics of high-quality software. Then we build into the requirements the attributes that are desired in the final product. The desired quality attributes are the part of the standard for measurement on the project. It is not always possible to measure each attribute directly, but some form of relative measurement must be made. Common among the characteristics are: completeness, correctness, depenendability, efficiency, reliability, maintainability, portability, robustness (the ability to minimize the impact of external factors, such as user errors or adverse environmental conditions), testability, and usability, but the list is not limited to these.

II. ASSUMPTIONS

1. The error detection phenomenon is modelled by NHPP.
2. Software is subject to failure at random times caused by errors remaining in the software.
3. Corresponding to the failure phenomenon at the user/manufacturer's end, there is an equivalent failure phenomenon at the manufacturer/user's end .
4. An error which caused a failure will not cause any further failure until removed.
5. All errors in the software are mutually independent.

BIBLIOGRAPHY

[1.] **Hossain S.A. and Dahiya R.C. (1993)**, "Estimating the Parameters of a Non-Homogeneous Poisson Process Model for Software Reliability", IEEE Trans. Reliability, vol. 42, pp. 604-612.

[2.] **Hou R.H., Kuo S.Y. and Chang Y.P. (1997)**, "Optimal Release Times for Software systems with Scheduled Delivery Time Based on the HGDM," IEEE Trans. Computers, vol. 46, no. 2, pp. 216-221.

[3.] **Huang C. Y. (2005)**, "Cost-reliability Optimal Release Policy for Software Reliability Models incorporating Improvements in Testing Efficiency", The Jouranal of Systems and Software, vol.77, pp.139-155.

[4.] **IEEE Standard 982.2 (1987)**, Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software.

[5.] **Kapur P.K. and Garg R.B. (1990)**, "Optimal software release policies for software reliability growth models under imperfect debugging" , R.A.I.R.O. 24 (3), pp. 295-305.

[6.] **Kapur P.K. and Garg R.B. (1990)**, "Cost reliability optimum release policies for a software system with testing effort", OPSEARCH, vol.27, No.2,pp.109-118.

[7.] **Kareer N., Kapur P.K. and Grover P.S. (1990)**," An S-Shaped software reliability growth model with two types of errors", Microelectronics and Reliability, vol. 30, no. 6, pp 1085-1090.

[8.] **Kapur P.K., Bai M. and Bhushan S. (1992)**, "Some stochastic models in software reliability based on NHPP", In Contribution to Stochastic, (Ed.) N. Venugopal, Wiley Eastern Limited, New Delhi.

[9.] **Kapur P.K. and Bhalla V.K. (1992)**,"Optimal release policies for a flexible software reliability growth model", Reliability Engineering and System Safety, vol. 35, pp 45-54.

[10.] **Kapur P.K., Garg R.B. (1992)**, "A software reliability growth model for an error removal phenomenon", Software Engineering Journal, 7; 291-294.

[11.] **Kapur P.K., Agarwala S. and Garg R.B. (1994)**, "Bicriterion release Policy for an exponential software reliability growth model", R.A.I.R.O., vol.28, pp. 165-180.

[12.] **Kapur P.K., Agarwala S. and Kumar S. (1994)**, "Bicriterion release Policy for discrete software reliability growth model under imperfect debugging", IJOMAS, vol 10, 2, pp.131-146.

[13.] **Kapur, P.K., Younes, S. (1995)**, "A general discrete software reliability growth model", Published in Operations Research-Theory and Practice, Spaniel Publishers, New Delhi.

[14.] **Kapur P.K., Garg R.B and Kumar, S. (1999)**, "Contributions to Hardware and Software Reliability", World Scientific, Singapore.

[15.] **Kapur P.K., Shatnawi O. and Singh O. (2006)**, "Discrete NHPP model for software reliability growth with imperfect debugging and fault generation model" International Journal of Performability Engineering vol 2.No.4 pp.351-368.

[16.] **Leung Y.W. (1992)**, "Optimal Software Release Time with a Given Cost Budget," J. Systems and Software, vol. 17, pp. 233-242.

[17.] **Lynch T., Pham H. and Kuo W. (1994)**, "Modeling Software Reliability with Multiple Failure-Types and Imperfect Debugging", Proceedings Annual Reliability and Maintainability Symposium, pp.235-240.

[18.] **Lyu M. (1996)**, Ed., Handbook Software Reliability Engineering. New York: McGraw-**Misra P.N. (1983)**, "Software Reliability Analysis," IBM Systems J., vol. 22, pp. 262-270 **Musa J.D., lannino A. and Okumoto K. (1988)**, "Software Reliability Measurements", Prediction and Application, Mc Graw Hill.

[19.] **Musa, J.D., lannino A. and Okumoto K. (1990)**, "Software Reliability: Measurement, Prediction, Application", Professional Edition: Software Engineering Series, McGraw–Hill, New York, NY.

[20.] **Okumoto K. and Goel A.L. (1980)**, "Optimum release time for software systems based on cost and reliability criteria, J. Systems and Software, vol.1, pp.315-318.