SOFTWARE TESTING- TOOLS AND TECHNIQUES

Dr. Rajneesh Talwar*

Dr. Bharat Bhushan**

Rakesh Gupta***

ABSTRACT

Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Although crucial to software quality and widely deployed by programmers and testers, software testing still remains an art, due to limited understanding of the principles of software. The difficulty in software testing stems from the complexity of software: we can not completely test a program with moderate complexity. Testing is more than just debugging. The purpose of testing can be quality assurance, verification and validation, or reliability estimation. Testing can be used as a generic metric as well. Correctness testing and reliability testing are two major areas of testing. Software testing is a trade-off between budget, time and quality. In this paper we study different testing techniques and comparative analysis of these techniques.

*Professor, RIMT, Mandi Gobindgarh, Punjab, India.

**Associate Professor, G. N. Khalsa College,Yamunnagar, Haryana, India.

***Assistant Professor, SDDIET, Barwala, Panchkula, Haryana, India.

I. INTRODUCTION

The importance of software testing and its impact on software cannot be underestimated. Software testing is a fundamental component of software quality assurance and represents a review of specification, design and coding. The greater visibility of software systems and the cost associated with software failure are motivating factors for planning, through testing. It is not uncommon for a software organization to spent 40% of its effort on testing [1-2].

This paper is organized as follows. Section 2 describes the fundamental of software testing. Software testing techniques are explained in section 3. Testing for real systems is discussed in section 4. Section 5 includes automated testing tools followed by conclusions in section 6.

II. SOFTWARETESTING FUNDAMENTALS

During testing the software engineering produces a series of test cases that are used to "rip apart" the software they have produced. Testing is the one step in the software process that can be seen by the developer as destructive instead of constructive. Software engineers are typically constructive people and testing requires them to overcome preconceived concepts of correctness and deal with conflicts when errors are identified [1].

A. Testing objectives

A number of rules that act as testing objectives are:

- Testing is a process of executing a program with the aim of finding errors.
- A good test case will have a good chance of finding an undiscovered error.
- A successful test case uncovers a new error.

B. Test information flow

Information flow for testing follows the pattern shown in the figure below. Two types of input are given to the test process: (1) a software configuration; (2) a test configuration. Tests are performed and all outcomes considered, test results are compared with expected results. When erroneous data is identified error is implied and debugging begins. The debugging procedure is the most unpredictable element of the testing procedure. An "error" that indicates a discrepancy of 0.01 percent between the expected and the actual results can take hours, days or months to identify and correct. It is the uncertainty in debugging that causes testing to be difficult to schedule reliability.

C. Test Case Design

The design of software testing can be a challenging process. However software engineers often see testing as an afterthought, producing test cases that feel right but have little assurance that they are complete. The objective of testing is to have the highest likelihood of

finding the most errors with a minimum amount of timing and effort. A large number of test case design methods have been developed that offer the developer with a systematic approach to testing. Methods offer an approach that can ensure the completeness of tests and offer the highest likelihood for uncovering errors in software. Any engineering product can be tested in two ways: (1) Knowing the specified functions that the product has been designed to perform, tests can be performed that show that each function is fully operational (2) knowing the internal workings of a product, tests can be performed to see if they jell. The first test approach is known as a black box testing and the second white box testing.

Black box testing relates to the tests that are performed at the software interface. Although they are designed identify errors, black box tests are used to demonstrate that software functions are operational; that inputs are correctly accepted and the output is correctly produced. A black box test considers elements of the system with little interest in the internal logical arrangement of the software. White box testing of software involves a closer examination of procedural detail. Logical paths through the software are considered by providing test cases that exercise particular sets of conditions and/or loops. The status of the system can be dentified at diverse points to establish if the expected status matches the actual status.

III. SOFTWARE TESTING TECHNIQUES

StaticTesting

The Verification activities falls into the category of Static Testing [4]. During static testing, you have a checklist to check whether the work you are doing is going as per the set standards of the organization. These standards can be for Coding, Integrating and Deployment. Reviews, Inspection's and Walkthrough's are static testing methodologies.

A. Dynamic Testing

Dynamic Testing involves working with the software, giving input values and checking if the output is as expected [2]. These are the Validation activities. Unit Tests, Integration Tests, System Tests and Acceptance Tests are few of the dynamic Testing methodologies. As we go further, let us understand the various Test Life Cycle's and get to know the Testing Terminologies.

B. Unit Testing

Top of FormBottom of FormIn computer programming, a unit test is a method of testing the correctness of a particular module of source code.

The idea is to write test cases for every non-trivial function or method in the module so that each test case is separate from the others if possible. This type of testing is mostly done by the developers. Unit testing helps eliminate uncertainty in the pieces themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts will make integration testing easier. It is important to realize that unit-testing will not catch every error in the program. By definition, it only tests the functionality of the units themselves. Therefore, it will not catch integration errors, performance problems and any other system-wide issues. In addition, it may not be trivial to anticipate all special cases of input the program unit under study
may receive in reality. Unit testing is only effective if it is used in conjunction with other software testing activities.

C. Integartion Testing

Integration testing (sometimes called Integration and Testing, abbreviated **I&T**) is the phase of software testing in which individual software modules are combined and tested as a group. It follows unit testing and precedes system testing.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing. The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing.

The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up.

D. Volume Testing

Whichever title you choose (for us volume test) here we are talking about realistically exercising an application in order to measure the service delivered to users at different levels of usage. We are particularly interested in its behavior when the maximum numbers of users are concurrently active and when the database contains the greatest data volume.

The creation of a volume test environment requires considerable effort. It is essential that the correct level of complexity exists in terms of the data within the database and the range of transactions and data used by the scripted users, if the tests are to reliably reflect the to be production environment. Once the test environment is built it must be fully utilised. Volume tests offer much more than simple service delivery measurement. The purpose of volume testing is to find weaknesses in the system with respect to its handling of large amount of data during extended time periods.

E. Stress Testing

The purpose of stress testing is to find defects of the system capacity of handling large numbers of transactions during peak periods. For example, a script might require users to login and proceed with their daily activities while, at the same time, requiring that a series of workstations emulating a large number of other systems are running recorded scripts that add, update, or delete from the database.

F. Performance Testing

System performance is generally assessed in terms of response time and throughput rates under differing processing and configuration conditions. According to Hamilton, the performance problems are most often the result of the client or server being configured inappropriately.

The best strategy for improving client-sever performance is a three-step process. First, execute controlled performance tests that collect the data about volume, stress, and loading tests. Second, analyse the collected data. Third, examine and tune the database queries and, if necessary, providetemporary data storage on the client while the application is executing.

G. Smoke Testing

Smoke testing is a term used in plumbing, woodwind repair, electronics, and computer software development. It refers to the first test made after repairs or first assembly to provide some assurance that the system under test will not catastrophically fail. After a *smoke test* proves that the pipes will not leak, the keys seal properly, the circuit will not burn, or the software will not crash outright, the assembly is ready for more stressful testing.

In plumbing, a *smoke test* forces actual smoke through newly plumbed pipes to find leaks, before water is allowed to flow through the pipes.

In woodwind instrument repair, a smoke test involves plugging one end of an instrument and blowing smoke into the other to test for leaks. (This test is no longer in common use)

In electronics, a *smoke test* is the first time a circuit is attached to power, which will sometimes produce actual smoke if a design or wiring mistake has been made.

In computer programming and software testing, *smoke testing* is a preliminary to further testing, which should reveal simple failures severe enough to reject a prospective software release. In this case, the smoke is metaphorical.

Smoke testing in software development

Smoke testing is done by developers before the build is released or by testers before accepting a build for further testing.

In software engineering, a *smoke test* generally consists of a collection of tests that can be applied to a newly created or repaired computer program. Sometimes the tests are performed by the automated system that builds the final software. In this sense a smoke test is the process of validating code changes before the changes are checked into the larger product's official source code collection. Next after code reviews, *smoke testing* is the most cost effective method for identifying and fixing defects in software; some even believe that it is the most effective of all.

In software testing, a *smoke test* is a collection of written tests that are performed on a system prior to being accepted for further testing. This is also known as a build verification test. This is a "shallow and wide" approach to the application.

Sanity Testing

A sanity test or sanity check is a basic test to quickly evaluate the validity of a claim or calculation. In mathematics, for example, when dividing by three or nine, verifying that the sum of the digits of the result is a multiple of 3 or 9 (casting out nines) respectively is a sanity test.

In computer science it is a very brief run-through of the functionality of a computer program, system, scalculation, or other analysis, to assure that the system or methodology works as expected, often prior to a more exhaustive round of testing.

In software development, the sanity test (a form of software testing which offers "quick, broad, and shallow testing, determines whether it is reasonable to proceed with further testing. Software sanity tests are commonly conflated with smoke tests. A smoke test determines whether it is possible to continue testing, as opposed to whether it is reasonable. A software smoke test determines whether the program launches and whether its interfaces are accessible and responsible (for example, the responsiveness of a web page or an input button). If the smoke test fails, it is impossible to conduct a sanity test. In contrast, the ideal sanity test exercises the smallest subset of application functions needed to determine whether the application logic is generally functional and correct (for example, an interest rate calculation for a financial application). If the sanity test fails, it is not reasonable to attempt

more rigorous testing. Both sanity tests and smoke tests are ways to avoid wasting time and effort by quickly determining whether an application is too flawed to merit any rigorous testing. Many companies run sanity tests on a weekly build as part of their development process.

Interface Testing

Interface testing is the testing in which the interfaces between system components are tested. Traditionally the incorrect mapping of data between the systems causes these bugs and these may result in the following types of bugs:

- Data is inconsistent between systems due to truncation or misinterpretation of the information.
- The software that interfaces between the two systems fails - and no data is transferred - (this often results in the entire interface failing).

Normally, this is done in two phases:

1. When the interfaces are tested individually during system testing (essentially using a "dummy" system or stub to mimic the distant system or a closed-loop system)
2. When the two systems are tested together with the systems communicating with one another during integration testing.

IV. TESTING FOR REAL-TIME SYSTEMS

The specific characteristics of real-time systems make them a major challenge when testing [7]. The time-dependent nature of real-time applications adds a new difficult element to testing. Not only does the developer have to look at black and white box testing, but also the timing of the data and the parallelism of the tasks. In many situation test data for real-time system may produce errors when the system is in one state but to in others. Comprehensive test cases design methods for real-time systems have not evolved yet. However, a four-stage approach can be put forward:

Task testing: The first stage is to test independently the tasks of the real-time software.

Behavioural testing: Using system models produced with CASE tools the behaviour of the real-time system and examines its actions as a result of external events.

Intertask testing: Once errors in individual tasks and in system behaviour have been observed testing passes to time-related external events.

Systems testing: Software and hardware are integrated and a full set of systems tests are introduced to uncover errors at the software and hardware interface.

V. AUTOMATED TESTING TOOLS

As testing can be 40% of the all effort expanded on the software development process tools that can assist by reducing the time involved is useful. As a response to this various researchers have produced sets of testing tools [8].

Miller described various categories for test tools:

Static analysers: This program-analysis supports "proving" of static allegations-weak statements about program architecture and format.

Code auditors: These special-purpose filters are used to examine the quality of software to ensure that it meets the minimum coding standards.

Assertion processors: These systems tell whether the programmer-supplied assertions about the program are actually met.

Test data generators: These processors assist the user with selecting the appropriate test data.

Output comparators: This tool allows us to contrast one set of outputs from a program with another set to determine the difference among them.

Dunn also identified additional categories of automated tools including:

Symbolic execution systems: This tool performs program testing using algebraic input, instead of numeric data values.

Environmental simulators: This tool is a specialized computer-based system that allows the tester to model the external environment of real-time software and simulate operating conditions.

Data flow analysers: This tool tracks the flow of data through the system and tries to identify data related errors.

VI. CONCLUSIONS

Software testing is more and more considered a problematic method toward better quality. Using testing to locate and correct software defects can be an endless process. Bugs cannot be completely ruled out. Just as the complexity barrier indicates: chances are testing and fixing problems may not necessarily improve the quality and reliability of the software.Software testing accounts for a large percentage of effort in the software development process, but we have only recently begun to understand the subtleties of systematic planning, execution and control.

REFERENCES

[1] Myers, Glenford J., "*The Art of Software Testing*," John Wiley and Sons, pp.145-146, 1979.

[2] Dustin, Elfriede, "*Effective Software Testing*," Addison Wesley, 2002.

[3] B. Kleb&B.Wood, Computational Simulations and the Scientific Method, NASA Langley Research Center, 2005.

[4] A.H. Watson & T.J. McCabe, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, National Institute of Standards and Technology, 1996.

[5] G. Dodig-Crnkovic, Scientific Methods in Computer Science, Department of Computer Science, Mlardalen University, 2002

[6] E. Dustin, Effective Software Testing: 50 Specific ways to improve your testing, Addison Wesley Professionals, 2002.

[7] B. Beizer, Software Testing Techniques, Van Nostrand Reinhold Co.,1990.

[8] M. Fewster and D. Graham, Software Test Automation: Effective use of test execution tools, Addison-Wesley, 1999.

[9] S.M. Baxter, S.W. Day, J.S. Fetrow& S.J. Reisinger, Scientific Software Development Is Not an Oxymoron, PLoSComputBiol 2(9): e87, 2006.