Software Testing

Varun Vashishtha, Tapodhan Singla, Sumeet Singh

Computer Science and Engineering Department,
Maharishi Dayanand University, Rohtak, Haryana, India

Abstract—

The paper attempts to provide a comprehensive view of software testing. Software testing is the process of evaluation of a software item to detect differences between given input and expected output.

Software testing provides a means to reduce errors, cut maintenance and overall software costs. It evaluates quality of a program and also for improving it, by identifying defects and problems.

One of the major problems within software testing area is how to get a suitable set of cases to test a software system. This set should assure maximum effectiveness with the least possible number of test cases. There are now numerous testing techniques available for generating test cases. is the activity where the errors remaining from all the previous phases must be detected. Hence, testing perform a very critical role for software assurance quality.

Keywords— **testing goals, Testing principals, Testing Process, Software quality, Limitations**

1. INTRODUCTION

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. Modern software systems must be extremely reliable and correct. Automatic methods for ensuring software correctness range from static techniques, such as (software) model checking or static analysis, to dynamic techniques, such as testing. All these techniques have strengths and weaknesses: model checking (with abstraction) is automatic, exhaustive, but may suffer from scalability issues. Static analysis, on the other hand, scales to very large programs but may give too many spurious warnings, while testing alone may miss important errors, since it is inherently incomplete. **IEEE STANDARDS**: Institute of Electrical and Electronics Engineers designed an entire set of standards for software and to be followed by the testers. **i.e.** Standard Glossary of Software Engineering Terminology, Standard for Software Quality Assurance Plan, Standard for Software Configuration Management Plan .

What is Software Testing?

Software testing is more than just error detection; Testing software is operating the software under controlled conditions, to (1) verify that it behaves “as specified”; (2) to detect errors, and (3) to validate that what has been specified is what the user actually wanted.

1.**Verification:** It is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements.

2. **Error Detection**: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn‟t or things don‟t happen when they should.

3. **Validation**: Validation looks at the system correctness – i.e. is the process of checking

that what has been specified is what the user actually wanted.

The definition of testing according to the ANSI/IEEE 1059 standard is that testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item. The purpose of testing is verification, validation and error detection in order to find problems – and the purpose of finding those problems is to get them fixed. **Most Common Software problems:** Inadequate software performance, Data searches that yields incorrect results. Incorrect data edits & ineffective data edits, Incorrect coding / implementation of business rules, Incorrect calculation, Incorrect data edits and ineffective data edits, Incorrect processing of data relationship, Incorrect or inadequate interfaces with other systems, Inadequate performance and security controls, Incorrect file handling, Inadequate support of business needs, Unreliable results or performance, Confusing or misleading data, Software usability by end users & Obsolete Software, Inconsistent processing.

1.1Need For Software Testing

Software development involves developing software against a set of requirements. Software testing is needed to verify and validate that the software that has been built has been built to meet these specifications. If not we may probably lose our client. So in order to make it sure, that we provide our client a proper software solution, we go for testing. Testing ensures that what you get in the end is what you wanted to build. We check out if there is any problem, any error in the system, which can make software unusable by the client. This helps in the prevention of errors in a system.

1.2Testing Techniques

Correctness is the minimum requirement of software. Correctness testing will need some type of oracle, to tell the right behavior from the wrong one. The tester may or may not know the inside details of the software module under test. Therefore either white box testing or black box testing can be used.

Correctness testing has following three forms:-

1**)** White box testing
2) Black box testing
3) Grey box Testing

A) White box testing *:*

White box testing is highly effective in detecting and resolving problems, because bugs can often be found before they cause trouble. White box testing is the process of giving the input to the system and checking how the system processes that input to generate the required output. White box testing is also called white box analysis, clear box testing or clear box analysis. White box testing is applicable at integration, unit and system levels of the software testing process.

B) Black box testing :

Black box testing is testing software based on output requirements and without any knowledge of the internal structure or coding in the program. Basically Black box testing is an integral part of “Correctness testing” but its ideas are not limited to correctness testing only. The goal is to test how well the component conforms to the published requirement for the component. Black box testing have little or no regard to the internal logical structure of the system, it only examines the fundamental aspect of the system. It makes sure that input is properly accepted and output is correctly produced.

C) Grey box testing :

The Gray box Testing Methodology is a software testing method used to test software applications. The methodology is platform and language independent. The current implementation of the Gray box methodology is heavily dependent on the use of a host platform debugger to execute and validate the software under test. Recent studies have confirmed that the Gray box method can be applied in real time using software executing on the target platform. Grey box testing techniques combined the testing methodology of white box and black box. Grey box testing technique is used for testing a piece of software against its specifications but using some knowledge of its internal working as

well. The understanding of internals of the program in grey box testing is more than black box testing, but less than clear box testing.

2. TESTING GOALS

A goal is a projected state of affairs that a person or system plans or intends to achieve. A goal has to be accomplishable and measurable. It is good if all goals are interrelated. In testing we can describe goals as intended outputs of the software testing process. Software testing has following goals:-

A .Verification:

Most misunderstood about testing is the primary objective. If you think it is to find defects then you are wrong. Defects will be found by everybody using the software. Testing is a quality control measure used to verify that a product works as desired. Testing provides a status report of the actual product in comparison to requirements (written and implicit). At its simplest this is a pass/fail listing of product features; at detail it includes confidence numbers and expectations of defect rates throughout the software.
This is important since a tester can hunt bugs forever yet not be able to say whether the product is fit for release. Having a multitude of defect reports is of a little use if there is no method by which to value them. A corporate policy needs to be in place regarding the quality of the product. It must state what conditions are required to release the software. The tester's job is to determine whether the software fulfils those conditions.

B. Unbiased :

Tests must balance the written requirements, real-world technical limitations, and user expectations. Regardless of the development process being employed there will be a lot unwritten or implicit requirements. It is the job of the tester to keep all such requirements in mind while testing the software. A tester must also realize they are not a user of the software, they are part of the development team. Their personal opinions are but one of many considerations. Bias in a tester invariably leads to a bias in coverage.
The end user's viewpoint is obviously vital to the success of the software, but it isn't all that matters. If the needs of the administrators can't be met the software may not be deployable. If the needs of the support team aren't met, it may be unsupportable. If the needs of marketing can't be met, it may be unsellable. The programmers also can't be ignored; every defect has to be prioritized with respect to their time limits and technical constraints.

C. Deterministic :

The discovery of issues should not be random. Coverage criteria should expose all defects of a decided nature and priority. Furthermore, later surfacing defects should be identifiable as to which branch in the coverage it would have occured, and can thus present a definite cost in detecting such defects in future testing.
This goal should be a natural extension to having traceable tests with priority coverage. It reiterates that the testing team should not be a chaotic black box. Quality control is a well structured, repeatable, and predictable process. Having clean insight into the process allows the business to better guage costs and to better direct the overall development.

D. Traceable :

Exactly what was tested, and how it was tested, are needed as part of an ongoing development process. In many environments such proof of activities are required as part of a certification effort, or simply as a means to eliminate duplicate testing effort. This shouldn't mean extra documentation, it simply means keeping your test plans clear enough to be reread and understood.
You will have to agree on the documentation methods; every member of the team should not have their own. Not all features should be documented the same way however: several different methods will likely be employed. Unfortunately there aren't a lot of commonly

agreed principles in this area, so in a way you're kind of on your own.

3. TESTING PRINCIPLES

A principle is an accepted rule or method for application in action that has to be, or can be desirably followed. Testing Principles offer general guidelines common for all testing which assists us in performing testing effectively and efficiently.

Principles for software testing are:

A . Testing shows presence of defects

Testing can show that defects are present ,but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

B. Exhaustive testing is impossible

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, we use risks and priorities to focus testing efforts.

C. Early testing

Testing activities should start as early as possible in the software or system development life cycle and should be focused on defined objectives. Cost of finding and fixing the error increases over time.

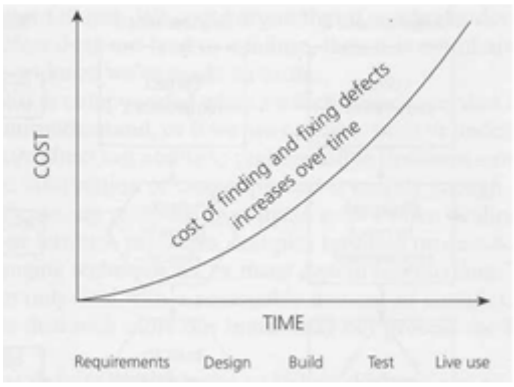

Figure 1

D. Defect clustering

A small number of modules contain most of the defects discovered during pre release testing or show the most operational failures.

E. Pesticide paradox

If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new bugs. To overcome this 'pesticide paradox', the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

H. Testing is context dependent.

Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.

G. Absence-of-errors fallacy

Finding and fixing defects does not help if the system built is unusable and does not fulfils the users' needs and expectations.

4. SOFTWARE TESTING LIFECYCLE-PHASES

A. Requirements study

 Testing Cycle starts with the study of clients requirements.

 Understanding of the requirements is very essential for testing the product.

B. Test Case Design and Development

 Component Identification
 Test Specification Design
 Test Specification Review

C. Test Execution

 Code Review
 Test execution and evaluation
 Performance and simulation

D. Test Closure

 Test summary report
 Project De-brief
 Project Documentation

E. Test Process Analysis

 Analysis done on the reports and improving the application‟s performance by implementing new technology and additional features.

5. TESTING PROCESS

Traditional waterfall development model

A common practice of software testing is that testing is performed by an independent group of testers after the functionality is developed, before it is shipped to the customer. This practice often results in the testing phase being used as buffer to compensate for project delays, thereby compromising the time devoted to testing. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

Agile or Extreme development model

In contrast, some emerging software disciplines such and movement, adhere to a model. In this process, are written first, by the (often with in the extreme programming methodology). Of course these tests fail initially; as they are expected to. Then as code is written it passes incrementally larger portions of the test suites. The test suites are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed. Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process). The ultimate goal of this test process is to achieve where software updates can be published to the public frequently. This methodology increases the testing effort done by development, before reaching any formal testing team. In some other development models, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

Top-down and bottom-up

Bottom Up Testing is an approach to integrated testing where the lowest level components (modules, procedures, and functions) are tested first, then integrated and used to facilitate the testing of higher level components. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. The process is repeated until the components at the top of the hierarchy are tested. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module. In both, and drivers are used to stand-in for missing components and are replaced as the levels are completed

.

A sample testing cycle

Although variations exist between organizations, there is a typical cycle for testing. The sample below is common among organizations employing model. The same practices are commonly found in other development models, but might not be as clear or explicit.

Testing should begin in the requirements phase of the during the design phase, testers work to determine what aspects of a design are testable and with what parameters those tests work.

- **Test planning** creation. Since many activities will be carried out during testing, a plan is needed.
- **Test development**: Test procedures, test datasets, test scripts to use in testing software.
- **Test execution**: Testers execute the software based on the plans and test documents then report any errors found to the development team.
- **Test reporting**: Once testing is completed, testers generate metrics and make final reports on their and whether or not the software tested is ready for release.
- **Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be assigned, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.
- **Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing team.
- **Regression testing**: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.
- **Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

6. LIMITATIONS

Limitation is a principle that limits the extent of something. Testing also has some limitations that should be taken into account to set realistic expectations about its benefits. In spite of being most dominant verification technique, software testing too has following limitations:

1. Testing can be used to show the presence of errors, but never to show their absence! . It can only identify the known issues or errors. It gives no idea about defects still uncovered. Testing cannot guarantee that the system under test is error free.

2. Testing provides no help when we have to make a decision to either "release the product with errors for meeting the deadline" or to "release the product late compromising the deadline".

3. Testing cannot establish that a product functions properly under all conditions but can only establish principles, limitations and concepts. Already lot of work has been done in this field, and even continues today. Implementing testing principles in real world software development, to accomplish testing goals to maximum extent keeping in consideration the testing limitations will validate the research and also will pave a way for future research that it does not function properly under specific conditions.

4. Software testing does not help in finding root causes which resulted in injection of defects in the first place. Locating root causes of failures can help us in preventing injection of such faults in future.

7. CONCLUSION

Software testing is a vital element in the SDLC and can furnish excellent results if done properly and effectively. Unfortunately, Software testing is often less formal andrigorous than it should, and a main reason for that is because we have struggled to define best practices, methodologies ,principles, standards for optimal software testing. To perform testing effectively and efficiently, everyone involved with testing should be familiar with basic software testing goals, principles, limitations and concepts. Already lot of work has been done in this field, and even continues today. Implementing testing principles in real world software development, to accomplish testing goals to maximum extent keeping in consideration the testing limitations will validate the research and also will pave a way for future research.

8. REFERENCES

[1] Antonia Bertolina, ‖Software Testing Research and Practice‖, Proceedings of the abstract state machines 10th international conference on Advances in theory and practice,1-21,2003.

[2] “Foundation of software testing”-Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black.

[3]. Nick Jenkins. ―A Software Testing Primer‖, 2008.

[4] Ian Somerville, ‖Software Engineering‖, Addison-Wesley,
2001.

[5].Miller, William E. Howden, "Tutorial, software testing & validation techniques", IEEE Computer Society Press,
1981.

[6] Peter Sestoft,‖ Systematic software testing‖, Version 2, 2008-02-25.

[7] Shari Lawrence Pfleeger, ―Software Engineering, Theory
and Practice‖, Pearson Education, 2001.