An comprehensive study of software Testing and its Techniques
Sunil, HOD & Assistant Professor, Dept. Of CSE, SSIET, Dinanagar

ABSTRACT
IEEE definition of software testing is process of executing the program with specific intent of finding errors. Software testing remains the primary concern used to gain consumer's confidence in the software. Ideally, testing of software guarantees the absence of errors in the software, but in reality it only discloses the presence of software errors, but never guarantees their absence. Even, systematic testing cannot guarantee the absence of errors, which are detected by discovering their effect. Software testing also identifies important defects, flaws, or errors in the application code that must be fixed. The modifier "important" in the previous sentence is, well, important because defects must be categorized by severity. In this chapter, we will provide introduction to various basic principle and key terms which are used in software testing.

Keywords: software testing; Regression Testing; White Box Testing; Black Box Testing; Equivalence Partitioning

1. INTRODUCTION
Software testing is a process of verifying and validating that a software application or program. it meets the business and technical requirements that guided its design and development, and Works as expected. The quality assurance aspect of software development documenting the degree to which the developers followed corporate standard processes or best practices is not addressed in this thesis because assuring quality is not a responsibility of the testing team but it also depend on testing strategy . The testing team cannot guarantee quality; they can only measure it, although it can be argued that doing things like designing tests before coding begins, will improve quality because the coders can then use that information while thinking about their designs and during coding and debugging.

Software testing has three main purposes: verification, validation, and defect finding.
1. The verification process confirms that the software meets its technical specifications. A specification is a description of a function in terms of a measurable output value given a specific input value under specific preconditions.
2. The validation process confirms that the software meets the business requirements. A defect is a variance between the expected and actual result.
3. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

2. TESTING PRINCIPLES
Following are principles of Software Testing [1]:
1. All tests should be traceable to customer requirements. The objective of system testing is to uncover errors. It follows that the most severe defects are those that cause the program to fail to meet its requirements.
2. Tests should be planned long before testing begins. Test planning can begin as soon as the requirements model is complete. Detailed definition of test cases can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.
3. Testing should begin "in the small" and progress toward testing "in the large". The first test planned and executed generally focus on individual program modules. As testing progresses, testing shifts focus in an attempt to find errors in integrated clusters of modules and ultimately in the entire system.
4. Exhaustive testing is not possible. The number of path permutations for even a moderately sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing.
5. To be most effective, testing should be conducted by an independent third party. By most effective, means testing that has the highest probability of finding errors. For this reason, the software engineer who created the system is not the best person to conduct all tests for the software.

3. TESTING PROCESS
The IEEE 829 standard [2] describes a framework within which the entire testing process can be managed. The framework allows easy communication between members of a testing project, organizes the testing process, and outlines the documents that should be made part of any compliant testing process.

3.1 Test Plan
Test plan describes the scope, approach, resources, and schedule of testing activities in a given project, and identifies the items to be tested, the features of those items to be tested, the individual testing tasks that are to be performed, and personnel responsible for those tasks, along with the risks associated with the plan.
Test plan should have the following structure:
1. Test plan identifier
2. Introduction

3. Items to be tested

4. Features to be tested

5. Features not to be tested

6. Testing approach

7. Test item pass and fail criteria

8. Test suspension criteria

9. Test resumption requirements

10. Testing tasks

11. Environmental needs

12. Responsibilities

13. Staffing and training requirements

14. Schedule

15. Risks and contingencies

16. Approvals

4. LEVEL OF TESTING

Testing is involved in every stage of software life cycle, but the testing done at each level of software development is different in nature and has different objectives.

4.1 Unit Testing

It is done at the lowest level. It tests the basic unit of software, which is the smallest testable piece of software, and is often called “unit”, “module”, or “component” interchangeably.

4.2 Integration Testing

It is performed when two or more tested units are combined into a larger structure. The test is often done on both the interfaces between the components and the larger structure being constructed, if its quality property cannot be assessed from its components.

4.3 System Testing

It tends to affirm the end-to-end quality of the entire system. System test is often based on the functional requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability, are also checked.

4.4 Acceptance Testing

It is done when the completed system is handed over from the developers to the customers or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors.

4.5 Regression Testing

It is any type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes, such as enhancements, patches or configuration changes, have been made to them. The intent of regression testing is to ensure that a change, such as a bug fix, did not introduce new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software. Common methods of regression testing include re-running previously run tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. Regression testing can be used to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.
5. TYPES OF TESTING
Software testing methods are traditionally divided into white and black box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

5.1 White Box Testing
White box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality. In white box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in circuit testing (ICT). While white box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system level test. Though this, method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements. White box test design techniques include, control flow testing, data flow testing, branch testing, path testing, statement coverage and decision coverage.

5.2 Black Box Testing
Black box testing is a method of software testing that tests the functionality of an application as opposed to its internal structures or workings. This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well. Specific knowledge of the application's code and internal structure and programming knowledge in general is not required. The tester is only aware of what the software is supposed to do but not how i.e. when he enters a certain input, he gets a certain output without being aware of how the output was produced in the first place. Maximum edges of an equivalence partition are tested. The values could be either input or output ranges of a software component. Since these boundaries are common locations for errors that result in software faults they are frequently exercised in test cases.

6. TEST CASE DESIGN TECHNIQUES
There are mainly six techniques to design test case. They are as follows:
Use Case: In software and systems engineering, a use case is a list of steps, typically defining interactions between a role (known in UML as an “actor”) and a system, to achieve a goal. The actor can be a human or an external system.

6.1 Decision Tables: Decision tables are a precise yet compact way to model complicated logic. Decision tables, like flowcharts and if-then-else and switch-case statements, associate conditions with actions to perform, but in many cases do so in a more elegant way.

6.2 All Pair Shortest Path: All-pairs testing or pair wise testing is a combinatorial software testing method that, for each pair of input parameters to a system (typically, a software algorithm), tests all possible discrete combinations of those parameters. Using carefully chosen test vectors, this can be done much faster than an exhaustive search of all combinations of all parameters, by parallelizing the tests of parameter pairs. The number of tests is typically O(nm), where n and m are the number of possibilities for each of the two parameters with the most choices.

6.3 State Transition Table: In automata theory and sequential logic, a state transition table is a table showing what state (or states in the case of a nondeterministic finite automation) a finite semi automation or finite state machine will move to, based on the current state and other inputs. A state table is essentially a truth table in which some of the inputs are the current state, and the outputs include the next state, along with other outputs. A state table is one of many ways to specify a state machine, other ways being a state diagram, and a characteristic equation.

6.4 Equivalence Partitioning: Equivalence partitioning (also called Equivalence Class Partitioning or ECP) is a software testing technique that divides the input data of a software unit into partitions of data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed.

6.5 Boundary Value Analysis: Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values. Values on the minimum and maximum edges of an equivalence partition are tested. The values could be either input or output ranges of a software component. Since these boundaries are common locations for errors that result in software faults they are frequently exercised in test cases.

7. REFERENCES