Software Testing

MUSHTAQ-AHMAD-DAR

DEPARTMENT-OF- I.T

MTECH- I.T

54-CHANDIANA GANDERBAL, J&K, INDIA

mushtaqdar@hotmail.com

1. ABSTRACT

Software testing is the process of running an application with the intent of finding software bugs (errors or other defects). Software demand has pushed the quality assurance of developed software towards new heights. It has been considered as the most critical stage of the software development life cycle. Testing can analyze the software item to identify the disparity between actual and prescribed conditions and to assess the characteristics of the software. Software testing leads to minimizing errors and cut down software costs. For this purpose, we discuss various software testing techniques and strategies. This paper aims to study diverse as well as improved software testing techniques for better quality assurance purposes.

2. INTRODUCTION

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.

4. Verification is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements. [Verification: Are we building the system right?]

5. 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.

6. 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.

Terminology:

- Mistake A human action that produces an incorrect result.

- Fault [or Defect] An incorrect step, process, or data definition in a program.

- Failure The inability of a system or component to perform its Required function within the specified performance requirement.

- Error The difference between a computed, observed, or Measured value or condition and the true, specifie d, or theoretically correct value or condition.

- Specification A document that specifies in a complet e, precise, Verifiable manner, the requirements, desi gn,

Definition And The Goal Of Testing Process of cr eating a program consists of the following phases:

1. defining a problem;

2. designing a program;

3. building a program;

4. analyzing performances of a program, and 5 final arr anging of a product.

According to this classification, software testing is a component of the third phase, and means checking if a program for specified inputs gives correctly and ex pected results.

Software testing is an important component of softw are quality assurance, and many software organizatio ns are spending up to 40% of their resources on testin g. For life-
critical software (e.g., flight control) testing can be hi ghly expensive. Because of that, many studies about risk analysis have been made. This term means the pr obability that

a software project will experience undesirable events , such as schedule delays, cost overruns, or outright c ancellation and more about this in. There are a many definitions of software testing, but one can shortly de fine that as:

A process of executing a program with the goal of fin ding errors. So, testing means that one inspects beha vior of a program on a finite set of test cases (a set of i nputs, execution preconditions, and expected outco mes developed for a particular objective, such as to e xercise a particular program path or to verify compli ance with a specific requirement, for which valued in puts always exist.

Testing is an activity performed for evaluating softw are quality and for improving it. Hence, the goal of te sting is systematical detection of different classes of errors error can be defined as a human action that pro duces an incorrect result, in a minimum amount of ti me and with a minimum amount of effort.

Figure 1: Test Information Flow

- Good test cases –
have a good chance of finding an yet undiscovered e rror; and

- Successful test cases – uncovers a new error.

A good test case is one which:

- Has a high probability of finding an error; Is not redu ndant;

- Should be best of breed;

- Should not be too simple or too complex.

3. LITERATURE REVIEW

In this section, we will outline the previous works of Software Testing. According to "The Theory of Soft ware Testing", testing is the means of showing the pr esence of errors in the program which can either be p erformed manually or automatically. It also includes the basic terminology of testing such as automated te sting, failure, testing team, and wrong test case select ion. This paper focuses on the process that should be followed to test the performance of new software and the entire system. The conclusion of the article is the complete view of software testing, preliminary testin g, and user acceptance testing.

4. TESTING METHODS

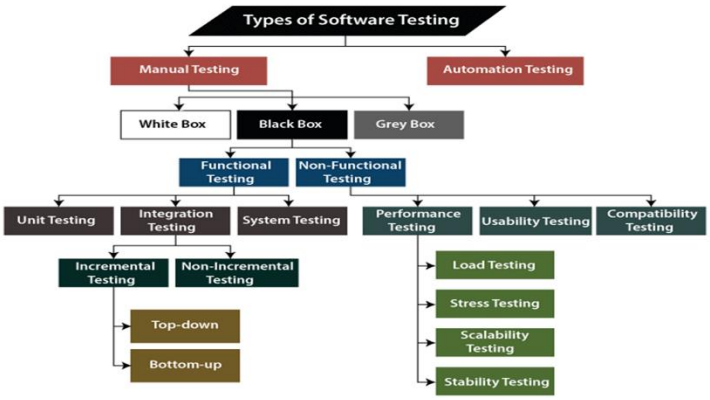
Test cases are developed using various test techniques to achieve more effective testing. By this, software completeness is provided and conditions of testing which get the greatest probability of finding errors are chosen. So, testers do not guess which test cases to chose, and test techniques enable them to design testing conditions in a systematic way. Also, if one combines all sorts of existing test techniques, one will obtain better results rather if one uses just one test technique.

Software can be tested in two ways, in another words, one can distinguish two different methods:

1. Black box testing, and
2. White box testing.

White box testing is highly effective in detecting and resolving problems, because bugs can often be found before they cause trouble. We can shortly define this method as testing software with the knowledge of the internal structure and coding inside the program. White box testing is also called white box analysis, clear box testing or clear box analysis. It is a strategy for software debugging (it is the process of locating and fixing bugs in computer program code or the engineering of a hardware device, in which the tester has excellent knowledge of how the program components interact. This method can be used for Web services applications, and is rarely practical for debugging in large systems and networks. Besides, in white box testing is considered as a security testing (the process to determine that an information system protects data and maintains functionality as intended, method that can be used to validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities.

Black box testing is testing software based on output requirements and without any knowledge of the internal structure or coding in the program. In another words, a black box is any device whose workings are not understood by or accessible to its user. For example, in telecommunications, it is a resistor connected to a phone line that makes it impossible for the telephone company's equipment to detect when a call has been answered a particular function, but in the financial world, it is a computerized trading system that doesn't make its rules easily available. In recent years, the third testing method has been also considered gray box testing. It is defined as testing software while already having some knowledge of its underlying code or logic.It is based on the internal data structures and algorithms for designing the test cases more than black box testing but less than white box testing. This method is important when conducting integration testing between two modules of code written by two different developers, where only interfaces are exposed for test. Also, this method can include reverse engineering to determine boundary values. Gray box testing is non-intrusive and unbiased because it doesn't require that the tester have access to the source code. The main characteristics and comparison between white box testing and black box testing are follows.



1. Black Box Testing Versus White Box Testing Black Box Testing:

Performing the tests which exercise all functional requirements of a program;

Finding the following errors:

- Incorrect or missing functions;

- Interface errors;
- Errors in data structures or external database access;
- Performance errors;
- Initialization and termination errors.

Advantages of this method:

- The number of test cases are reduced to achieve reas onable testing;
- The test cases can show presence or absence of class es of errors.

White Box Testing:

Considering the internal logical arrangement of soft ware;

- The test cases exercise certain sets of conditions and loops;
- Advantages of this method:
- All independent paths in a module will be exercised a t least once;
- All logical decisions will be exercised;
- All loops at their boundaries will be executed;
- Internal data structures will be exercised to maintain their validity.

3. SOFTWARE TESTING STRATEGIES

Software Testing strategies provide a method of inte grating software test case design methods into a well -
planned Series of steps that can result in the successf ul construction of software. It provides the road map for testing. The software testing Strategy should be p liable enough to develop a customized testing approa ch. The software testing strategy is actually produced by project managers, software engineers, and testing specialists. There are four different types of software testing strategies: 1) Unit testing 2) Integration testin g 3) Acceptance/Validation testing 4) System testing 1. Unit testing Unit is the smallest testable part, i.e. t he most modest collection of lines of code which can be tested. Unit testing is done by the developer as the proper knowledge about the core programming desig ning is required. Generally, unit testing is considered as a white-
box testing class because it is partisan to evaluate the code as implemented rather than assessing conforma nce to some set of requirements. Benefits of Unit Tes ting: 1) Cost-
effective testing technique. 2) Simple testing techniq ue because the smallest testable unit of the code is tes ted here. 3) Individual parts are tested when necessar y, without waiting for another part of the system. 4) Unit testing can be performed in parallel by fixing pr oblems simultaneously by many engineers. 5) Detect ion and removal of defects are much costeffective co mpared to other levels of testing. 6) Be able to take a dvantage of several formal testing approaches availa ble for unit testing. 7) Clarify debugging by limiting t o a small unit the possible code areas in which to sear ch for bugs. 8) Be able to test internal logic that is not easily reached by external inputs in the broader integ rated systems. 9) Attain a high level of structural cov erage of the code. 10) When debugging severe proble ms, it avoids lengthy compile-build-
debug cycles. Unit testing techniques: Unit testing us es several effective testing techniques. The testing te chniques categorize into three types: i. Functional Te sting ii. Structural Testing iii. Heuristic or Intuitive T esting

4. GENERAL CLASSIFICATION OF TEST TEC HNIQUES

In this paper, the most important test techniques are s hortly described, as it is shown Techniques

- Equivalence Partitioning Summary: equivalence cla ss

This technique divides the input domain of an progra m onto equivalence classes.

Equivalence classes set of valid or invalid states for i nput conditions, and can be defined in the following way:

1. An input condition specifies a range one valid and tw o invalid equivalence classes are defined;
2. An input condition needs a specific value one valid a nd two invalid equivalence classes are defined;
3. An input condition specifies a member of a set one v alid and one invalid equivalence class are defined
4. An input condition is Boolean one valid and one inva lid equivalence class are defined.

Well, using this technique, one can get test cases whi ch identify the classes of errors.

- Boundary Value Analysis

Summary: complement equivalence Partitioning this technique is like the technique Equivalence Partition ing, except that for creating the test cases beside inpu t domain use output domain.

One can form the test cases in the following way:

2. An input condition specifies a range bounded by valu es a and bt est cases should be made with values just above and just below a and b, respectively;
3. An input condition specifies various values test cases should be produced to exercise the minimum and ma ximum numbers;
4. Rules 1 and 2 apply to output conditions;

If internal program data structures have prescribed b oundaries, produce test cases to exercise that data str ucture at its boundary.

Comparison Testing Summary: independent version s of an application In situations when reliability of so ftware is critical, redundant software is produced. In that case one uses this technique.

Fuzz Testing Summary: random input

Fuzz testing is often called fuzzing, robustness testin g or negative testing. It is developed by Barton Mille r at the University of Wisconsin in 1989. This techni que feeds random input to application. The main char acteristic of fuzz testing are:

- the input is random;
- the reliability criteria: if the application crashes or ha ngs, the test is failed;
- fuzz testing can be automated to a high degree.

A tool called fuzz tester which indicates causes of fo unded vulnerability, works best for problems that

can cause a program to crash such as buffer overflow , cross-
site scripting, denial of service attacks, format bug an d SQL injection. Fuzzing is less effective for spywar e, some viruses, worms, Trojans, and keyloggers. Ho wever, fuzzers are most effective when are used toge ther with extensive black box testing techniques.

Model-based testing

Model-
based testing is automatic generation of efficient test procedures/vectors using models of system requirem ents and specified functionality.

In this method, test cases are derived in whole or in p art from a model that describes some aspects of the s ystem under test. These test cases are known as the a bstract test suite, and for their selection different tech niques have been used:

- generation by theorem proving;

- generation by constraint logic programming;
- generation by model checking;
- generation by symbolic execution;
- generation by using an event-flow model;

Basis Path Testing Summary: basis set, independent path, flow graph, cyclomatic complexity, graph matrix, link weight

If one uses this technique, one can evaluate logical complexity of procedural design. After that, one can employ this measure for description basic set of execution paths.

Based on the software engineers intuition and experience:

2. Ad hoc testing Test cases are developed basing on the software engineers skills, intuition, and experience with similar programs;
3. Exploratory testing This testing is defined like simultaneous learning, which means that test are dynamically designed, executed, and modified.

Specification-based techniques:

4. Equivalence partitioning;
5. Boundary-value analysis;
6. Decision table Decision tables represent logical relationships between inputs and outputs (conditions and actions), so test cases represent every possible combination of inputs and outputs;
7. Finite-state machine-based Test cases are developed to cover states and transitions on it;
8. Testing from formal specifications

The formal specifications (the specifications in a formal language) provide automatic derivation of functional test cases and a reference output for checking test results;

9. Random testing Random points are picked within the input domain which must be known, so test cases are based on random.

5. CONCLUSION

Software testing is a component of software quality control (SQC). SQC means control the quality of software engineering products, which is conducting using tests of the software system.

These tests can be: unit tests (this testing checks each coded module for the presence of bugs), integration tests (interconnects sets of previously tested modules to ensure that the sets behave as well as they did as independently tested modules), or system tests (checks that the entire software system embedded in its actual hardware environment behaves according to the requirements

- Testing can show the presence of faults in a system; it cannot prove there are no remaining faults.
- Component developers are responsible for component testing; system testing is the responsibility of a separate team.
- Integration testing is testing increments of the system; release testing involves testing a system to be released to a customer.
- Use experience and guidelines to design test cases in defect testing.
- Interface testing is designed to discover defects in the interfaces of composite components.
- Equivalence partitioning is a way of discovering test cases –

all cases in a partition should behave in the same wa y.

- Structural analysis relies on analysing program and d eriving tests from this analysis.

- Test automation reduces testing costs by supporting t he test process with a range of software tools.

6. REFERENCES

1. Stacey, D. A., Software Testing Techniques

2. Guide to the Software Engineering Body of Knowle dge, Swebok A project of the IEEE Computer Societ y Professional Practices Committee, 2004.

3. Software Engineering: A Practitioners Approach, 6/e ; Chapter 14: Software Testing Techniques, R.S.Pres sman & Associates, Inc., 2005.

4. Wikipedia, The Free Encyclopedia, http://