ISBN: 978-93-83758-09-8

REVIEW PAPER ON SOFTWARE TESTING

Komal
M-TECH Scholar

komalpopli8@gmail.com

I. 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.

1. 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?*]

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 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 analysing 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, specified, or theoretically correct value or condition.
- Specification – A document that specifies in a complete, precise, Verifiable manner, the requirements, design,

Definition And The Goal Of Testing Process of creating 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 arranging 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 expected results.

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

ISBN: 978-93-83758-09-8

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

A process of executing a program with the goal of finding errors. So, testing means that one inspects behavior of a program on a finite set of test cases (a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement, for which valued inputs always exist.

Testing is an activity performed for evaluating software quality and for improving it. Hence, the goal of testing is systematical detection of different classes of errors *error can be defined as a human action that produces an incorrect result*, in a minimum amount of time and with a minimum amount of effort.

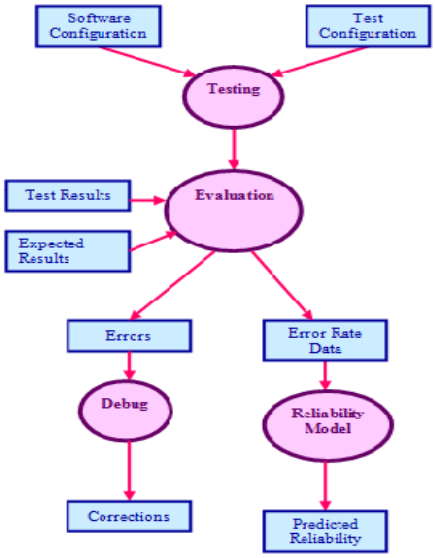

Figure 1: Test Information Flow

- *Good test cases* - have a good chance of finding an yet undiscovered error; 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 redundant;
- Should be "best of breed";
- Should not be too simple or too complex.

II. 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.

ISBN: 978-93-83758-09-8

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.

2.1. 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 reasonable testing;
- The test cases can show presence or absence of classes of errors.

White Box Testing:
Considering the internal logical arrangement of software;

- The test cases exercise certain sets of conditions and loops;
- Advantages of this method:
- All independent paths in a module will be exercised at 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.

III. GENERAL CLASSIFICATION OF TEST TECHNIQUES

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

3.1. Equivalence Partitioning Summary: *equivalence class*

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

Equivalence classes – set of valid or invalid states for input conditions, and can be defined in the following way:
1. An input condition specifies a range → one valid and two invalid equivalence classes are defined;
2. An input condition needs a specific value → one valid and two invalid equivalence classes are defined;
3. An input condition specifies a member of a set → one valid and one invalid equivalence class are d efined
4. An input condition is Boolea→n one valid and one invalid equivalence class are defined.
Well, using this technique, one can get test cases which identify the classes of errors.

3.2. Boundary Value Analysis
Summary: complement equivalence Partitioning this technique is like the technique Equivalence Partitioning, except that for creating the test cases beside input domain use output domain.
One can form the test cases in the following way:
1. An input condition specifies a range bounded by values a and→ bt est cases should be made with values just above and just below a and b, respectively;
2. An input condition specifies various values → test cases should be produced to exercise the minimum and maximum numbers;
3. Rules 1 and 2 apply to output conditions;
If internal program data structures have prescribed boundaries, produce test cases to exercise that data structure at its boundar.

Comparison Testing Summary: *independent versions of an application* In situations when reliability of software 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 testing or negative testing. It is developed by Barton Miller at the University of Wisconsin in 1989. This technique feeds random input to application. The main characteristic of fuzz testing, according to the [26] are:

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

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

ISBN: 978-93-83758-09-8

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

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

In this method, test cases are derived in whole or in part from a model that describes some aspects of the system under test. These test cases are known as the abstract test suite, and for their selection different techniques 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 engineer's intuition and experience:
1. Ad hoc testing – Test cases are developed basing on the software engineer's skills, intuition, and experience with similar programs;
2. Exploratory testing – This testing is defined like simultaneous learning, which means that test are dynamically designed, executed, and modified.

Specification-based techniques:
1. Equivalence partitioning;
2. Boundary-value analysis;
3. 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;
4. Finite-state machine-based – Test cases are developed to cover states and transitions on it;
5. 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;
6. Random testing – Random points are picked within the input domain which must be known, so test cases are based on random.

IV. 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 way.
- Structural analysis relies on analysing a program and deriving tests from this analysis.
- Test automation reduces testing costs by supporting the test process with a range of software tools.

REFERENCES

1. Stacey, D. A., "Software Testing Techniques"
2. Guide to the Software Engineering Body of Knowledge, Swebok – A project of the IEEE Computer Society Professional Practices Committee, 2004.
3. "Software Engineering: A Practitioner's Approach, 6/e; Chapter 14: Software Testing Techniques", R.S.Pressman & Associates, Inc., 2005.
4. Wikipedia, The Free Encyclopedia, http://