REVIEW PAPER ON SOFTWARE TESTING

Pooja Yadav, Pooja Kumari
Dronacharya College of Engineering, Gurgaon.

Abstract: - Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user. Software Testing is the process of executing a program to locate an error. This paper describes most commonly used software testing techniques for detecting error. They are: white box testing, black box testing and grey box testing.

Index Terms— Black box, Detection, White box, Prevention, Software testing, Grey box

1 Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.[1] Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

It involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects).

Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.[1]

1.1. Objective of Testing

The objective of testing is to find problems and fix them to improve quality. Software testing typically represents 40% of a software development budget.

There are four main objectives of software testing:

a) Demonstration: It demonstrates functions under special conditions and shows that products are ready for integration or use.

b) Detection: It discovers defects, errors and deficiencies. It determines system capabilities and limitations, quality of components, work products and the system.

c) Prevention: It provides information to prevent or reduce the number of errors clarify system specifications and performance. Identify ways to avoid risk and problems in the future.

d) Improving Quality: By doing effective testing, we can minimize errors and hence improve the quality of software. [2]

1.2. Black Box Testing

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The testers are only aware of what the software is supposed to do, not how it does it.[3]. This is purely done based on customers view...
point only tester knows the set of inputs and predictable outputs.[4][5]

![Blackbox Testing Diagram](image)

Fig 1. blackbox testing

Black-box testing methods include:

1) Equivalence Partitioning: This technique divides the input domain of a program into equivalence classes from which test cases can be derived, so it can reduce the number of test cases.

2) Boundary Value Analysis: It focuses on testing at boundaries, or where the extreme boundary values are chosen. It includes minimum, maximum, just inside/outside boundaries, error values and typical values.

3) Fuzzing: This technique feeds random input to application. It is used for finding implementation bugs, using malformed/semi-malformed data injection in an automated or semi-automated session.

4) Cause-Effect Graph: In this technique, testing begins by creating a graph and establishing the relation between effect and its causes.

5) Orthogonal Array Testing: It can be applied where input domain is very small, but too large to accommodate exhaustive testing.

6) All Pair Testing: In this technique, test cases are designed to execute all possible discrete combinations of each pair of input parameters. Its main objective is to have a set of test cases that covers all the pairs.

7) State Transition Testing: This type of testing is useful for testing state machine and also for navigation of graphical user interface.

Advantages:

1. Testers need not to have knowledge on specific programming language.
2. Testing is done from user’s point of view.
3. It helps to expose any ambiguities or inconsistencies in the requirement specifications.[6]
4. Programmer and tester both are independent of each other.

Disadvantages:

1. Test cases are hard to design without clear specifications.
2. Chances of having repetition of tests that are already done by programmer.
3. Some parts of back end are not tested at all.

1.3. White-Box Testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. 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.

Techniques used in white-box testing include:

- API testing (application programming interface) – testing of the application using public and private APIs
• Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)

• Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies

• Mutation testing methods

• Static testing methods

Code coverage tools can evaluate the completeness of a test suite that was created with any method, including black-box testing. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.[7] Code coverage as a software metric can be reported as a percentage for:

• Function coverage, which reports on functions executed

• Statement coverage, which reports on the number of lines executed to complete the test

100% statement coverage ensures that all code paths or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

Advantages:

1. It reveals error in hidden code by removing extra lines of code.

2. Maximum coverage is attained during test scenario writing.[8]

3. Developer carefully gives reasons about implementation.

Disadvantages:

1. A skilled tester is needed to carry out this testing because knowledge of internal structure is required.

2. Many paths will remain untested as it is very difficult to look into every nook and corner to find out hidden

1.4. Grey-box testing

Grey-box testing (American spelling: gray-box testing) involves having knowledge of internal data structures and algorithms for purposes of designing tests, while executing those tests at the user, or black-box level. The tester is not required to have full access to the software's source code.[9] Following are some subtypes of gray-box testing:[10]-[17]

State-Model-Testing: It tests each method of an object, transition & transition paths at each state of an object.

Class-Diagram Testing: It tests all the derived classes of the base class.

Sequence-Diagram Testing: It tests all the methods occurring in sequence diagram.

It includes two main types as following:

Thread-Based Testing: In this all the classes of a single Use Case are integrated together and then testing is performed. This process is going on until all the classes of all Use Cases have been considered.

Use-Based Testing: It performs the testing on the classes that either need the services from other classes or does not need any services

Advantages:

1. It provides combined benefit of black box and white box testing techniques.

2. In grey box testing, tester can design excellent test scenarios.

3. Unbiased testing

4. Create an intelligent test authoring.

Disadvantages:

1. Test coverage is limited as the access to source code is not available.

2. Many program paths remain untested.
3. The test cases can be redundant.[8]

1.5. Testing Level

There are generally four recognized levels of tests: unit testing, integration testing, component interface testing, and system testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model.[18] Other test levels are classified by the testing objective.[18]

1.5.1. Unit testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.[19]

1.5.2. Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.[20]

1.5.3. Component interface testing

The practice of component interface testing can be used to check the handling of data passed between various units, or subsystem components, beyond full integration testing between those units[21][22]The data being passed can be considered as "message packets" and the range or data types can be checked, for data generated from one unit, and tested for validity before being passed into another unit. One option for interface testing is to keep a separate log file of data items being passed, often with a timestamp logged to allow analysis of thousands of cases of data passed between units for days or weeks. Tests can include checking the handling of some extreme data values while other interface variables are passed as normal value[21] Unusual data values in an interface can help explain unexpected performance in the next unit. Component interface testing is a variation of black-box testing [22]with the focus on the data values beyond just the related actions of a subsystem component.

1.5.4. System testing

System testing, or end-to-end testing, tests a completely integrated system to verify that it meets its requirements[23] For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

CONCLUSION

- Software Testing is and will forever be a fundamental activity of Software Engineering.
- We will never find a test approach that is guaranteed to deliver a “perfect” product, whichever is the effort we employ.
- Software Testing is a trial-and-error methodology.
- Software Testing can never be satisfactorily completed because of the input domain from customer.
- Testing costs can be reduced by using different test automation tools.
- Testing helps to detect the errors in system but does not prove that system is error free.
- Testing Object Oriented Programs provides new features but including additional complications as well.
• Object Oriented Testing Techniques takes more time as compare to testing of Procedural Program

References

[7]. Introduction, Code Coverage Analysis, Steve Cornett.
[16]. Jovanovi,ć, Irena,’Software Testing Methods and Techniques’Page No-30-41