Software Testing

Software testing is an essential part of the software development process, which is used to identify the correctness, completeness and quality of developed software. Its main objective is to detect errors in the software. Errors prevent software from producing outputs according to user requirements.

Software testing is a process, which is used to identify the correctness, completeness and quality of software. IEEE defines testing as "the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results."

Software testing is often used in association with the terms verification and validation. Verification refers to checking or testing of items, including software, for conformance and consistency with an associated specification. For verification, techniques like reviews, analysis, inspections and walkthroughs are commonly used. While validation refers to the process of checking that, the developed software is according to the requirements specified by the user. Verification and validation can be summarised as follows:

- Verification: Are we developing the software right?
- Validation: Are we developing the right software?

Objectives of Software Testing

Software testing evaluates the software by manual and automated means to ensure that it is functioning in accordance with user requirements. The main objectives of software testing are listed below:

- To remove errors, which prevent software from producing outputs according to user requirements?
- To remove errors that lead to software failure.
- To determines whether the system meets business and user needs.
- To ensure that software is developed according to user requirements.
- To improve the quality of software by removing maximum possible errors from it.

Principles of Software Testing

There are certain principles that are followed during software testing. These principles act as a standard to test software and make testing more effective and efficient. The commonly used software testing principles are listed below:

- Define the expected output: When programs are executed during testing, they may or may not produce the expected outputs due to different types of errors present in the software. To avoid this, it is necessary to define the expected output before software testing begins. Without knowledge of the expected results, testers may fail to detect an erroneous output.
- Inspect the output of each test completely: Software testing should be performed once the software is complete in order to check its performance and functionality. Also, testing should be performed to find the errors that occur in various phases of software development.
- Include test cases for invalid and unexpected conditions: Generally, the software produces correct outputs when it is tested using accurate inputs. However, if unexpected input is given to the software, it may produce erroneous outputs. Hence, test cases that detect errors even when unexpected and incorrect inputs are specified should be developed.
- Test the modified program to check its expected performance: Sometimes, when certain modifications are made in software (like adding of new functions) it is possible that software produces unexpected outputs. Hence, software should be tested to verify that it performs in an expected manner even after modifications.

Test Plan

A test plan describes how testing would be accomplished. A test plan is defined as a document that describes the objectives, scope, method, and purpose of software testing. This plan identifies test items, features to be tested, testing tasks and the persons involved in performing these tasks. It also identifies the test environment and the test design and measurement techniques that are to be used. Note that a properly defined test plan is an agreement between testers and users describing the role of testing in software.

A complete test plan helps people outside the test group to understand the 'why' and 'how' of product validation. Whereas an incomplete test plan can result in a failure to check how the software works on different hardware and operating systems or when the software is used with other software. To avoid this problem, IEEE states some components that should be covered in a test plan. These components are listed in the table below:

Component	Purpose
Responsibilities	Assigns responsibilities, keeps people on track and focused.
Assumptions	Avoids misunderstandings about schedules.
Test	Outlines the entire process and maps specific tests. The testing scope, schedule, and duration are also outlined.
Communication	Communication plan (who, what, when, how about the people) is developed.
Risk Analysis	Identifies areas that are critical for success.
Defect Reporting	Specifies how to document a defect so that it can be reproduced, fixed, and retested.
Environment	Specifies the technical environment, data, work area, and interfaces used in testing. This reduces or eliminates misunderstandings and sources of the potential delay.

Test Case Design

A test case is a document that describes an input, action, or event and its expected result, in order to determine whether the software or a part of the software is working correctly or not. IEEE defines test case as "a set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement". Generally, a test case contains particulars, such as test case identifier, test case name, its objective, test conditions/setup, input data requirements, steps, and expected results.

Incomplete and incorrect test cases lead to incorrect and erroneous test outputs. To avoid this, a test case should be developed in such a manner that it checks software with all possible inputs. This process is known as exhaustive testing and the test case, which is able to perform exhaustive testing, is known as the ideal test case.

Software Testing Strategies

Software testing strategies can be considered as various levels of testing that are performed to test the software. The first level starts with testing of individual units of software. Once the individual units are tested, they are integrated and checked for interfaces established between them. After this, the entire software is tested to ensure that the output produced is according to user requirements. There are four levels of software testing namely Unit Testing, Integration Testing, System Testing, and Acceptance Testing.

Unit Testing

Unit testing is performed to test the individual units of software. Since the software is made of a number of units/modules, detecting errors in these units is simple and consumes less time, as they are small. However, it is possible that the outputs produced by one unit become input for another unit. Hence, if incorrect output produced by one unit is provided as input to the second unit, then it also produces wrong output. If this process is not corrected, the entire software may produce unexpected outputs.

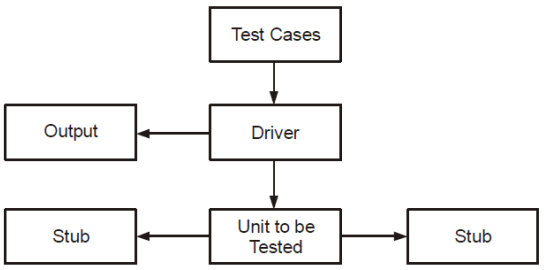

Figure 5.9 Unit Testing Environment

To avoid this, all the units in the software are tested independently using unit testing. Unit-level testing is not just performed once during the software development, rather it is repeated whenever software is modified or used in a new environment.

- Driver is a module that passes input to the unit to be tested. It accepts test case data and then passes the data to the unit being tested. After this, the driver prints the output produced.
- Stub is a module that works as a unit referenced by the unit being tested. It uses the interface of the subordinate unit, does minimum data manipulation, and returns control back to the unit being tested.

Integration Testing

Once unit testing is complete, integration testing begins. In integration testing, the units validated during unit testing are combined to form a subsystem. The purpose of integration testing is to ensure that all the modules continue to work in accordance with user/customer requirements even after integration.

The objective of integration testing is to take all the tested individual modules, integrate them, test them again, and develop the software, which is according to design specifications.

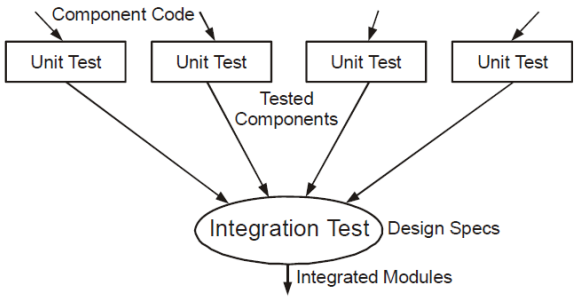

Figure 5.10 Integration of Individual Modules

The Big Bang approach and Incremental Integration approach are used to integrate modules of a program.

- In the Big Bang approach, initially, all modules are integrated and then the entire program is tested. However, when the entire program is tested, it is possible that a set of errors is detected. It is difficult to correct these errors since it is difficult to isolate the exact cause of the errors when program is very large. In addition, when one set of errors is corrected, new sets of errors arise and this process continues indefinitely.
- The Incremental Integration approach tests program in small increments. It is easier to detect errors in this approach because only a small segment of software code is tested at a given instance of time. Moreover, interfaces can be tested completely if this approach is used. Various kinds of approaches are used for performing incremental integration testing:
  - Top-down integration testing
  - Bottom-up integration testing
  - Regression testing
  - Smoke testing

System Testing

Software is integrated with other elements, such as hardware, people, and database to form a computer-based system. This system is then checked for errors using system testing. IEEE defines system testing as "testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements".

System testing compares the system with the non-functional system requirements, such as security, speed, accuracy, and reliability. The emphasis is on validating and verifying the functional design specifications and examining how modules work together. This testing also evaluates external interfaces to other applications and utilities or the operating environment.

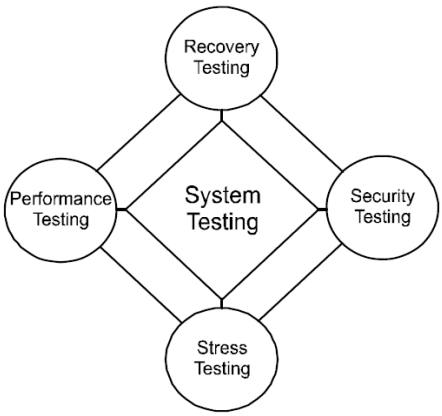

Figure 5.16 Types of System Testing

Validation/Acceptance Testing

Validation testing is performed to determine whether the software meets all the functional, behavioural and performance requirements or not. During validation testing, software is tested and evaluated by a group of users at either the developer's site or user's site. This enables the users to test the software themselves and analyse whether the software is meeting their requirements or not.

IEEE defines acceptance testing as "formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorised entity to determine whether or not to accept the system".

- Alpha Testing: Alpha testing is conducted by the users at the developer's site. In other words, this testing assesses the performance of the software in the environment in which it is developed. On completion of alpha testing, users report the errors to software developers so that they can correct them. Note that alpha testing is often employed as a form of internal acceptance testing. The advantages of alpha testing are listed below:
  - Identifies all the errors present in the software.
  - Checks whether all the functions mentioned in the requirements are implemented properly in software or not.
- Beta Testing: Beta testing assesses the performance of software at user's site. This testing is 'live' testing and is conducted in an environment, which is not controlled by the developer. That is, this testing is performed without any interference from the developer. Beta testing is performed to know whether the developed software satisfies the user requirements and fits within the business processes or not. The advantages of beta testing are listed below:
  - Evaluates the entire documentation of software. For example, it examines the detailed description of the software code, which forms a part of documentation of software.
  - Checks whether the software is operating successfully in user environment or not.

Testing Techniques

Once the software is developed, it should be tested in a proper manner before the system is delivered to the user. For this, two techniques that provide systematic guidance for designing tests are used. These techniques are listed below:

- Once the internal working of software is known, tests are performed to ensure that all internal operations of software are performed according to specifications. This is referred to as white box testing.
- Once the specified function for which software has been designed is known, tests are performed to ensure that each function is working properly. This is referred to as black-box testing.

White Box Testing

White box testing, also known as structural testing is performed to check the internal structure of a program. To perform white box testing, the tester should have a thorough knowledge of the program code and the purpose for which it is developed. The basic strength of this testing is that the entire software implementation is included while testing is performed. This facilitates error detection even when the software specification is vague or incomplete.

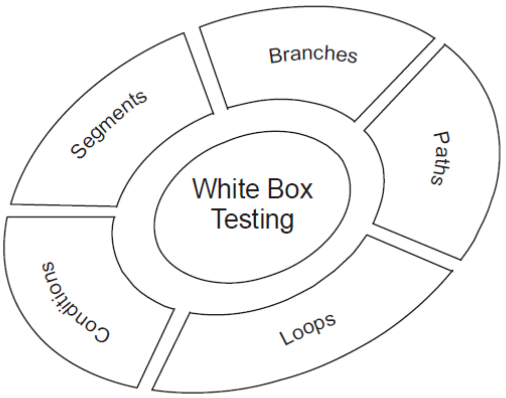

Figure 5.20 White Box Testing

The objective of white-box testing is to ensure that the test cases (developed by software testers by using white box testing) exercise each path through a program. That is, test cases ensure that all internal structures in the program are developed according to design specifications.

The effectiveness of white-box testing is commonly expressed in terms of test or code coverage metrics, which measure the fraction of code exercised by test cases. The various types of testing which occur as a part of white-box testing are Basis Path Testing, Control Structure Testing and Mutation Testing.

Advantages

- Covers the larger part of the program code while testing.
- Uncovers typographical errors.
- Detects design errors that occur when incorrect assumptions are made about execution paths.

Disadvantages

- Tests that cover most of the program code may not be good for assessing the functionality of surprise (unexpected) behaviours and other testing goals.
- Tests based on design may miss other system problems.
- Tests cases need to be changed if implementation changes.

Black Box Testing

Black box testing, also known as functional testing, checks the functional requirements and examines the input and output data of these requirements. The functionality is determined by observing the outputs to corresponding inputs. For example, when black-box testing is used, the tester should only know the 'legal' inputs and what the expected outputs should be, but not how the program actually arrives at those outputs.

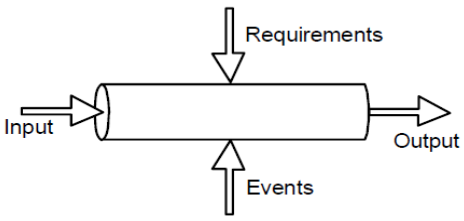

Figure 5.28 Black Box Testing

In this testing, various inputs are exercised and the outputs are compared against specification to validate the correctness. Note that test cases are derived from these specifications without considering implementation details of the code. The outputs are compared with user requirements and if they are as specified by the user, then the software is considered to be correct, else the software is tested for the presence of errors in it. The various methods used in black-box testing are Equivalence Class Partitioning, Boundary Value Analysis, Orthogonal Array Testing, and Cause-Effect Graphing.

Advantages

- Tester requires no knowledge of implementation and programming language used.
- Reveals any ambiguities and inconsistencies in the functional specifications.
- Efficient when used on larger systems.
- A non-technical person can also perform black-box testing.

Disadvantages

- Only a small number of possible inputs can be tested as testing every possible input consumes a lot of time.
- There can be an unnecessary repetition of test inputs if the tester is not informed about the test cases that software developer has already tried.
- Leaves many program paths untested.
- Cannot be directed towards specific segments of code, hence is more error-prone.

Difference b/w White Box Testing & Black Box Testing

Basis	White Box Testing	Black Box Testing
Purpose	It is used to test the internal structure of the software.	It is used to test the functionality of the software.
	It is concerned only with testing software and does not guarantee the complete implementation of all the specifications mentioned in user requirements.	It is concerned only with testing specifications and does not guarantee that all the components of the software that are implemented are tested.
	It addresses the flow and control structure of a program.	It addresses validity, behaviour and performance of the software.
Stage	It is performed in the early stages of testing.	It is performed in the later stages of testing.
Requirement	Knowledge of the internal structure of a program is required for generating a test case.	No knowledge of the internal structure of a program is required to generate a test case.
Test Cases	Here test cases are generated based on the actual code of the module to be tested.	Here the internal structure of modules or programs is not considered for selecting test cases.
Example	The inner software present inside the calculator (which is known by the developer only) is checked by giving inputs to the code.	In this testing, it is checked whether the calculator is working properly or not by giving inputs by pressing the buttons in the calculator.

Gray Box Testing

Gray box testing does not require full knowledge of the internals of the software that is to be tested instead it is a test strategy, which is based partly on the internals. This testing technique is often defined as a mixture of black-box testing and white box testing techniques. Grey box testing is especially used in web applications as these applications are built around loosely integrated components that connect through relatively well-defined interfaces. Testing in this methodology is done from the outside of the software similar to black-box testing. However, testing choices are developed through the knowledge of how the underlying components operate and interact.

- Gray box testing is platform and language independent.
- The current implementation of grey box testing is heavily dependent on the use of a host platform debugger(s) to execute and validate the software under test.
- Gray box testing can be applied in real-time systems.
- Gray box testing utilises automated software-testing tools to facilitate the generation of test cases.
- Module drivers and stubs are created by automation means thus, saving the time of testers.