**\* Static and Dynamic Testing**

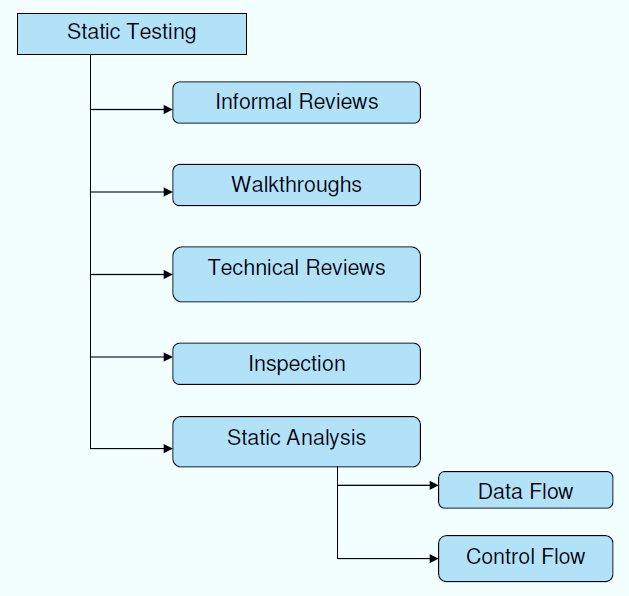
**Static Testing Technique**

**Static Testing** is a software testing technique which is used to check defects in software application without executing the code. Static testing is done to avoid errors at an early stage of development as it is easier to identify the errors and solve the errors. It also helps finding errors that may not be found by Dynamic Testing.

The two main types of static testing techniques are

**Manual examinations**: Manual examinations include analysis of code done manually, also known as REVIEWS.

**Automated analysis using tools**: Automated analysis are basically static analysis which is done using tools.

****

Tools used for static testing are Checkstyle, Soot, Sourcemeter

A **review** in Static Testing is a process or meeting conducted to find the potential defects in the design of any program. Another significance of review is that all the team members get to know about the progress of the project and sometimes the diversity of thoughts may result in excellent suggestions. Documents are directly examined by people and discrepancies are sorted out.

Reviews can further be classified into four parts:

* Informal reviews
* Walkthroughs
* Technical review
* Inspections

During the Review process four types of participants that take part in testing are: Moderator, Author, Manager, Scribe, Reviewer

Types of defects which can be easier to find during static testing are:

* Deviations from standards
* Non-maintainable code
* Design defects
* Missing requirements
* Inconsistent interface specifications

Usually, the defect discovered during static testing are due to security vulnerabilities, undeclared variables, boundary violations, syntax violations, inconsistent interface, etc.

**Data Flow Analysis:**

**Description:** Data Flow Analysis is a static code analysis technique used to analyze how data moves through a program. It tracks the flow of data variables and identifies potential issues like uninitialized variables, data dependencies, and security vulnerabilities.

**Purpose: To** detect data-related errors and security vulnerabilities in code.

**Methods:** Abstract interpretation, program slicing, and data flow analysis algorithms.

**Benefits:** Identifies data-related defects, enhances code understanding, and helps ensure data integrity and security.

**Tool**: ESLint

**Control Flow Analysis:**

**Description:** Control Flow Analysis is another static analysis technique that examines the flow of control within a program. It analyzes the sequence of statements and identifies potential issues like dead code, infinite loops, and unreachable code paths.

**Purpose**: To ensure that the program's logic and control flow are correct and that no unexpected behaviors occur.

**Methods:** Control flow graphs, path analysis, and static analysis tools.

**Benefits:** Helps find control-related defects, improves code reliability, and enhances code maintainability. Tool: PMD

**Cyclomatic Complexity** in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs or with respect to functions, modules, methods or classes within a software program.

Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths.

**How to calculate:**

V(G) = E - N+2

Where, E = Number of Edges and N = Number of Nodes

V(G) = P+1

Where P = Number of predicate nodes (node that contains condition)

Example:

i = 0;

n=4; //N-Number of nodes present in the graph

while (i<n-1) do

j = i + 1;

while (j<n) do

if A[i]<A[j] then

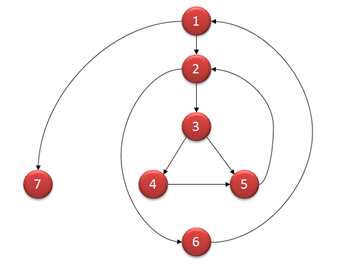
swap(A[i], A[j]);

end do;

j=j+1;

end do;

Flow graph of this program will be:



Computing mathematically,

* V(G) = 9 – 7 + 2 = 4
* V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
* Basis Set – A set of possible execution path of a program
* 1, 7
* 1, 2, 6, 1, 7
* 1, 2, 3, 4, 5, 2, 6, 1, 7
* 1, 2, 3, 5, 2, 6, 1, 7

**Properties of Cyclomatic complexity:**

Following are the properties of Cyclomatic complexity:

* V (G) is the maximum number of independent paths in the graph
* V (G) >=1
* G will have one path if V (G) = 1
* Minimize complexity to 10

**Dynamic Testing Technique**

**Dynamic Testing** is a software testing method used to test the dynamic behaviour of software code. The main purpose of dynamic testing is to test software behaviour with dynamic variables or variables which are not constant and finding weak areas in software runtime environment. The code must be executed in order to test the dynamic behavior.

We all know that Testing is verification and validation, and it takes 2 Vs to make testing complete. Out of the 2 Vs, Verification is called a Static testing and the other “V”, Validation is known as Dynamic testing.

Suppose we are testing a Login Page where we have two fields say “Username” and “Password” and the Username is restricted to Alphanumeric.

When the user enters Username as “Guru99”, the system accepts the same. Where as when the user enters as Guru99@123 then the application throws an error message. This result shows that the code is acting dynamically **based on the user input.**

Dynamic testing is when you are working with the actual system by providing an input and comparing the actual behavior of the application to the expected behavior. In other words, working with the system with the intent of finding errors.

Dynamic Testing is classified into two categories

* White Box Testing
* Black Box Testing

A diagram of a software testing process

Description automatically generated

**White Box Testing** is a software testing method in which the internal structure/ design is known to the tester. The main aim of White Box testing is to check on how System is performing based on the code. It is mainly performed by the Developers or White Box Testers who has knowledge on the programming. **White Box Testing** is a testing technique in which software’s internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.

Following are important WhiteBox Testing Techniques:

* Statement Coverage
* Decision Coverage
* Branch Coverage
* Condition Coverage
* Multiple Condition Coverage
* Finite State Machine Coverage
* Path Coverage
* Control flow testing
* Data flow testing

White box testing tools:

* [EclEmma](https://www.eclemma.org/download.html)
* [NUnit](http://nunit.org/)
* [PyUnit](https://www.guru99.com/python-unit-testing-guide.html)
* [HTMLUnit](http://htmlunit.sourceforge.net/)
* [CppUnit](https://sourceforge.net/projects/cppunit/)

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a [Test Case](https://www.guru99.com/test-case.html) suite.

Following are major code coverage methods

* Statement Coverage
* Decision Coverage
* Branch Coverage
* Toggle Coverage
* FSM Coverage

**Statement Coverage** is a white box testing technique in which all the executable statements in the source code are executed at least once. It is used for calculation of the number of statements in source code which have been executed. The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code.

Statement coverage is used to derive scenario based upon the structure of the code under test.

A red line between green text

Description automatically generated

**What is covered by Statement Coverage?**

* Unused Statements
* Dead Code
* Unused Branches
* Missing Statements

**Decision Coverage** is a white box testing technique which reports the true or false outcomes of each boolean expression of the source code. The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage.



**Branch Coverage** is a white box testing method in which every outcome from a code module(statement or loop) is tested. The purpose of branch coverage is to ensure that each decision condition from every branch is executed at least once. It helps to measure fractions of independent code segments and to find out sections having no branches.

For example, if the outcomes are binary, you need to test both True and False outcomes.

A black text on a white background

Description automatically generated

**Condition Coverage** or expression coverage is a testing method used to test and evaluate the variables or sub-expressions in the conditional statement. The goal of condition coverage is to check individual outcomes for each logical condition. Condition coverage offers better sensitivity to the control flow than decision coverage. In this coverage, expressions with logical operands are only considered.

For example, if an expression has Boolean operations like AND, OR, XOR, which indicates total possibilities. Condition coverage does not give a guarantee about full decision coverage.

A black text on a white background

Description automatically generated

Code coverage tools:

* Cobertura
* Clover
* DevPartner
* Emma
* KaliStick
* Coview and Coant
* Bullseye for c++
* Sonar

**Path testing** is a structural testing method that involves using the source code of a program in order to find every possible executable path. It helps to determine all faults lying within a piece of code. This method is designed to execute all or selected path through a computer program.

Any software program includes, multiple entry and exit points. Testing each of these points is a challenging as well as time-consuming. In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

**Basis Path Testing** in software engineering is a [White Box Testing](https://www.guru99.com/white-box-testing.html) method in which test cases are defined based on flows or logical paths that can be taken through the program. The objective of basis path testing is to define the number of independent paths, so the number of test cases needed can be defined explicitly to maximize test coverage.

In [software engineering](https://www.guru99.com/what-is-software-engineering.html), Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases. It is a hybrid method of branch testing and path testing methods.

The basic steps involved in basis path testing include

* Draw a control graph (to determine different program paths)
* Calculate [Cyclomatic complexity](https://www.guru99.com/cyclomatic-complexity.html) (metrics to determine the number of independent paths)
* Find a basis set of paths
* Generate test cases to exercise each path

**Black Box Testing**

**Black Box Testing** is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Following are the prominent [Test Strategy](https://www.guru99.com/how-to-create-test-strategy-document.html) amongst the many used in Black box Testing:

**Equivalence Class Testing**: It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.

**Boundary Value Testing**: Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.

**Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.

There are many types of Black Box Testing but the following are the prominent ones –

**Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers.

**Non-functional testing** – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

**Regression testing** – [Regression Testing](https://www.guru99.com/regression-testing.html) is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black box testing largely depends on the type of black box testing you are doing.

For Functional/ Regression Tests you can use – [**QTP**](https://www.guru99.com/quick-test-professional-qtp-tutorial.html)**,** [**Selenium**](https://www.guru99.com/selenium-tutorial.html)

For Non-Functional Tests, you can use – [**LoadRunner**](https://www.guru99.com/loadrunner-v12-tutorials.html)**,** [**Jmeter**](https://www.guru99.com/jmeter-tutorials.html)

**Functional Testing** is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application Under Test. The testing can be done either manually or using automation.

Following is a step by step process on How to do Functional Testing :

* Understand the Functional Requirements
* Identify test input or test data based on requirements
* Compute the expected outcomes with selected test input values
* Execute test cases
* Compare actual and computed expected results

Here are Examples of Functional Testing Types

* Unit testing
* Smoke testing
* User Acceptance
* Integration Testing
* Regression testing
* Localization
* Globalization
* Interoperability

**Non-Functional Testing** is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

An excellent example of non-functional test would be to check how many people can simultaneously login into a software.

A group of colorful squares with white text

Description automatically generated

Following are the most common Types of Non Functional Testing :

* Performance Testing
* Load Testing
* Failover Testing
* Compatibility Testing
* Usability Testing
* Stress Testing
* Maintainability Testing
* Scalability Testing
* Volume Testing
* Security Testing
* Disaster Recovery Testing
* Compliance Testing
* Portability Testing
* Efficiency Testing
* Reliability Testing
* Baseline Testing
* Endurance Testing
* Documentation Testing
* Recovery Testing
* Internationalization Testing
* Localization Testing

|  |  |
| --- | --- |
| Static Testing | Dynamic Testing |
| Testing is done without executing the program | Testing is done by executing the program |
| This testing does the verification process | Dynamic testing does the validation process |
| Static testing is about prevention of defects | Dynamic testing is about finding and fixing the defects |
| Static testing gives an assessment of code and documentation | Dynamic testing gives bugs/bottlenecks in the software system. |
| Static testing involves a checklist and process to be followed | Dynamic testing involves test cases for execution |
| This testing can be performed before compilation | Dynamic testing is performed after compilation |
| Static testing covers the structural and statement coverage testing | Dynamic testing techniques are Boundary Value Analysis & Equivalence Partitioning. |
| Cost of finding defects and fixing is less | Cost of finding and fixing defects is high |
| Return on investment will be high as this process involved at an early stage | Return on investment will be low as this process involves after the development phase |
| More reviews comments are highly recommended for good quality | More defects are highly recommended for good quality. |
| Requires loads of meetings | Comparatively requires lesser meetings |