ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Khoa cntt – Bộ môn kỹ thuật phần mềm

CHƯƠNG 4

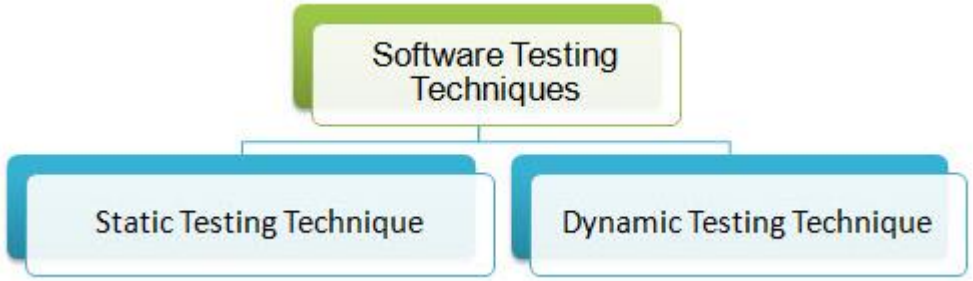
STATIC TESTING VS DYNAMIC TESTING

OBJECTIVES

STUDENTS CAN:

- Understanding and distinguishing static and dynamic testing
- Understanding and approaching the technical requirements of each of them
- Processes of them
- Frame and tools supporting

CONTENT


Software Testing Techniques
Static Testing Technique
Dynamic Testing Technique


Static Testing And Dynamic Testing
Static testing
Inspection
Walkthrough
Technical Reviews
Informal Reviews
Dynamic testing
Unit Testing
Integration
System
Acceptance
www.techbeamers.com

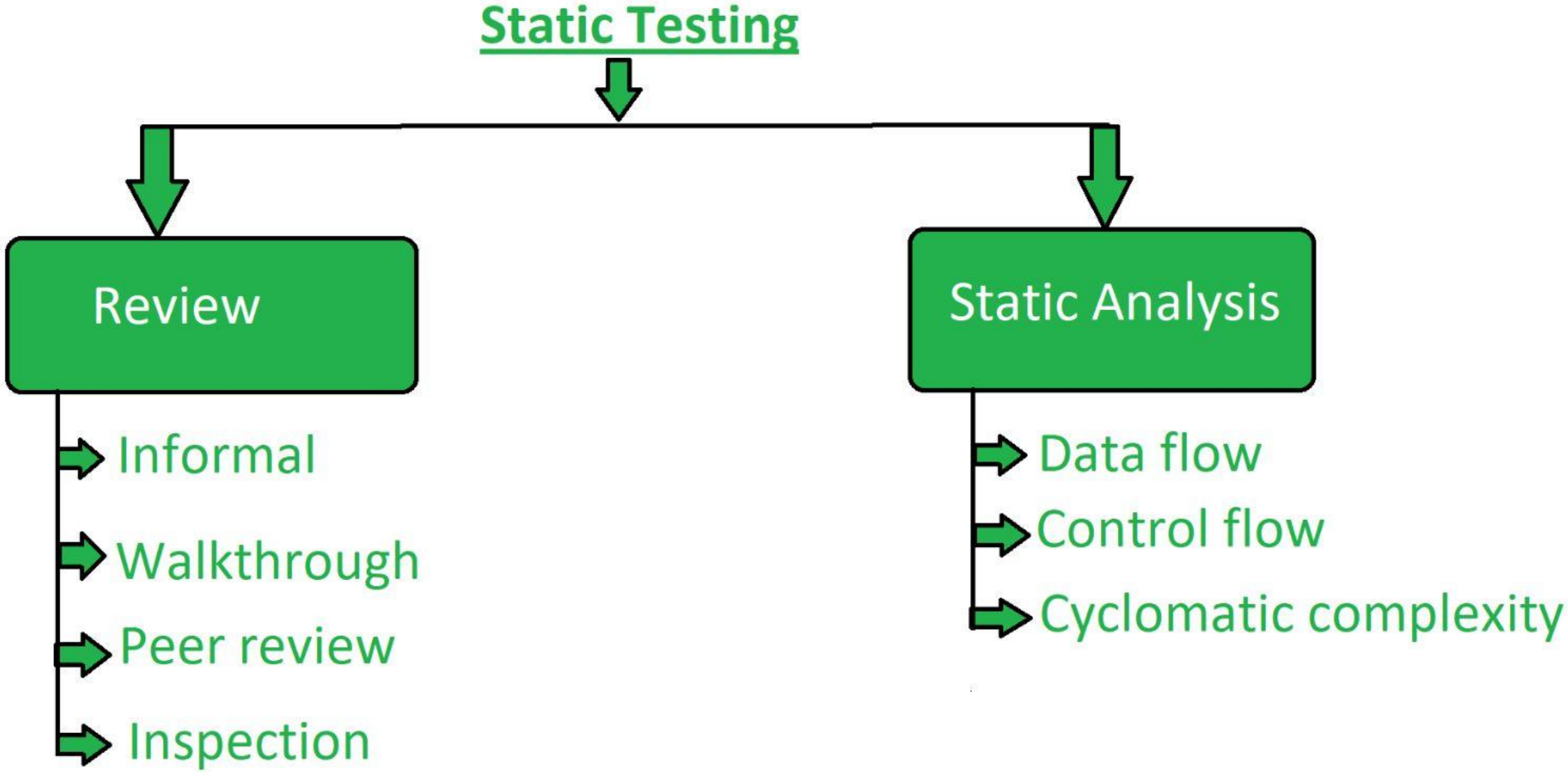
STATIC TESTING

- **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.

FORMS OF STATIC TESTING

- **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.
-

Static Testing Techniques

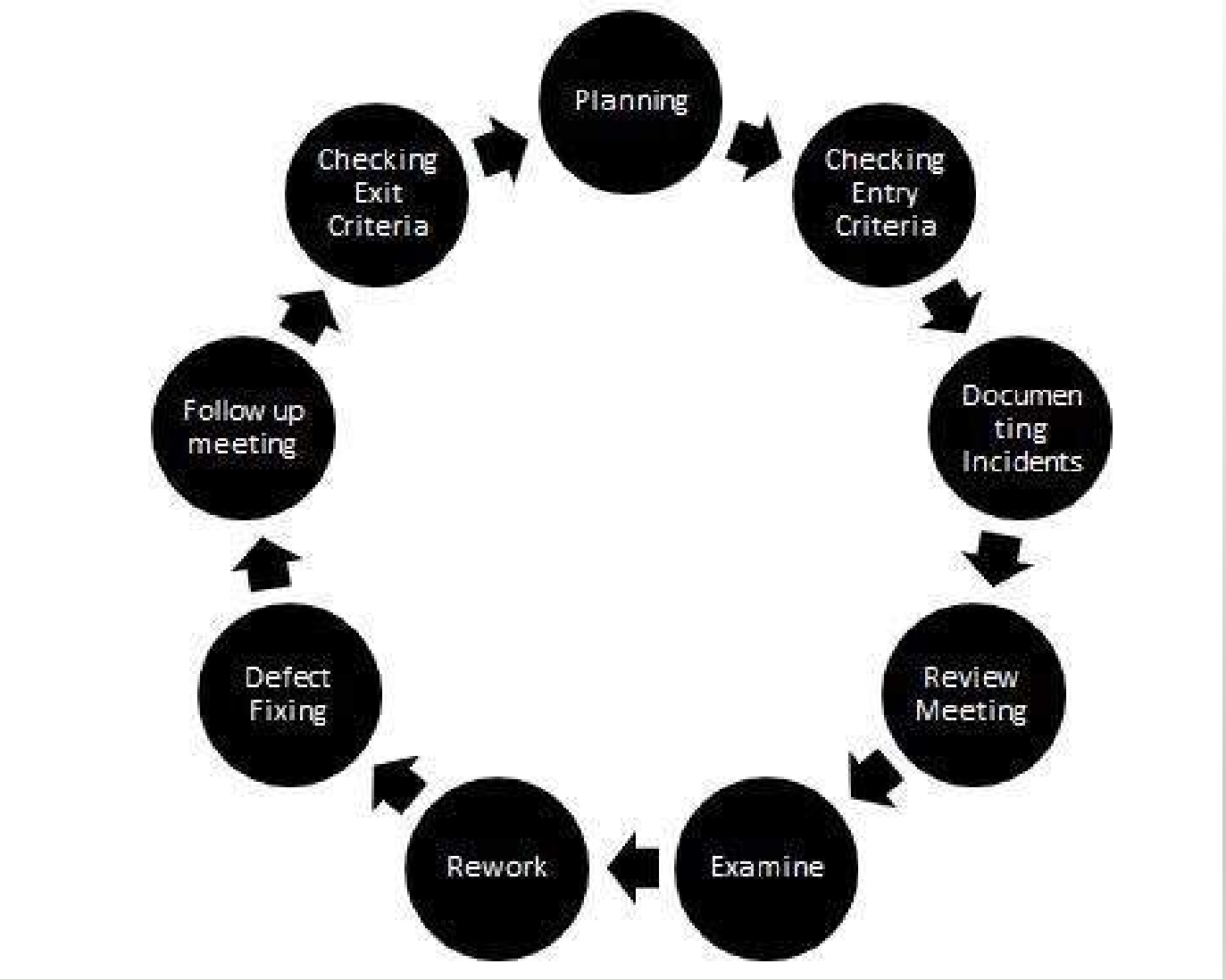


TYPES OF STATIC TESTING=> REVIEW

REVIEW

In static testing review is a process or technique that is performed to find the potential defects in the design of the software. It is process to detect and remove errors and defects in the different supporting documents like software requirements specifications. People examine the documents and sorted out errors, redundancies and ambiguities.

Review stages - workflow:



GOAL of Review

- Productivity of Dev team is improved and timescales reduced because the correction of defects in early stages and work-products will help to ensure that those work-products are clear and unambiguous.
- Testing costs and time is reduced as there is enough time spent during the initial phase.
- Reduction in costs because fewer defects in the final software.
  - Deviations from standards either internally defined or defined by regulatory or a trade organisation.
  - Requirements defects/ Missing req
  - Design defects.
  - Incorrect/Inconsistent interface specifications.

Participants

- **Moderator**: Performs entry check, follow up on rework, coaching team member, schedule the meeting.
- **Author**: Takes responsibility for fixing the defect found and improves the quality of the document
- **Scribe**: It does the logging of the defect during a review and attends the review meeting
- **Reviewer**: Check material for defects and inspects
- **Manager**: Decide on the execution of reviews and ensures the review process objectives are met.

4 types

- **Informal:**
  In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.
- **Walkthrough:**
  It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.
- **Peer review:**
  Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.
- **Inspection:**
  Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

Informal rereview

Informal reviews take place between two or three people. The review conference is scheduled at their convenience.

- This meeting is generally scheduled during the free time of the team members.
- There is no planning for the meeting.
- If any errors occur, they are not corrected in the informal reviews.
- There is no guidance from the team.
- This review is less effective compared to the formal review

Walthrough

- It is not a formal process/review
- It is led by the authors
- Author guide the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
- Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
- Is especially useful for higher level documents like requirement specification, etc.

The goals of a walkthrough:

- To present the documents both within and outside the software discipline in order to gather the information regarding the topic under documentation.
- To explain or do the knowledge transfer and evaluate the contents of the document
- To achieve a common understanding and to gather feedback.
- To examine and discuss the validity of the proposed solutions
-

Peer Review

Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues.

A peer review, a review technique, which is a static white-box testing which are conducted to spot the defects early in the life cycle that cannot be detected by black box testing techniques.

- Peer Review Characteristics:

- Peer Reviews are documented and uses a defect detection process that has peers and technical specialist as part of the review process.
- The Review process doesn't involve management participation.
- It is usually led by trained moderator who is NOT the author.
- The report is prepared with the list of issues that needs to be addressed.

Inspections

- Inspection is the most formal form of reviews, a strategy adopted during static testing phase.
- Characteristics of Inspection :

- Inspection is usually led by a trained moderator, who is not the author. Moderator's role is to do a peer examination of a document
- Inspection is most formal and driven by checklists and rules.
- This review process makes use of entry and exit criteria.
- It is essential to have a pre-meeting preparation.
- Inspection report is prepared and shared with the author for appropriate actions.
- Post Inspection, a formal follow-up process is used to ensure a timely and a prompt corrective action.
- Aim of Inspection is NOT only to identify defects but also to bring in for process improvement.

How Static Testing is Performed

- **Carry out the inspection process to completely inspect the design of the application**
- **Use a checklist for each document under review to ensure all reviews are covered completely**
- The various activities for performing Static Testing are:

1. Unit Test Cases
2. Business Requirements Document (BRD)
3. Use Cases
4. System/Functional Requirements
5. Prototype
6. Prototype Specification Document
6. DB Fields Dictionary Spreadsheet
7. Test Data
8. Traceability Matrix Document
9. User Manual/Training Guides/Documentation
10. Test Plan Strategy Document/Test Cases
11. Automation/Performance Test Scripts

Static Analysis

- Static Analysis includes the evaluation of the code quality that is written by developers. Different tools are used to do the analysis of the code and comparison of the same with the standard.
- It also helps in following identification of following defects:

(a)Unused variables
(b)Dead code
(c)Infinite loops
(d)Variable with undefined value
(e)Wrong syntax

STATIC ANALISYS TYPES

- **Data Flow:**
  Data flow is related to the stream processing.
- **Control Flow:**
  Control flow is basically how the statements or instructions are executed.
- **Cyclomatic Complexity:**
  Cyclomatic complexity is the measurement of the complexity of the program that is basically related to the number of independent paths in the control flow graph of the program.

Tools used for Static Testing

1. Checkstyle (https://checkstyle.sourceforge.io/)
2. Soot (https://github.com/soot-oss/soot)
3. SourceMeter (https://www.sourcemeter.com/)

Requirements review

- **SRS document format**
- .1. Introduction
- **(i)** Purpose of this document
- **(ii)** Scope of this document
- **(iii)** Overview
- 2. General description
  3. Functional Requirements
  4. Interface Requirements
  5. Performance Requirements
  6. Design Constraints
  7. Non-Functional Attributes
  8. Preliminary Schedule and Budget
  9. Appendices
- (https://www.geeksforgeeks.org/software-requirement-specification-srs-format/?ref=lbp)

Quality Characteristics of a good SRS

1. Correctness:
2. Completeness:
3. Consistency:
4. Unambiguousness:
5. Ranking for importance and stability:
6. Modifiability:
7. Verifiability:
8. Traceability:
9. Design Independence:
10. Testability:
11. Understandable by the customer:
12. Right level
13. (https://www.geeksforgeeks.org/software-engineering-quality-characteristics-of-a-good-srs/?ref=rp of abstraction:)

Design review

- Check the checklist
- https://www.smartsheet.com/sites/default/files/2020-06/IC-Software-Design-Review-Checklist-10816_PDF.pdf
- https://evelyne24.github.io/system-design-checklist/

Code reviews

- Verify feature requirements
- Code readability
- Coding Style/coding standards
- Clear naming
- Code duplication
- Tests/Unit test
- Documentation
- https://blog.haposoft.com/code-review-checklist-tap-trung-vao-van-de-quan-trong/

Coding standards

Coding standards are a set of guidelines, best practices, programming styles and conventions that developers adhere to when writing source code for a project. All big software companies have them.

- **Purpose of Having Coding Standards:**

- A coding standard gives a uniform appearance to the codes written by different engineers.
- It improves readability, and maintainability of the code and it reduces complexity also.
- It helps in code reuse and helps to detect error easily.
- It promotes sound programming practices and increases efficiency of the programmers.

There are somes:

Coding Standards and Guidelines – GeeksforGeeks

C# Coding Standards Best Practices – Dofactory

C# Coding Standards – GeeksforGeeks

(….)

Coding conventions

Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices, and methods for each aspect of a program written in that language. These conventions usually cover file organization, indentation, comments, declarations, statements, white space, naming conventions, programming practices, programming principles, programming rules of thumb, architectural best practices, etc. These are guidelines for software structural quality.

➔ The differences of coding standards and coding convention?

Code coverage

- **Code coverage is a measure which describes the degree of which the source code of the program has been tested. It is one form of white box testing which finds the areas of the program not exercised by a set of test cases. It also creates some test cases to increase coverage and determining a quantitative measure of code coverage.**
- **In most cases, code coverage system gathers information about the running program. It also combines that with source code information to generate a report about the test suite's code coverage.**

Why?

- It helps you to measure the efficiency of test implementation
- It offers a quantitative measurement.
- It defines the degree to which the source code has been tested.

Methods?

1. Statement Coverage
2. Decision Coverage
3. Branch Coverage
4. Condition Coverage

Statement Coverage

Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once. It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.

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

$$\text{Statement Coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

Statement Coverage

Ex

Scenerio 1:

If a= 3, b = 9

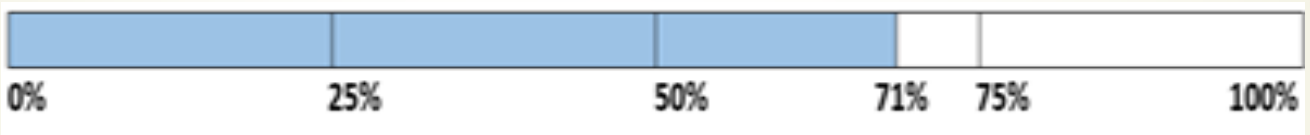
```
Prints (int a, int b) {                          ------------  Printsum is a functi
on
    int result = a+ b;
    If (result> 0)
        Print ("Positive", result)
    Else
        Print ("Negative", result)
    }                                           -----------   End of the source cod
e
```

```
1  Prints (int a, int b) {
2  int result = a+ b;
3  If (result> 0)
4      Print ("Positive", result)
5  Else
6      Print ("Negative", result)
7  }
```

Statement Coverage (cnt)

1. Statement Coverage

- The statements marked in yellow color are those which are executed as per the scenario
- Number of executed statements = 5, Total number of statements = 7
- Statement Coverage: 5/7 = 71%



Scenario 2:

- If a = -3, b = -9

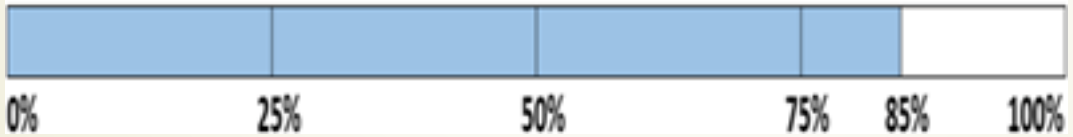
```
1  Prints (int a, int b) {
2  int result = a+ b;
3  If (result> 0)
4      Print ("Positive", result)
5  Else
6      Print ("Negative", result)
7  }
```

Statement Coverage(cnt)

- The statements marked in yellow color are those which are executed as per the scenario.
- Number of executed statements = 6
- Total number of statements = 7

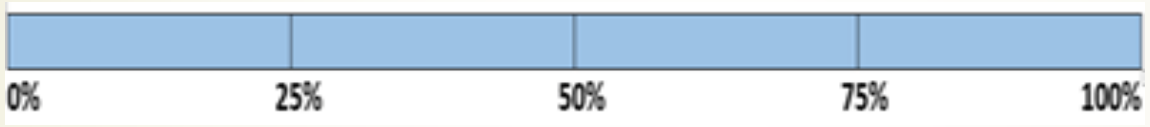
Statement Coverage: 6/7 = 85%

$$\text{Statement Coverage} = \frac{Number\ of\ executed\ statments}{Total\ number\ of\ statments}$$



Statement Coverage (cnt)

But overall if you see, all the statements are being covered by 2nd scenario's considered. So we can conclude that overall statement coverage is 100%.



- **What is covered by Statement Coverage?**
  - Unused Statements
  - Dead Code
  - Unused Branches
  - Missing Statements

Decision Coverage

Decision coverage reports the true or false outcomes of each Boolean expression. In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage.

That's why there are many different methods of reporting this metric. All these methods focus on covering the most important combinations. It is very much similar to decision coverage, but it offers better sensitivity to control flow.

$$Decision\ Coverage = \frac{Number\ of\ Decision\ Outcomes\ Excercised}{Total\ Number\ of\ Decision\ Outcomes}$$

Decision Coverage (cnt)

Scenario 1:

- Value of a is 2
- The code highlighted in yellow will be executed. Here the "No" outcome of the decision If (a>5) is checked.
- Decision Coverage = 50%

```
Demo(int a) {
      If (a> 5)
         a=a*3
      Print (a)
     }
```

```
1  Demo(int a) {
2        If (a> 5)
3           a=a*3
4        Print (a)
5  }
```

Decision Coverage (cnt)

Scenario 2:

- Value of a is 6
- The code highlighted in yellow will be executed. Here the "Yes" outcome of the decision If (a>5) is checked.
- Decision Coverage = 50%

```
1  Demo(int a) {
2      If (a> 5)
3          a=a*3
4      Print (a)
5  }
```

Test Case	Value of A	Output	Decision Coverage
1	2	2	50%
2	6	18	50%

Branch Coverage

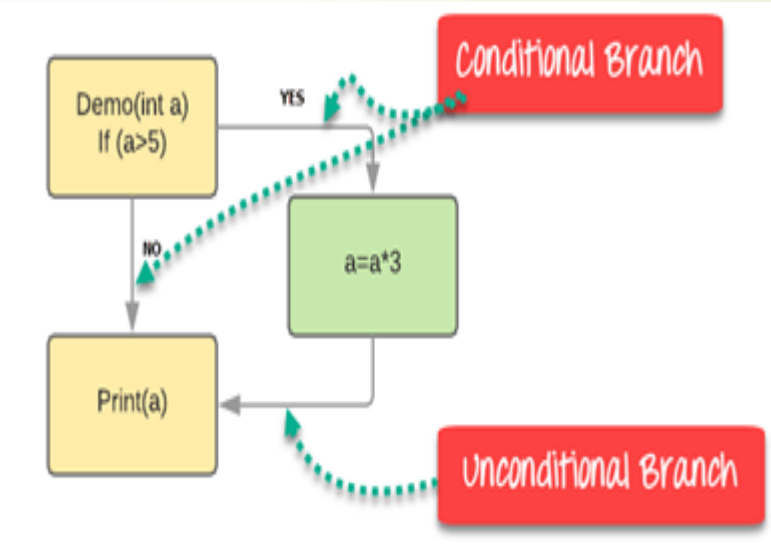
In the branch coverage, every outcome from a code module is tested. For example, if the outcomes are binary, you need to test both True and False outcomes.

- It helps you to ensure that every possible branch from each decision condition is executed at least a single time.
- By using Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which is sections of code don't have any branches.
- The formula to calculate Branch Coverage:

$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

Branch Coverage (cnt)

```
Demo(int a) {
     If (a> 5)
        a=a*3
     Print (a)
    }
```



Branch Coverage will consider unconditional branch as well

Test Case	Value of A	Output	Decision Coverage	Branch Coverage
1	2	2	50%	**33%**
2	6	18	50%	**67%**

Branch Coverage (cnt)

Advantages of Branch coverage

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation
- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

Condition coverage

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.

$$Condition\ Coverage = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands}$$

Condition coverage

Ex:

```
1   IF (x < y) AND (a>b) THEN
```

- For the above expression, we have 4 possible combinations
  - TT
  - FF
  - TF
  - FT

Consider the following input

X=3 Y=4	(x<y)	TRUE	Condition Coverage is ¼ = 25%
A=3 B=4	(a>b)	FALSE	

Finite State Machine Coverage

Finite state machine coverage is certainly the most complex type of code coverage method. This is because it works on the behavior of the design. In this coverage method, you need to look for how many time-specific states are visited, transited. It also checks how many sequences are included in a finite state machine.

Which Type of Code Coverage to Choose

- **This is certainly the most difficult answer to give. In order to select a coverage method, the tester needs to check that the**

  - code under test has single or multiple undiscovered defects
  - cost of the potential penalty
  - cost of lost reputation
  - cost of lost sale, etc.

  ==> The higher the probability that defects will cause costly production failures, the more severe the level of coverage you need to choose.

Code Coverage vs. Functional Coverage

Code Coverage	Functional Coverage
Code coverage tells you how well the source code has been exercised by your test bench.	Functional coverage measures how well the functionality of the design has been covered by your test bench.
Never use a design specification	Use design specification
Done by developers	Done by Testers

Code Coverage Tools

Cobertura	It is an open source code coverage tool. It measures test coverage by instrumenting a code base and analyze which lines of code are executing and which are not executed when the test suite runs.
Clover	Clover also reduces testng time by only running the tests which cover the application code which was modified since the previous build.
DevPartner	DevPartner enables developers to analyze Java code for Code Quality and Complexity.
Emma	EMMA supports class, method, line, and base block coverage, aggregated source file, class, and method levels.
Kalistick	Kalistick is a third party application which analyzes the codes with different perspectives.
CoView and CoAnt	Coding Software is a code coverage tool for metrics, mock object creation, code testability, path & branch coverage, etc.
Bullseye for C++	BulseyeCoverage is a code coverage tool for C++ and C.
Sonar	Sonar is an open code coverage tool which helps you to manage code quality.

Advantages and Disadvantages of Using Code Coverage

Code Coverage Advantages	Code Coverage Disadvantages
Helpful to evaluate a quantitative measure of code coverage	Even when any specific feature is not implemented in design, code coverage still report 100% coverage.
It allows you to create extra test cases to increase coverage	It is not possible to determine whether we tested all possible values of a feature with the help of code coverage
It allows you to find the areas of a program which is not exercised by a set of test cases	Code coverage is also not telling how much and how well you have covered your logic
	In the case when the specified function hasn't implemented, or a not included from the specification, then structure-based techniques cannot find that issue.

DYNAMIC TESTING

- **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.

In V model → Static testing is verification, dynamic testing is validation!!

DYNAMIC TESTING METHODS

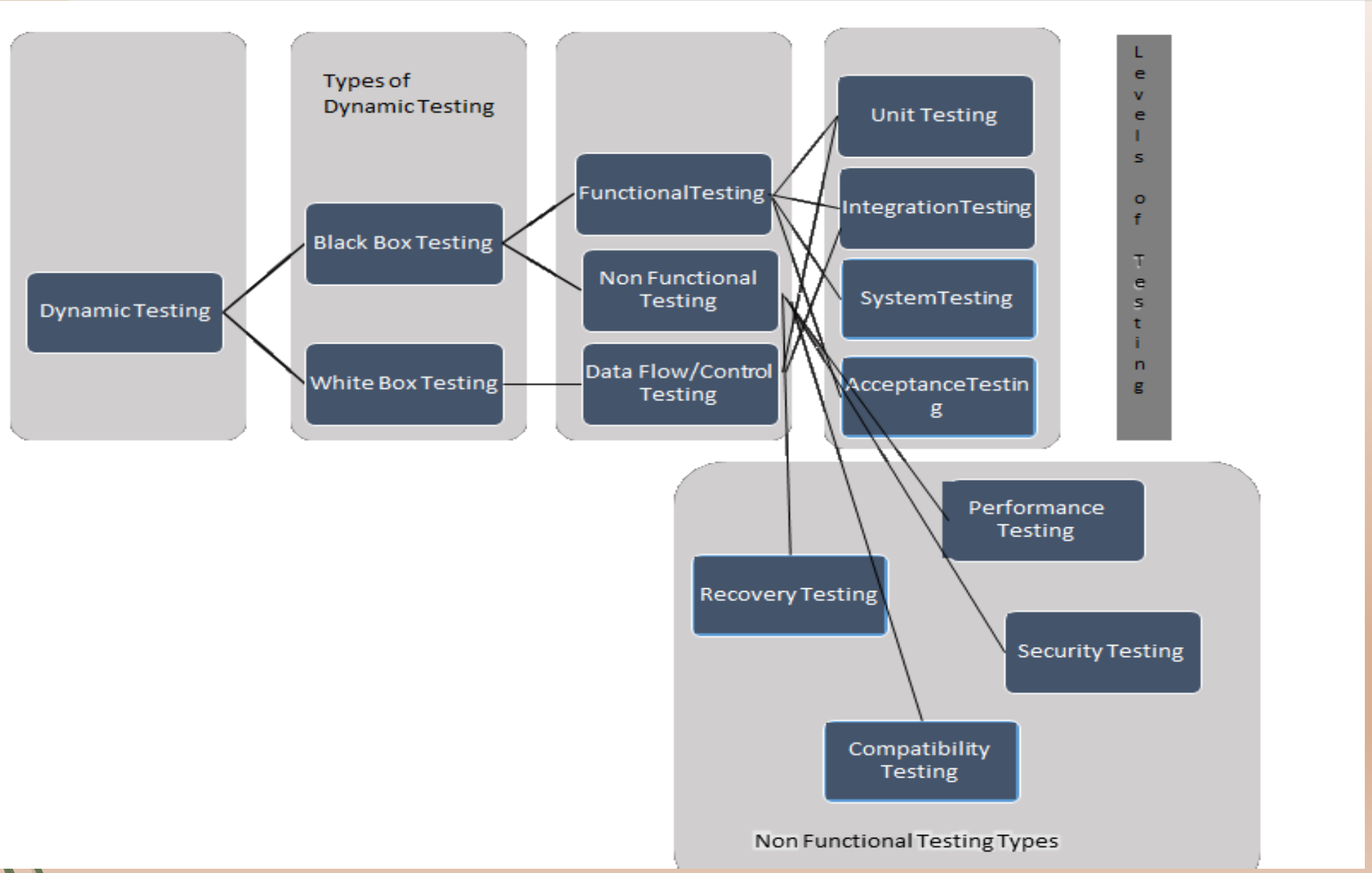
White Box Testing – White Box Testing is a software testing method in which the internal structure/ design is known to the tester. The main aim of White Box testing is to check on how System is performing based on the code. It is mainly performed by the Developers or White Box Testers who has knowledge on the programming.

Black Box Testing – Black Box Testing is a method of testing in which the internal structure/ code/design is **NOT** known to the tester. The main aim of this testing to verify the functionality of the system under test and this type of testing requires to execute the complete test suite and is mainly performed by the Testers, and there is no need of any programming knowledge.

The **Black Box** Testing

- Functional Testing
- Non-Functional Testing

DYNAMIC TESTING TYPES



DYNAMIC TESTING LEVEL OF TESTING

Functional Testing:

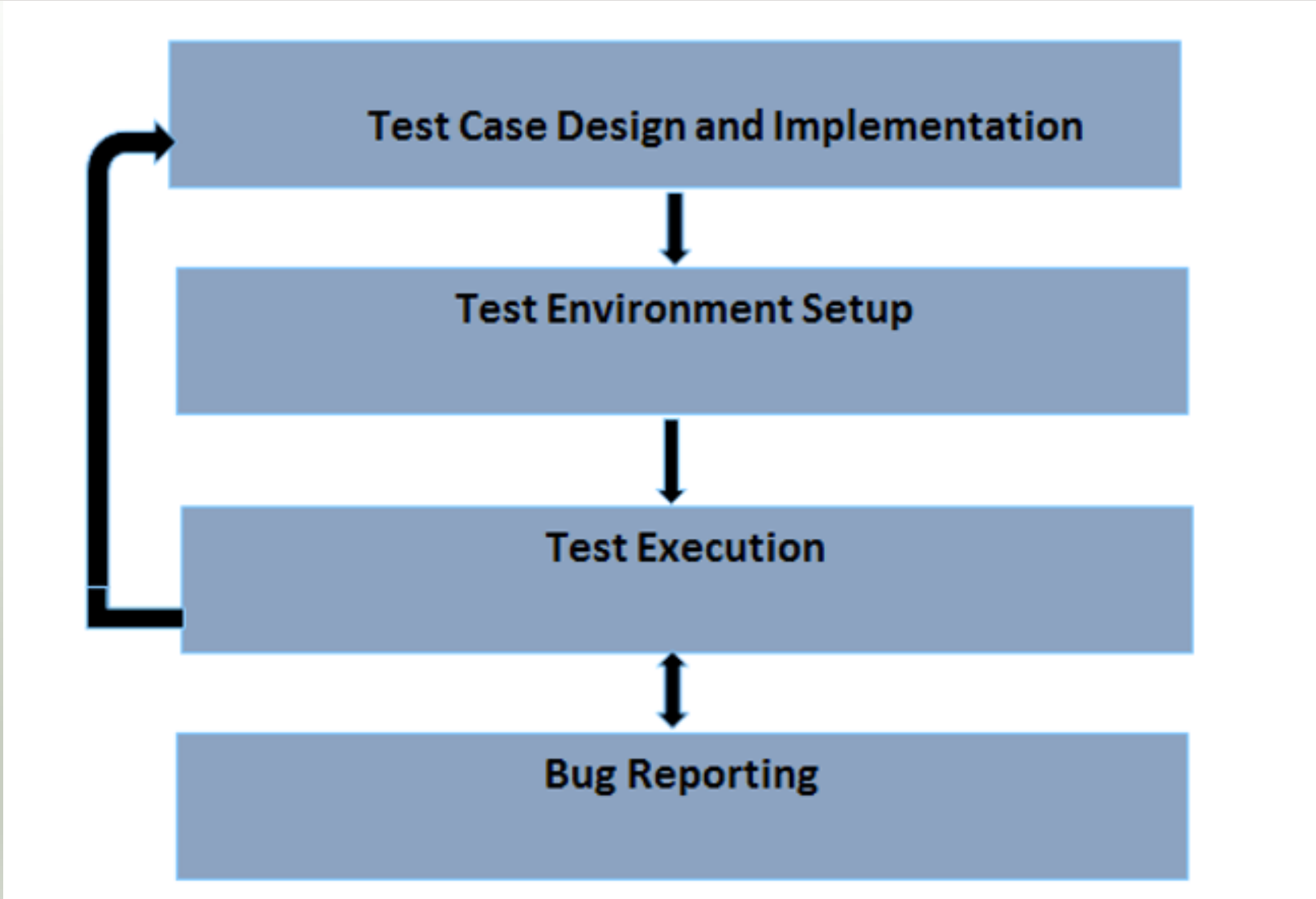
- **Unit Testing** – Generally Unit is a small piece of code which is testable, Unit Testing is performed at individual unit of software and is performed by developers
- **Integration Testing** – Integration Testing is the testing which is performed after Unit Testing and is performed by combining all the individual units which are testable and is performed either by developers or testers
- **System Testing** – System Testing is a performed to ensure whether the system performs as per the requirements and is generally performed when the complete system is ready, it is performed by testers when the Build or code is released to QA team
- **Acceptance Testing** – Acceptance testing is performed to verify whether the system has met the business requirements and is ready to use or ready for deployment and is generally performed by the end users.

DYNAMIC TESTING LEVEL OF TESTING

Non- Functional Testing:

- **Performance Testing** – Performance Testing is performed to check whether the response time of the system is normal as per the requirements under the desired network load.
- **Recovery Testing** – Recovery testing is a method to verify on how well a system is able to recover from crashes and hardware failures.
- **Compatibility Testing** – Compatibility testing is performed to verify how the system behaves across different environments.
- **Security testing** – Security testing is performed to verify the robustness of the application, i.e to ensure that only the authorizes users/roles are accessing the system
- **Usability testing** – Usability testing is a method to verify the usability of the system by the end users to verify on how comfortable the users are with the system.

DYNAMIC TESTING TECHNIQUES



DYNAMIC TESTING TECHNIQUES

1. **What is Test design and Implementation:**

   - In this phase we identify the,
   - Features to be tested
   - Derive the Test Conditions
   - Derive the coverage Items
   - Derive the Test Cases

2. **Test Environment Setup**

   We have to ensure that Testing Environment should always be similar to the Production environment, in this phase we have to install the build and manage the test machines.

DYNAMIC TESTING TECHNIQUES

3. **Test Execution**

During this phase, test cases are actually executed.

4. **4. Bug report captured**

Based on the Execution if the Expected and Actual Results are not same then the Test case has to be marked as Fail and a Bug should be logged.

Advantages of Dynamic Testing

- Dynamic Testing can reveal the uncovered defects that are considered to be too difficult or complicated and which cannot be covered through static Analysis
- In Dynamic Testing, we execute the software, end to end, ensuring error free software which in turn increases the quality of a product and project.
- Dynamic Testing becomes an essential Tool for detecting any security Threats

Disadvantages of Dynamic Testing

- Dynamic Testing is Time Consuming because it executes the application/software or code which requires huge amount of Resources
- Dynamic Testing increases the cost of project/product because it does not start early in the software lifecycle and hence any issues fixed in later stages can result in an increase of cost.

Q &A