UNIT-IV

SOFTWARE STATIC AND DYNAMIC ANALYSIS:

Static Analysis:

Static analysis involves no dynamic execution of the software under test and can detect possible defects in an early stage, before running the program.

Static analysis is done after coding and before executing unit tests.

Static analysis can be done by a machine to automatically "walk through" the source code and detect non complying rules. The classic example is a compiler which finds lexical, syntactic and even some semantic mistakes.

Static analysis can also be performed by a person who would review the code to ensure proper coding standards and conventions are used to construct the program.

Static code analysis advantages:

- It can find weaknesses in the code at the exact location.
- It can be conducted by trained software assurance developers who fully understand the code.
- Source code can be easily understood by other or future developers
- It allows a quicker turn around for fixes
- Weaknesses are found earlier in the development life cycle, reducing the cost to fix.
- Less defects in later tests
- Unique defects are detected that cannot or hardly be detected using dynamic tests
    - Unreachable code
    - Variable use (undeclared, unused)
    - Uncalled functions
    - Boundary value violations

Static code analysis limitations:

- It is time consuming if conducted manually.
- Automated tools produce false positives and false negatives.
- There are not enough trained personnel to thoroughly conduct static code analysis.
- Automated tools can provide a false sense of security that everything is being addressed.
- Automated tools only as good as the rules they are using to scan with.
- It does not find vulnerabilities introduced in the runtime environment.

Dynamic Analysis:

Dynamic analysis is based on the system execution, often using tools.

Dynamic program analysis is the analysis of computer software that is performed with executing programs built from that software on a real or virtual processor (analysis performed without executing programs is known as static code analysis). Dynamic program analysis tools may require loading of special libraries or even recompilation of program code.

The most common dynamic analysis practice is executing Unit Tests against the code to find any errors in code.

Dynamic code analysis advantages:

- It identifies vulnerabilities in a runtime environment.
- It allows for analysis of applications in which you do not have access to the actual code.
- It identifies vulnerabilities that might have been false negatives in the static code analysis.
- It permits you to validate static code analysis findings.
- It can be conducted against any application.

Dynamic code analysis limitations:

- Automated tools provide a false sense of security that everything is being addressed.
- Cannot guarantee the full test coverage of the source code
- Automated tools produce false positives and false negatives.

- Automated tools are only as good as the rules they are using to scan with.
- It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

CODE INSPECTIONS:

Code Inspection is the most formal type of review, which is a kind of static testing to avoid the defect multiplication at a later stage.

- The main purpose of code inspection is to find defects and it can also spot any process improvement if any.
- An inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.
- Preparation before the meeting is essential, which includes reading of any source documents to ensure consistency.
- Inspections are often led by a trained moderator, who is not the author of the code.
- The inspection process is the most formal type of review based on rules and checklists and makes use of entry and exit criteria.
- It usually involves peer examination of the code and each one has a defined set of roles.
- After the meeting, a formal follow-up process is used to ensure that corrective action is completed in a timely manner.

SOFTWARE TESTING FUNDAMENTALS:

Software testing is an activity performed to uncover errors. It is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The purpose of software testing is to ensure whether the software functions appear to be working according to specification and performance requirements.

Testing objective:

- Testing is a process of executing a program with the intend of finding an error.
- A good test case is one that has high probability of finding an undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.

Testing principles:

The following basic principles and fundamentals are general guidelines applicable for all types of real-time testing:

- Testing proves the presence of defects. It is generally considered better when a test reveals defects than when it is error-free.
- Testing the product should be accomplished considering the risk factor and priorities
- Early testing helps identify issues prior to the development stage, which eases error correction and helps reduce cost
- Normally a defect is clustered around a set of modules or functionalities. Once they are identified, testing can be focused on the defective areas, and yet continue to find defects in other modules simultaneously.
- Testing will not be as effective and efficient if the same kinds of tests are performed over a long duration.
- Testing has to be performed in different ways, and cannot be tested in a similar way for all modules. All testers have their own individuality, likewise the system under test.
- Just identifying and fixing issues does not really help in setting user expectations. Even if testing is performed to showcase the software's reliability, it is better to assume that none of the software products are bug-free.

SOFTWARE TEST PROCESS:

Testing is a process rather than a single activity. This process starts from test planning then designing test cases, preparing for execution and evaluating status till the test closure. So, we can divide the activities within the fundamental test process into the following basic steps:

- Planning and Control
- Analysis and Design
- Implementation and Execution
- Evaluating exit criteria and Reporting
- Test Closure activities

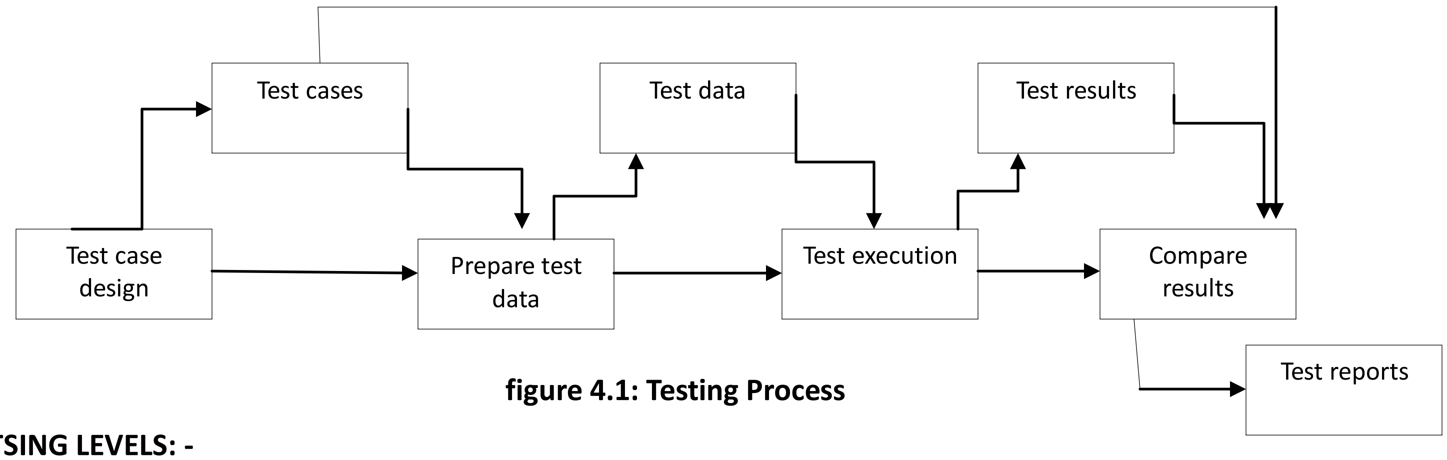

figure 4.1: Testing Process

TETSING LEVELS: -

The testing can be typically carried out in levels. In software development process at each phase some faults may get introduced. These faults are eliminated in the next software development phase but at the same time some new faults may get introduced. Each level of testing performs some typical activity. Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Unit Testing: In this type of testing errors are detected from each software component individually.

Integration Testing: In this type of testing technique interacting component are verified and the interface errors are detected.

System Testing: In this type of testing all the system elements forming the system is tested as a whole.

Acceptance Testing: Acceptance testing is a kind of testing conducted to ensure that the software works correctly in user's working environment.

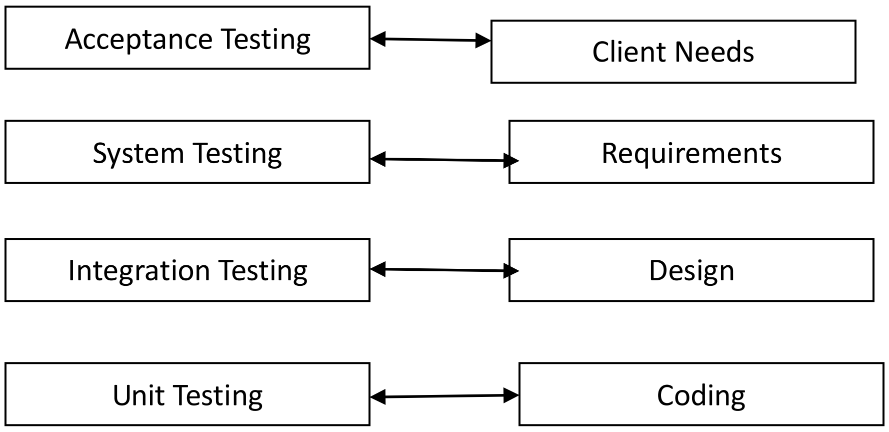

Figure 4.2: Levels of Testing

TEST CRITERIA AND TEST CASE DESIGN:

- Test cases are used to determine the presence of fault in the program.
- Executing test cases require money because- 1)machine time is required to execute test cases 2) Human efforts are involved in executing test case. Hence in the project testing minimum number of test vases should be there as far as possible.
- The testing activity should involve two goals-1) Maximize the number of errors detected. 2) Minimize the number of test cases.
- The selection of test case should be such that faulty module or program segment must be exercised by at least one test case.
- Test selection criterion can be defined as the set of conditions that must be satisfied by the set of test cases.
- Testing criterion is based on two fundamental properties – reliability and validity.
- A test criterion is reliable if all the test cases detect same set of errors.
- A test criterion is valid if, for any error in the program there is some test case which causes error in the program.
- Generating test cases to satisfy criteria is complex task.

TEST ORACLES:

Test Oracles is a mechanism for determining whether a test has passed or failed. The use of oracles involves comparing the output(s) of the system under test, for a given test-case input, to the output(s) that the oracle determines that product should have. Suppose we have written 2 test cases one test case is for the program which we want to test and other for the test oracle. If the output of both is the same then that means program behaves correctly otherwise there is some fault in the program.

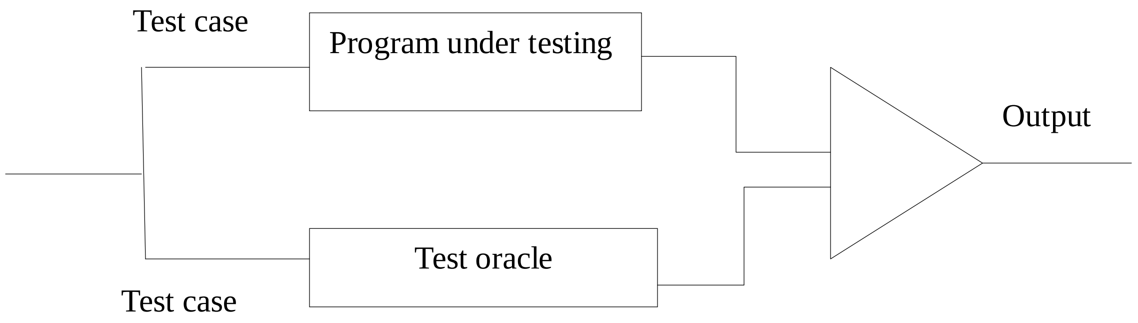

Figure 4.3: Testing with test oracle

TEST TECHNIQUES:

There are various testing techniques are available in which internal structure/design/implementation of the item being tested. There are different methods that can be used for software testing.

- Black box testing
- White box testing
- Integration testing
- Unit testing
- System testing

BLACK BOX TESTING:

Black box testing is also called as behavioral testing. Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle. This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

Black box testing uncovers following types of errors:

- Incorrect or missing functions
- Interface errors
- Errors in data structures
- Performance errors
- Initialization or termination errors

There are different techniques involved in Black Box testing.

- Equivalence partitioning
- Boundary Value Analysis
- Cause-Effect Graphing.
- Error-Guessing.

Advantages	Disadvantages
• Well suited and efficient for large code segments. • Code access is not required. • Clearly separates user's perspective from the developer's perspective through visibly defined roles. • Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	• Limited coverage, since only a selected number of test scenarios is actually performed. • Inefficient testing, due to the fact that the tester only has limited knowledge about an application. • Blind coverage, since the tester cannot target specific code segments or error-prone areas. • The test cases are difficult to design.

WHITE BOX TESTING:

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

White box testing techniques includes:

- **Statement Coverage -** This technique is aimed at exercising all programming statements with minimal tests.
- **Branch Coverage -** This technique is running a series of tests to ensure that all branches are tested at least once.
- **Path Coverage -** This technique corresponds to testing all possible paths which means that each statement and branch is covered.

Advantages	Disadvantages
• As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	• Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased. • Sometimes it is impossible to look into every

• It helps in optimizing the code. • Extra lines of code can be removed which can bring in hidden defects. • Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	nook and corner to find out hidden errors that may create problems, as many paths will go untested. • It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

UNIT TESTING:

Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.
The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Advantages:

- Reduces Defects in the newly developed features or reduces bugs when changing the existing functionality.
- Reduces Cost of Testing as defects are captured in very early phase.
- Improves design and allows better refactoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

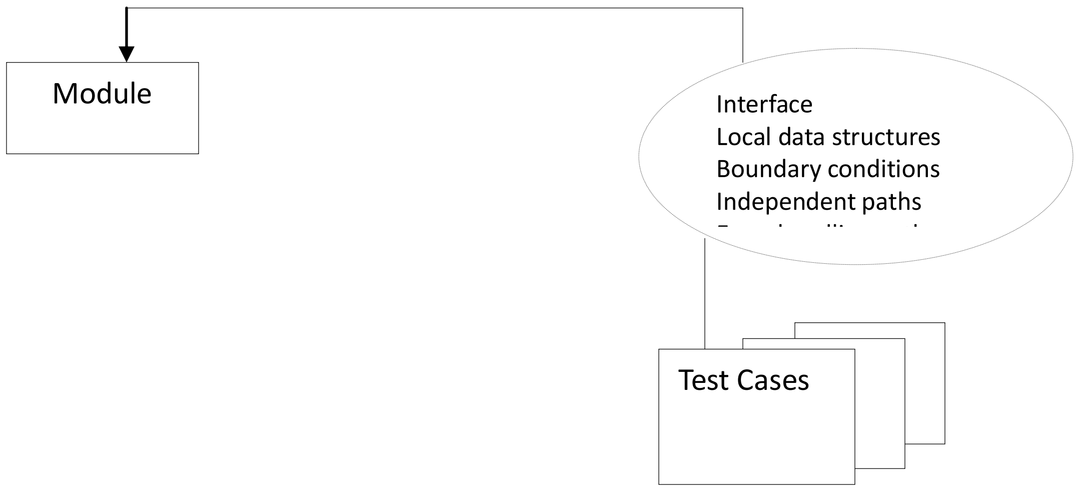

Figure 4.4: Unit Testing

Unit Testing Techniques:

- **Black Box Testing -** Using which the user interface, input and output are tested.
- **White Box Testing -** used to test each one of those functions behavior is tested.
- **Gray Box Testing -** Used to execute tests, risks and assessment methods.

TESTING FRAMEWORKS:

Testing frameworks are an essential part of any successful automated testing process. They can reduce maintenance costs and testing efforts and will provide a higher return on investment (ROI) for QA teams looking to optimize their agile processes.
A testing framework is a set of guidelines or rules used for creating and designing test cases. A framework is comprised of a combination of practices and tools that are designed to help QA professionals test more efficiently.

INTEGRATION TESTING:

In integration testing, individual software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. Integration Testing focuses on checking data communication amongst these modules.

Need of integration testing:

Although each software module is unit tested, defects still exist for various reasons like:

- A Module in general is designed by an individual software developer whose understanding and programming logic may differ from other programmers. integration Testing becomes necessary to verify the software modules work in unity
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues.

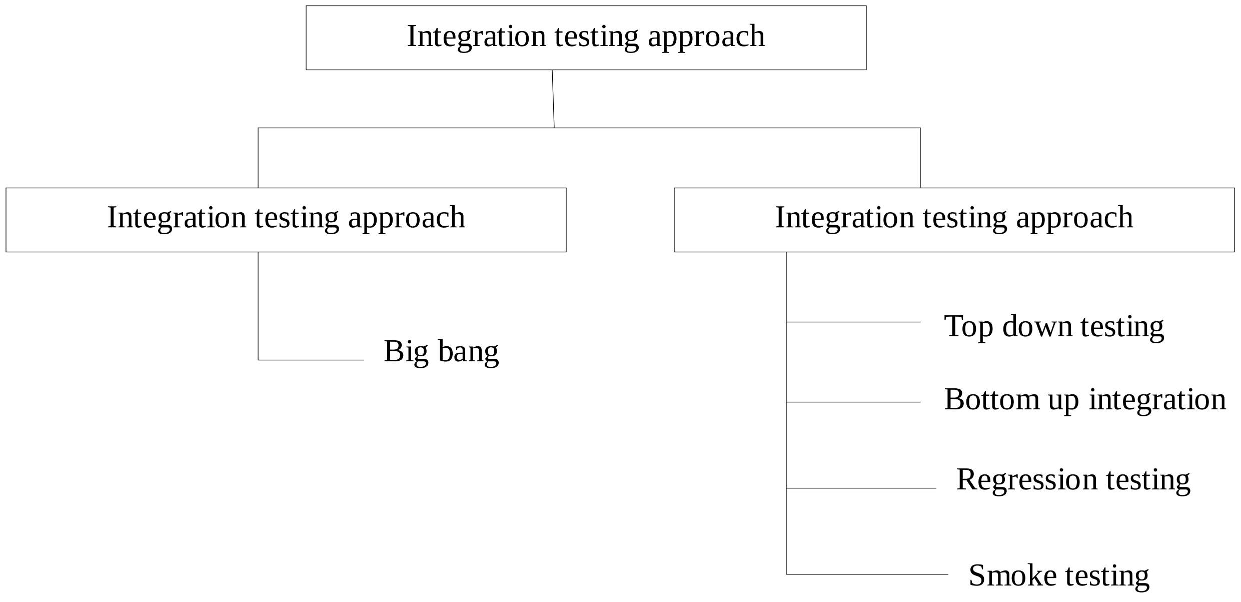
Integration testing approach
Integration testing approach
Integration testing approach
Big bang
Top down testing
Bottom up integration
Regression testing
Smoke testing

Figure 4.5: Integration testing approach

Bottom-up integration:
This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
Top-down integration:
In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.
In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.
Regression Testing:
Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
Smoke Testing:
Smoke testing is a type of software testing which ensures that the major functionalities of the application are

working fine. This testing is also known as ‘Build Verification testing’. It is a non-exhaustive testing with very limited test cases to ensure that the important features are working fine and we are good to proceed with the detailed testing.

SYSTEM TESTING:

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons:

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

OTHER SPECIALIZED TESTING:

There are lots of testing types. Below we have listed types of system testing a large software development company would typically use:

Usability Testing: Usability Testing mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives

Load Testing: Load Testing is necessary to know that a software solution will perform under real-life loads.

Regression Testing: Regression Testing involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.

Recovery Testing: Recovery testing is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.

Migration Testing: Migration testing is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.

Functional Testing: Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.

Hardware/Software Testing: IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

Acceptance Testing: The acceptance testing is a kind of testing to ensure that the software works correctly in the user work environment. Acceptance testing, a testing technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.

There are various forms of acceptance testing:

- User acceptance Testing
- Business acceptance Testing
- Alpha Testing
- Beta Testing

Stress Testing: Stress testing is the process of determining the ability of a computer, network, program or device to maintain a certain level of effectiveness under unfavorable conditions. It is used to test the stability & reliability of the system. This test mainly determines the system on its robustness and error handling under extremely heavy load conditions.

TEST PLAN:

Test planning, the most important activity to ensure that there is initially a list of tasks and milestones in a baseline plan to track the progress of the project. It also defines the size of the test effort. It is the main document often called as master test plan or a project test plan and usually developed during the early phase of the project.

S.No.	Parameter	Description
1.	Test plan identifier	Unique identifying reference.
2.	Introduction	A brief introduction about the project and to the document.
3.	Test items	A test item is a software item that is the application under test.
4.	Features to be tested	A feature that needs to tested on the test ware.
5.	Features not to be tested	Identify the features and the reasons for not including as part of testing.
6.	Approach	Details about the overall approach to testing.
7.	Item pass/fail criteria	Documented whether a software item has passed or failed its test.
8.	Test deliverables	The deliverables that are delivered as part of the testing process, such as test plans, test specifications and test summary reports.
9.	Testing tasks	All tasks for planning and executing the testing.
10.	Environmental needs	Defining the environmental requirements such as hardware, software, OS, network configurations, tools required.
11.	Responsibilities	Lists the roles and responsibilities of the team members.
12.	Staffing and training needs	Captures the actual staffing requirements and any specific skills and training requirements.
13.	Schedule	States the important project delivery dates and key milestones.
14.	Risks and Mitigation	High-level project risks and assumptions and a mitigating plan for each identified risk.
15.	Approvals	Captures all approvers of the document, their titles and the sign off date.

Table 4.1 Test Plan Identifiers

Test Planning Activities:

- To determine the scope and the risks that need to be tested and that are NOT to be tested.
- Documenting Test Strategy.
- Making sure that the testing activities have been included.
- Deciding Entry and Exit criteria.
- Evaluating the test estimate.
- Planning when and how to test and deciding how the test results will be evaluated, and defining test exit criterion.
- The Test art facts delivered as part of test execution.
- Defining the management information, including the metrics required and defect resolution and risk issues.
- Ensuring that the test documentation generates repeatable test assets.

TEST METRICS:

In software testing, Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute. Measurement is nothing but quantitative indication of size / dimension / capacity of an attribute of a product / process. Software metric is defined as a quantitative measure of an

attribute a software system possesses with respect to Cost, Quality, Size and Schedule.
Example-
Measure - No. of Errors
Metrics - No. of Errors found per person
The most commonly used metric is cyclomatic complexity and Hallstead complexity.
Cyclomatic complexity: Cyclomatic complexity is software metric used to measure the complexity of a program. These metric, measures independent paths through program source code. Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths. Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.
In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.

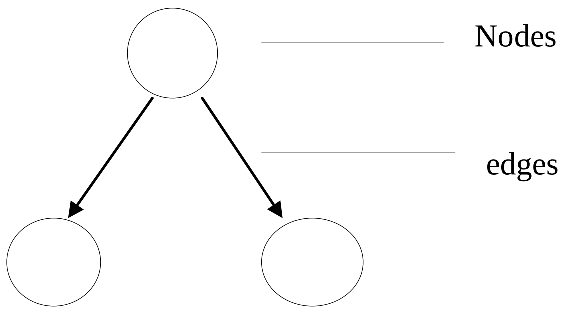

Figure 4.6: Nodes representation

Mathematically, it is set of independent paths through the graph diagram. The complexity of the program can be defined as -

$$V(G) = E - N + 2$$

Where,
E - Number of edges
N - Number of Nodes

$$V(G) = P + 1$$

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

Halstead complexity: Halstead complexity measurement was developed to measure a program module's complexity directly from source code, with emphasis on computational complexity.
The Halstead effort can be defined as:

$$e = V/PL$$

Where V is the program volume and Pl is the program level. The program level can be computed as:
$PL=1/[(n1/2)*(N2/n2)]$
Where n1 is total distinct operators,
N2 is total distinct operands and
N2 is all the operand in the program.
The % of overall testing efforts = testing effort of specific module/testing efforts of all the modules

<u>TESTING TOOLS:</u>
Tools from a software testing context can be defined as a product that supports one or more test activities right from planning, requirements, creating a build, test execution, defect logging and test analysis.
Classification of Tools
Tools can be classified based on several parameters. They include:

- The purpose of the tool
- The Activities that are supported within the tool
- The Type/level of testing it supports

- The Kind of licensing (open source, free ware, commercial)

The technology used

S. No.	Tool Type	Used for	Used by
1.	Test Management Tool	Test Managing, scheduling, defect logging, tracking and analysis.	testers
2.	Configuration management tool	For Implementation, execution, tracking changes	All Team members
3.	Static Analysis Tools	Static Testing	Developers
4.	Test data Preparation Tools	Analysis and Design, Test data generation	Testers
5.	Test Execution Tools	Implementation, Execution	Testers
6.	Test Comparators	Comparing expected and actual results	All Team members
7.	Cover age measurement tools	Provides structural coverage	Developers
8.	Performance Testing tools	Monitoring the performance, response time	Testers
9.	Project planning and Tracking Tools	For Planning	Project Managers
10.	Incident Management Tools	For managing the tests	Testers

Table 4.2: Testing tool

Tools Implementation - process

- Analyze the problem carefully to identify strengths, weaknesses and opportunities.
- The Constraints such as budgets, time and other requirements are noted.
- Evaluating the options and Short listing the ones that are meets the requirement.
- Developing the Proof of Concept which captures the pros and cons.
- Create a Pilot Project using the selected tool within a specified team.
- Rolling out the tool phase wise across the organization.

<u>INTRODUCTION TO OBJECT-ORIENTED ANALYSIS AND DESIGN:</u>

Object-oriented analysis and design (OOAD) is a popular technical approach to analyzing, designing an application, system, or business by applying the object-oriented paradigm and visual modeling throughout the development life cycles for better stakeholder communication and product quality.

Object-Oriented Analysis:

Object–Oriented Analysis (OOA) is the procedure of identifying software engineering requirements and developing software specifications in terms of a software system's object model, which comprises of interacting objects.

The primary tasks in object-oriented analysis (OOA) are –

- Identifying objects
- Organizing the objects by creating object model diagram
- Defining the internals of the objects, or object attributes
- Defining the behavior of the objects, i.e., object actions
- Describing how the objects interact

The common models used in OOA are use cases and object models.

Object-Oriented Design:

Object–Oriented Design (OOD) involves implementation of the conceptual model produced during object-oriented analysis. In OOD, concepts in the analysis model, which are technology–independent, are mapped onto implementing classes, constraints are identified and interfaces are designed, resulting in a model for the solution domain.

The implementation details generally include –

- Restructuring the class data (if necessary),
- Implementation of methods, i.e., internal data structures and algorithms,
- Implementation of control, and
- Implementation of associations.

COMPARISON WITH STRUCTURED SOFTWARE ENGINEERING:

Key differences between structured and object oriented analysis and design are as follows:

Phase	**Structured**	**Object Oriented**
Analysis	Structuring Requirements • DFD's • Structured English • Decision Table/Tree • ER Analysis	Requirement Engineering • Use Case Model(Find use cases, flow of events) • Object Model – Find classes and class relationship – Object interaction: Sequence and collaboration diagram ,state machine diagram – Object to ER mapping
Design	DB Design • DB normalization GUI Design • Forms and reports	Physical DB design Design elements • Design system architecture • Design classes • Design components GUI Design

Table 4.3: comparison table between structured and object oriented