# Software Engineering
# Unit - IV

Compiled by
**M S Anand**

Department of Computer Science

# Software Engineering
## Introduction

**Text Book(s):**

1. “Software Engineering: Principles and Practice", Hans van Vliet, Wiley India, 3rd Edition, 2010.
2. “Software Testing – Principles and Practices”, Srinivasan Desikan and Gopalaswamy Ramesh, Pearson, 2006.

**Reference Book(s):**

1. “Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling” by By Jennifer Davis, Ryn Daniels, O’ Reilly Publications, 2018
2. “Software Engineering: A Practitioner’s Approach”, Roger S Pressman, McGraw Hill, 6th Edition 2005
3. “Software Engineering”, International Computer Science Series, Ian Somerville, Pearson Education, 9th Edition, 2009.
4. “Foundations of Software Testing ", Aditya Mathur, Pearson, 2008
5. “Software Testing, A Craftsman’s Approach ", Paul C. Jorgensen, Auerbach, 2008.
6. IEEE SWEBOK, PMBOK, BABOK and Other Sources from Internet.

## Introduction

Software testing is a process of examining the functionality and behavior of the software through verification and validation.

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose.

Barry Boehm described verification and validation as the following:
Verification**:** Are we building the product right?
Validation**:** Are we building the right product?

## Software Testing

Verification is a process of determining if the software is designed and developed as per the specified requirements.

Validation is the process of checking if the software (end product) has met the client's true needs and expectations.

Software testing is incomplete until it undergoes verification and validation processes. Verification and validation are the main elements of software testing workflow because they:

1. Ensure that the end product meets the design requirements.
2. Reduce the chances of defects and product failure.
3. Ensure that the product meets the quality standards and expectations of all stakeholders involved.

According to a survey, the software testing industry is estimated to grow from $40 billion in 2020 to $60 billion in 2027.

## Software Testing

Software testing also involves measuring attributes to build confidence in the software.

**“Testing software shows only the presence of errors, not their absence”**

## What are the real objectives of testing?

Demonstration

- System used with acceptable risk
- Functions under special conditions
- Product ready for integration/use

Detection

- Discover defects, errors and deficiencies
- Determine capabilities and limitations
- Quality of components

Prevention

- Information to prevent/reduce errors
- Reduce error propagation
- Clarify System specifications and performance
- Identify ways to avoid risks and problems.

## Verification

Are we building the product right?

Process of checking that software achieves its goals to ensure that products and deliverables meet requirements.

Static Testing - Does not include execution of code

Checking documents, design, code (reviews, walkthroughs, inspections)

Finds the bugs early in development

Targets software architecture, design, database, etc.

Occurs before validation

## Validation

Are we building the right product?
Focuses on product related activities that determine if the system or project deliverables meet customer/client expectations

Dynamic Testing - Includes execution of code

Validates the capabilities and features in project scope and requirements

Typically done by testing team

Methods: Black Box testing, White Box testing, and non-functional Testing.

## Software Engineering
## Verification & Validation (V & V)

| | Verification | Validation |
|---|---|---|
| Definition | It is a process of checking if a product is developed as per the specifications. | It is a process of ensuring that the product meets the needs and expectations of stakeholders. |
| What it tests or checks for | It tests the requirements, architecture, design, and code of the software product. | It tests the usability, functionalities, and reliability of the end product. |
| Coding requirement | It does not require executing the code. | It emphasizes executing the code to test the usability and functionality of the end product. |
| Activities include | A few activities involved in verification testing are requirements verification, design verification, and code verification. | The commonly-used validation activities in software testing are usability testing, performance testing, system testing, security testing, and functionality testing. |

## Verification & Validation (V & V)

| | Verification | Validation |
|---|---|---|
| Types of testing methods | A few verification methods are inspection, code review, desk-checking, and walkthroughs. | A few widely-used validation methods are black box testing, white box testing, integration testing, and acceptance testing. |
| Teams or persons involved | The quality assurance (QA) team would be engaged in the verification process. | The software testing team along with the QA team would be engaged in the validation process. |
| Target of test | It targets internal aspects such as requirements, design, software architecture, database, and code. | It targets the end product that is ready to be deployed. |

## V & V

Different project management and software development methods use verification and validation in different ways.

For instance, both verification and validation happen simultaneously in agile development methodology due to the need for continuous refinement of the system based on the end-user feedback.

## Testing related terminology

Testing is the process of identifying defects, where a defect is any variance between actual and expected results.

“A mistake in coding is called Error, an error found by a tester is called a Defect, a defect accepted by development team is called a Bug and if a build does not meet the requirements, then it is Failure.”

**Defect:**
It can be simply defined as a variance between expected and actual. The defect is an error found AFTER the application goes into production. It commonly refers to several troubles with the software products, with their external behavior or with its internal features. In other words, a Defect is a difference between expected and actual results in the context of testing. It is the deviation from the customer requirement.

## Testing related terminology

**Error:**

An error is a mistake, misconception, or misunderstanding on the part of a software developer. In the category of the developer, we include software engineers, programmers, analysts, and testers. For example, a developer may misunderstand a design notation, or a programmer might type a variable name incorrectly – leads to an Error. It is the one that is generated because of the wrong login, loop or syntax. The error normally arises in software; it leads to a change in the functionality of the program.

**Bug:**

A bug is the result of a coding error. An Error found in the development environment before the product is shipped to the customer. A programming error that causes a program to work poorly, produce incorrect results or crash. An error in software or hardware that causes a program to malfunction. A bug is the terminology of Tester.

## Testing related terminology

**Failure:**

A failure is the inability of a software system or component to perform its required functions within specified performance requirements. When a defect reaches the end customer it is called a Failure. During development, Failures are usually observed by testers.

**Fault:**

An incorrect step, process or data definition in a computer program that causes the program to perform in an unintended or unanticipated manner. A fault is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification. It is the result of the error.

**Issue**

Raised by end user when product doesn’t meet expectations.

## Characterizing testing

There are different types of testing (will discuss later). How do we characterize these?

Why

Observations based on test objective

Fault identification, Fitness for use

How did we arrive at Test Cases

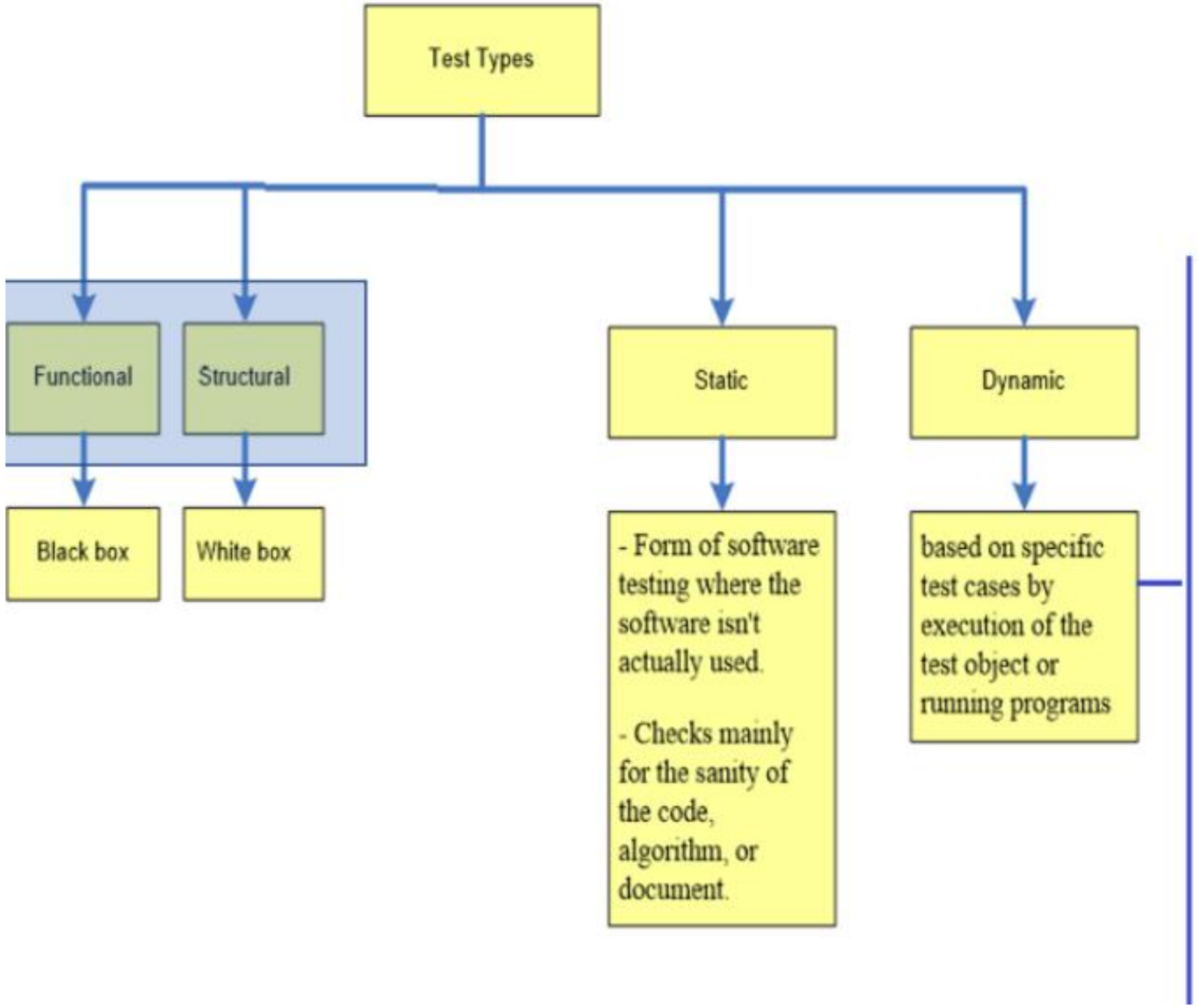
Based on test strategy or approach

Which of the samples

Should we observe based on test selection

Random, Ad-Hoc, Algorithmic, Statistical

## Software Engineering
## Broad categorization



Test Types
Functional
Structural
Static
Dynamic
Black box
White box
- Form of software testing where the software isn't actually used.
- Checks mainly for the sanity of the code, algorithm, or document.
based on specific test cases by execution of the test object or running programs

**Levels (Lifecycle) of testing Types**

- UNIT testing
- Integration teting
- System/Functional testing
- Acceptance Testing

**Technique based testing type**

- Coverage based testing
- Fault based testing

**Manual**
**Automatic**

## Software Engineering
## Characterizing testing

A note on test case

**What is a test case?**

Test cases define how to test a system, software or an application. A test case is a singular set of actions or instructions for a tester to perform that validates a specific aspect of a product or application functionality. If the test fails, the result might be a software defect that the organization can triage.

A tester or QA professional typically writes test cases, which are run after the completion of a feature or the group of features that make up the release. Test cases also confirm whether the product meets its software requirements.

A group of test cases is organized in a test suite, which tests a logical segment of the application, such as a specific feature.

## Software Engineering
## Characterizing testing

<u>Test case format</u>

Test case documentation typically includes all the pertinent information to run and collect data from the test. While the specific test case format might differ between organizations, most include the following details:

**Module name**. This is the module or feature under test.

**Test ID and/or name**. This is a unique identifier that should follow a standard naming convention.

**Tester name**. The person conducting the test.

**Test data**. This describes the dataset(s) to use for the test.

**Assumptions or preconditions**. Describe the various steps that must be accomplished prior to testing, or what we can assume situationally about the test, such as “after a successful login.”

**Test priority**. Define whether the test is low, medium or high priority.

**Test scenarios**. This is the high-level action from which the test case derives.

**Testing environment**. Identify the name and/or characteristics of the environment for testing.

**Testing steps**. Detail the steps for the tester to follow in the desired order.

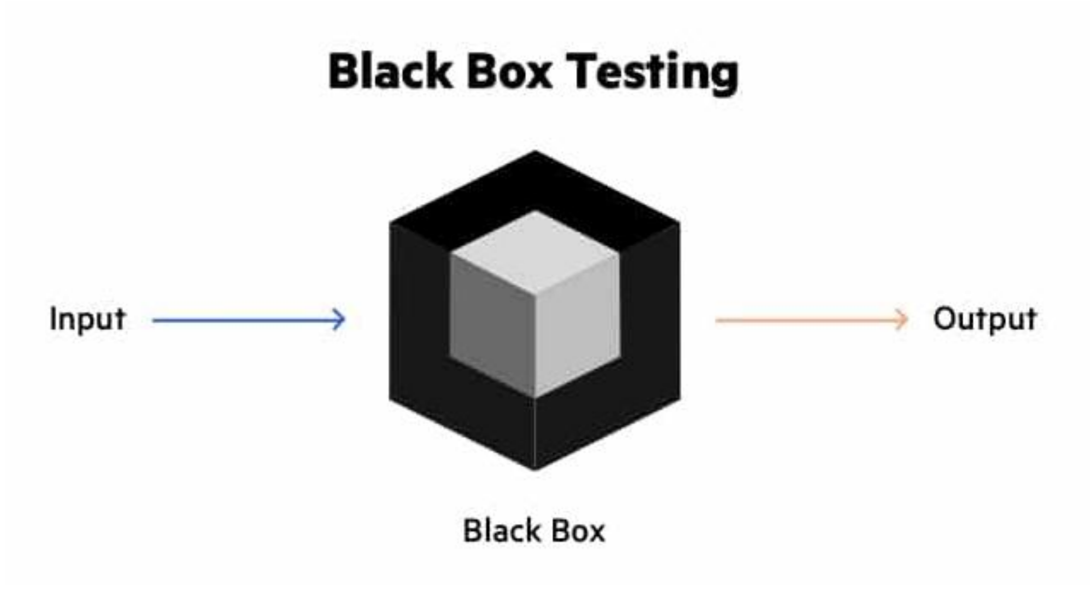
**Expected results**. This is the output you expect to receive from the system.

**Actual results**. This is the output you actually receive from the system.

**Pass/fail determination**. If the actual results match the expected results, the test passes. If not, the test fails.

## Black box testing

Black box testing involves testing a system with no prior knowledge of its **internal** workings.
A tester provides an input, and observes the output generated by the system under test.
This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.



## Black box testing

Black Box Testing – Functional
Treats software as a black box without regard to the internal structure or logic
Concerned with external behavior of the product
Objective: <u>to identify defects in the output, as a result of valid and invalid input</u>
Tested from user's point of view
Test cases are provided to the tester
Test case result is verified
<u>Advantages</u>

- ➢Best for larger units of code
- ➢Early test planning

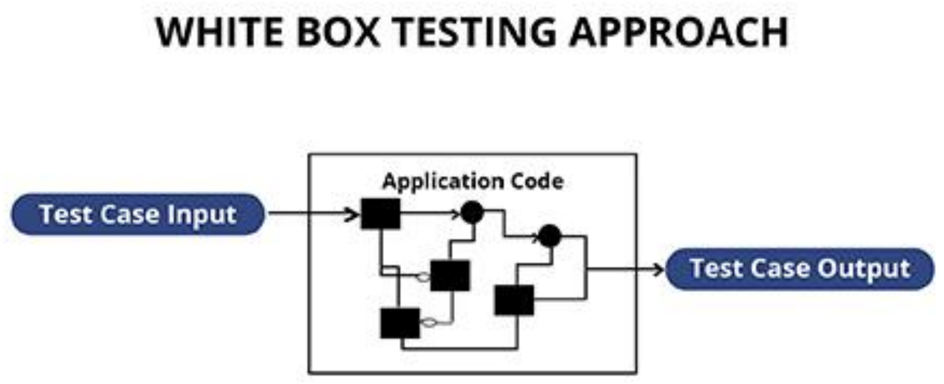
<u>Disadvantages</u>

- ➢Some paths may not be tested
- ➢Hard to direct tests to error prone code

## White Box Testing - Structural

White box testing is an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations with external systems.

White box testing is an essential part of automated build processes in a modern Continuous Integration/Continuous Delivery (CI/CD) development pipeline.



## White Box Testing - Structural

White Box Testing – Structural
Factors in the internal logic and structure of the code
Test specifier uses knowledge of internal structure to derive test cases
Test cases cannot be determined until code has been written
Testing from developer’s point of view
Testers need to have programming skills
Advantages

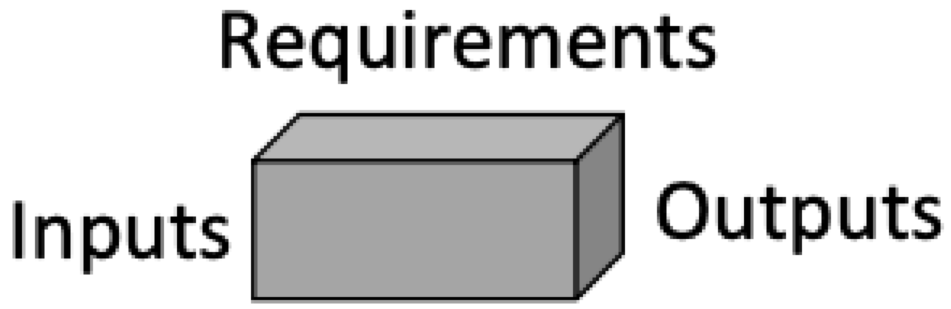
- ➢Partitioning by execution equivalence
- ➢Reveals hidden errors

Disadvantages

- ➢Needs skilled testers
- ➢Hard to test all of the code

## Grey Box Testing - Structural

Grey Box Testing – Functional & Structural
Involves having access to internal data structures and algorithms to build test cases
Testing is done at the user level
Integration testing between modules
Runs tests as a user and compares to a database



## Static and Dynamic Testing

Static testing is “Verification”.
Dynamic testing is “Validation”.

Static testing

Checks for defects in the software without executing the code.
Performed in early stages of development to avoid errors

➢ Easier to find the source and solution when discovered earlier

Types of Static testing

- Document reviews
- Walkthroughs
- Inspection
- Feasibility analysis or any other form of analysis to determine if the software is what it should be or not
- Code review

## Static and Dynamic Testing

Static analysis

Evaluation of code quality
Data flow
Control flow
Cyclomatic complexity

Note:
Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program.

***Cyclomatic complexity, M = E – N + 2P,*** *where,*

*E = the number of edges in the control flow graph*
*N = the number of nodes in the control flow graph*
*P = the number of connected components*

## Static and Dynamic Testing

Control Flow Graph

A **Control Flow Graph (CFG)** is the graphical representation of control flow or computation during the execution of programs or applications.

**Characteristics of Control Flow Graph:**

1. Control flow graph is process oriented.
2. Control flow graph shows all the paths that can be traversed during a program execution.
3. Control flow graph is a directed graph.
4. Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

There exist 2 designated blocks in Control Flow Graph:

**Entry Block:**

Entry block allows the control to enter into the control flow graph.

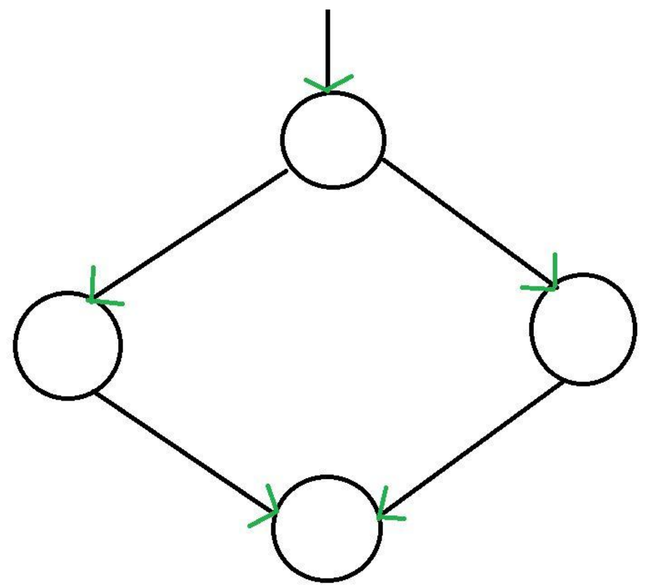
**Exit Block:**

Control flow leaves through the exit block.

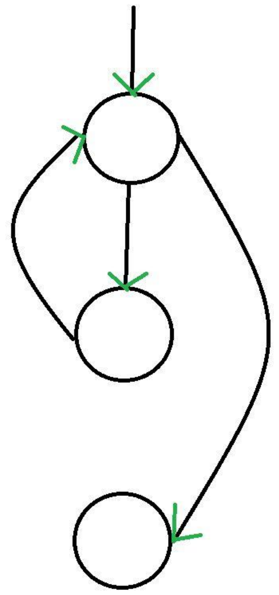
Hence, the control flow graph is comprised of all the building blocks involved in a flow diagram such as the start node, end node and flows between the nodes.

## Static and Dynamic Testing

**General Control Flow Graphs:**
Control Flow Graph is represented differently for all statements and loops. Following images describe it:

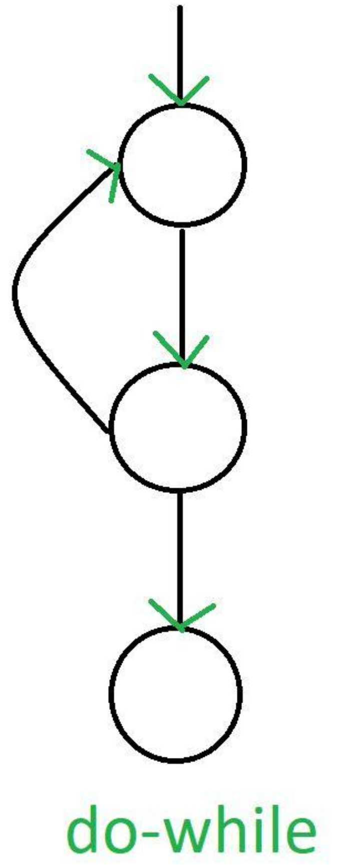


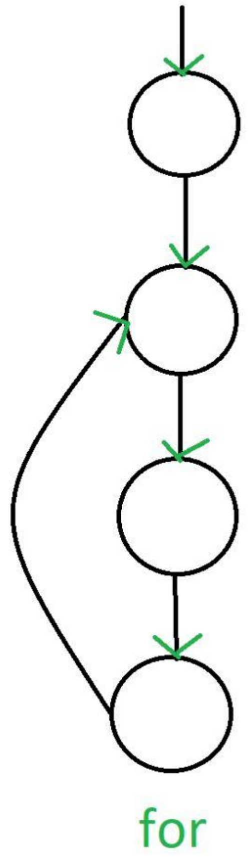
If-then-else



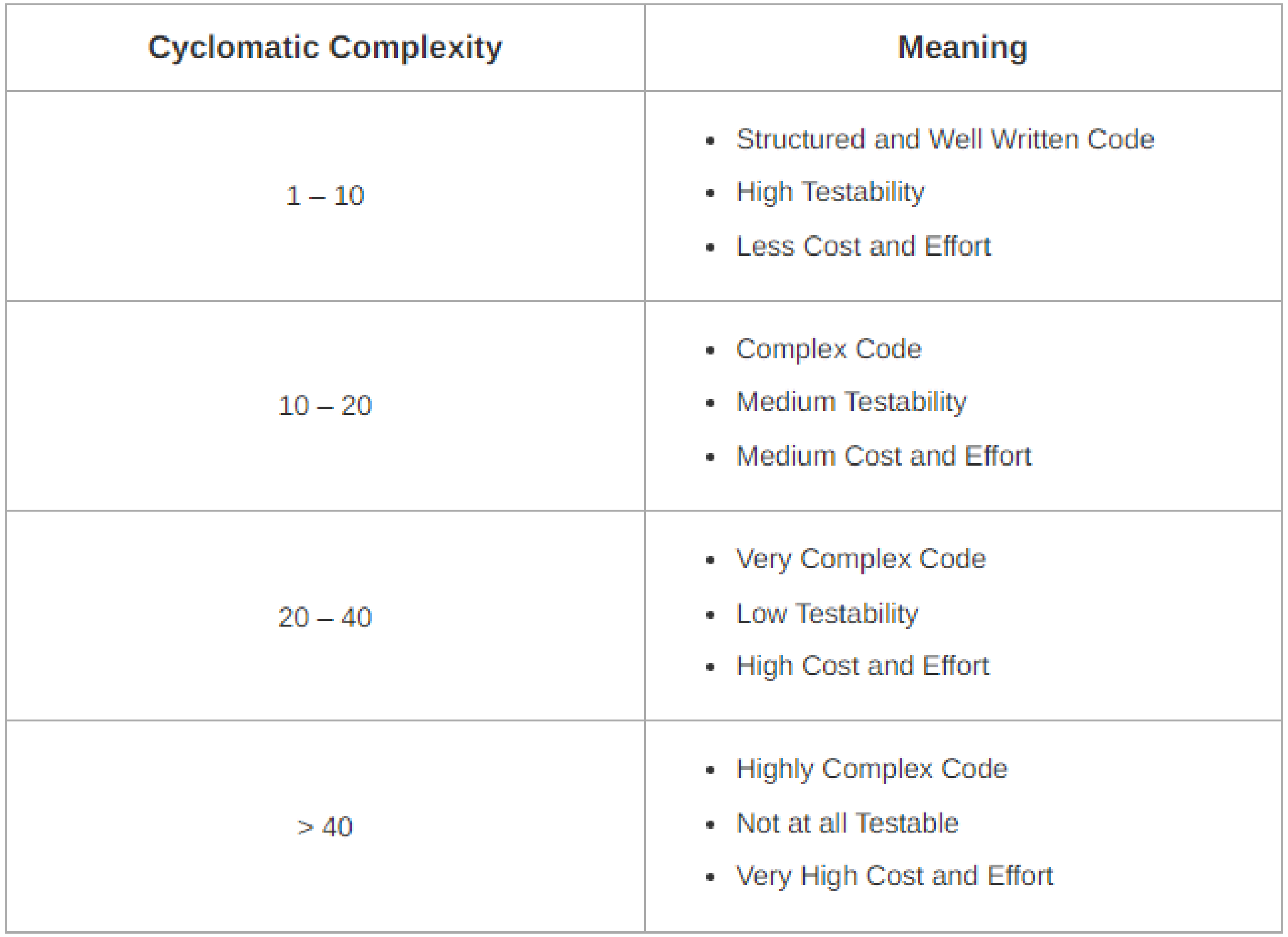
while

## Static and Dynamic Testing



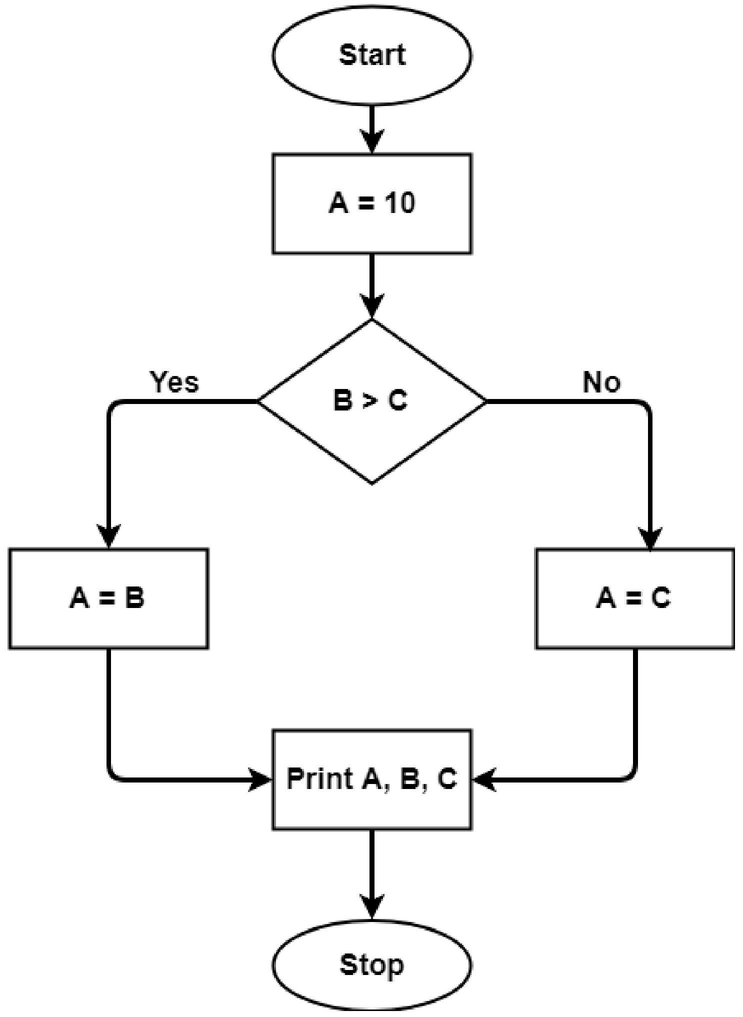


## Software Engineering
## Static and Dynamic Testing



| Cyclomatic Complexity | Meaning |
|---|---|
| 1 – 10 | • Structured and Well Written Code<br>• High Testability<br>• Less Cost and Effort |
| 10 – 20 | • Complex Code<br>• Medium Testability<br>• Medium Cost and Effort |
| 20 – 40 | • Very Complex Code<br>• Low Testability<br>• High Cost and Effort |
| > 40 | • Highly Complex Code<br>• Not at all Testable<br>• Very High Cost and Effort |

## Software Engineering
## Static and Dynamic Testing



For the control graph, shown in the figure:

The graph shows seven shapes(nodes), seven lines(edges), hence cyclomatic complexity is

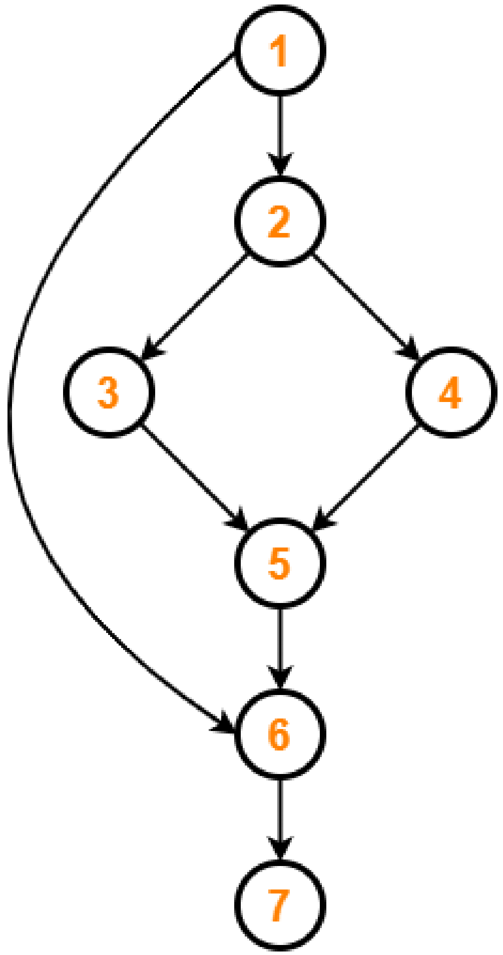
7-7+2 = 2.

## Static and Dynamic Testing

For the given code, draw the control flow graph and compute the cyclomatic complexity of the code.

```
IF A = 354
THEN IF B > C
THEN A = B
ELSE A = C
END IF
END IF
PRINT A
```

## Static and Dynamic Testing



Control Flow Graph

Software Engineering

## Static and Dynamic Testing

### Computing Cyclomatic Complexity

Cyclomatic Complexity

$= E – N + 2$

$= 8 – 7 + 2$

$= 3$

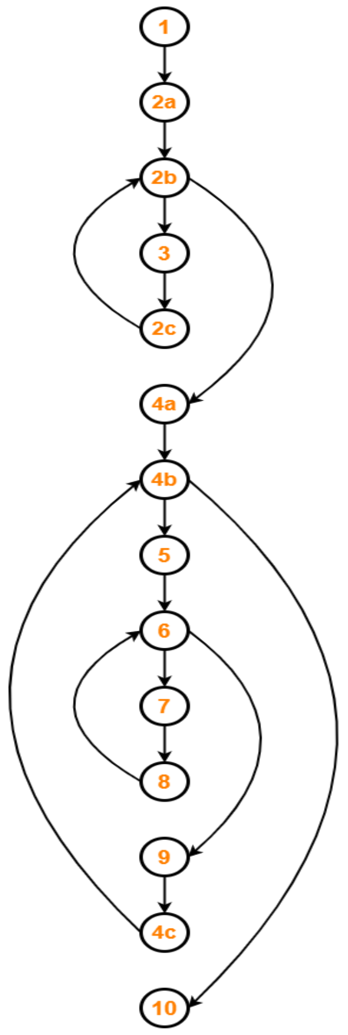
# Software Engineering

## Static and Dynamic Testing

### Calculate Cyclomatic Complexity

```
{ int i, j, k;
for (i=0 ; i<=N ; i++)
p[i] = 1;
for (i=2 ; i<=N ; i++)
{
k = p[i]; j=1;
while (a[p[j-1]] > a[k] {
p[j] = p[j-1];
j--;
}
p[j]=k;
}
```

## Software Engineering
## Static and Dynamic Testing



Control Flow Graph

## Static and Dynamic Testing

### Computing Cyclomatic Complexity

Cyclomatic Complexity

$= E – N + 2$

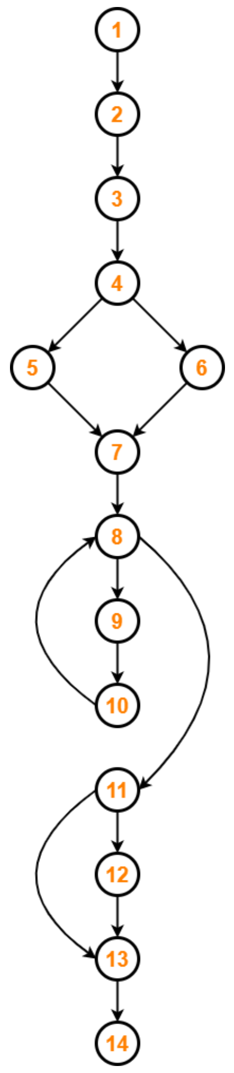
$= 16 – 14 + 2$

$= 4$

## Calculate Cyclomatic Complexity

```
begin int x, y, power;
float z;
input(x, y);
if(y<0)
power = -y;
else power = y;
z=1;
while(power!=0)
{    z=z*x;
power=power-1;
} if(y<0)
z=1/z;
output(z);
end
```

## Software Engineering
## Static and Dynamic Testing



## Computing Cyclomatic Complexity

Cyclomatic Complexity
= E – N + 2
= 16 – 14 + 2
= 4

## Software Engineering
## Static and Dynamic Testing

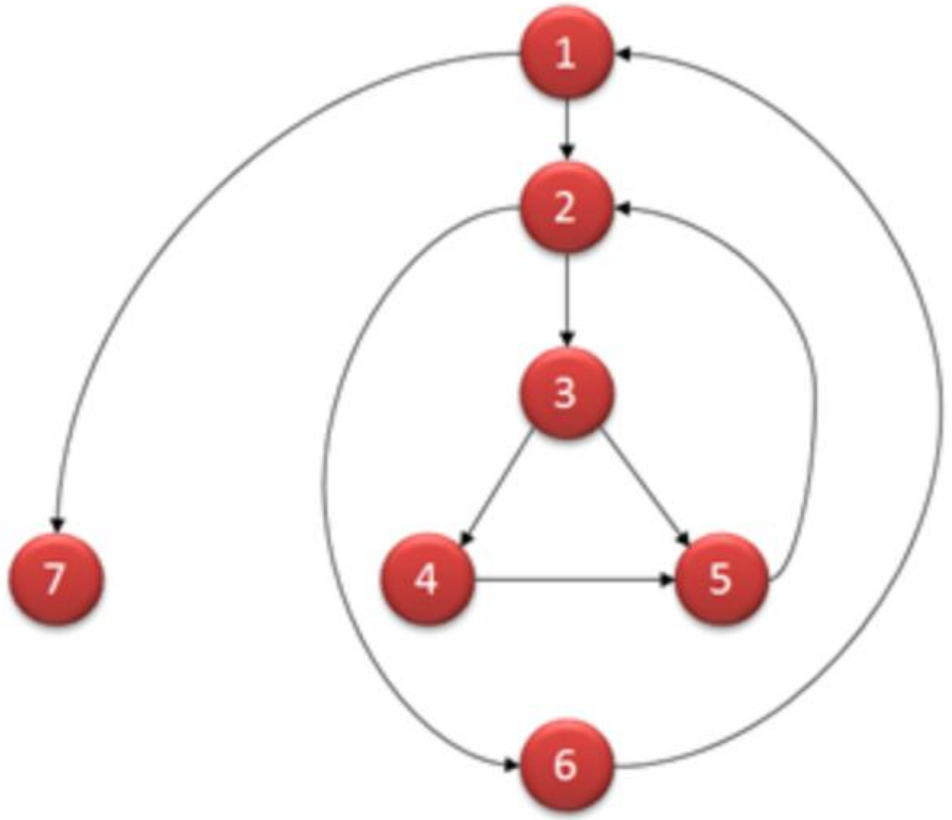
For the given code, draw the control flow graph and compute the cyclomatic complexity of the code.

Code

```
i = 0, n=4;
1> while (i < n-1) do
j = i+1
2> while (j < n) do
3> if A[i] < A[j] then
4> Swap (A[i],A[j])
5> end do
6> i = i+1
7>end do
```

## Software Engineering
## Static and Dynamic Testing

Control Flow Graph



## Computing Cyclomatic Complexity

Cyclomatic Complexity = E– N + 2*P
Where, E = Number of Edges
N = Number of Nodes
P = Number of Nodes with Exit Points

Here, E = 9, N = 7, P = 1
Cyclomatic Complexity = 9 - 7 + 2
= 4

## Dynamic Testing

Dynamic testing is when you are working with the actual system (not some artifact or model that represents the system), providing an input, receiving output, and comparing the output to the expected behavior.

It is hands-on working with the system with the intent of finding errors.

Advantage

Find difficult and complex defects

Disadvantages

Time and Budget

## Dynamic Testing

Kinds of dynamic testing

Based on:

1. Code or Fault based
2. Approach for testing
3. How testing is done
4. Levels of testing

## Dynamic Testing – code based approaches

Control Flow Based Criteria

1. Control flow path: graphical representation of all the paths that might be traversed during execution
2. Path testing: Statement, Branch and Condition testing
3. Branch Coverage
   a. Executes every path at least once
   b. No branch leads to abnormal behavior

Data Flow Based Criteria

1. Selecting paths through programs control flow in order to explore sequences of events related to status of variables or data objects
2. Statement Coverage
   a. Executes all statements at least once
   b. No side effects?

## Dynamic Testing – fault based approaches

Error Guessing

Error guessing is **a type of testing method in which prior experience in testing is used to uncover the defects in software**. It is an experience based test technique in which the tester uses his/her past experience or intuition to gauge the problematic areas of a software application.

Most plausible faults

Historical information and experience

Fault Seeding

Fault seeding is **a technique for evaluating the effectiveness of a testing process**. One or more faults are deliberately introduced into a code base, without informing the testers. The discovery of seeded faults during testing can be used to calibrate the effectiveness of the test process

Fault injected into a copy of the code

Injected and other faults detected are analyzed

## Software Engineering
## Dynamic Testing – fault based approaches

Mutation Testing

**Mutation Testing** is a type of software testing in which certain statements of the source code are changed/mutated to check if the test cases are able to find errors in source code.
The goal of Mutation Testing is ensuring the quality of test cases in terms of robustness that it should fail the mutated source code.

- Tests seldom-executed code with well-defined mutation operators
- Mutants: single statement is changed in several copies
- Test between original and mutants
- All mutants should fail
- Reveal more complex and real faults

## Dynamic Testing – based on approach used

Equivalence Partitioning and Boundary Value analysis

- ➢Uncover errors and reduce test cases
- ➢Divide input data into partitions from which test cases are derived per partition
- ➢Boundary value analysis handles edge case

A good reference is here:
https://www.youtube.com/watch?v=uydAyjqTSiw

Specification Based

***Specification Based Test Design Technique*** uses the specification of the program as the point of reference for test data selection and adequacy. A specification can be anything like a written document, collection of use cases, a set of models or a prototype.

- ➢Test functionality according to stated requirements
- ➢Test the behavior of the system based on specific input.

## Dynamic Testing – based on approach used

Intuition Based

- ➢Rely on the testers experience to design test cases for the product.

Usage Based

- ➢Finding defects that are revealed by user as they interact with the product.

Application Domain

- ➢Use specific knowledge of product domain to develop test cases.

## Dynamic Testing – based on mode of testing

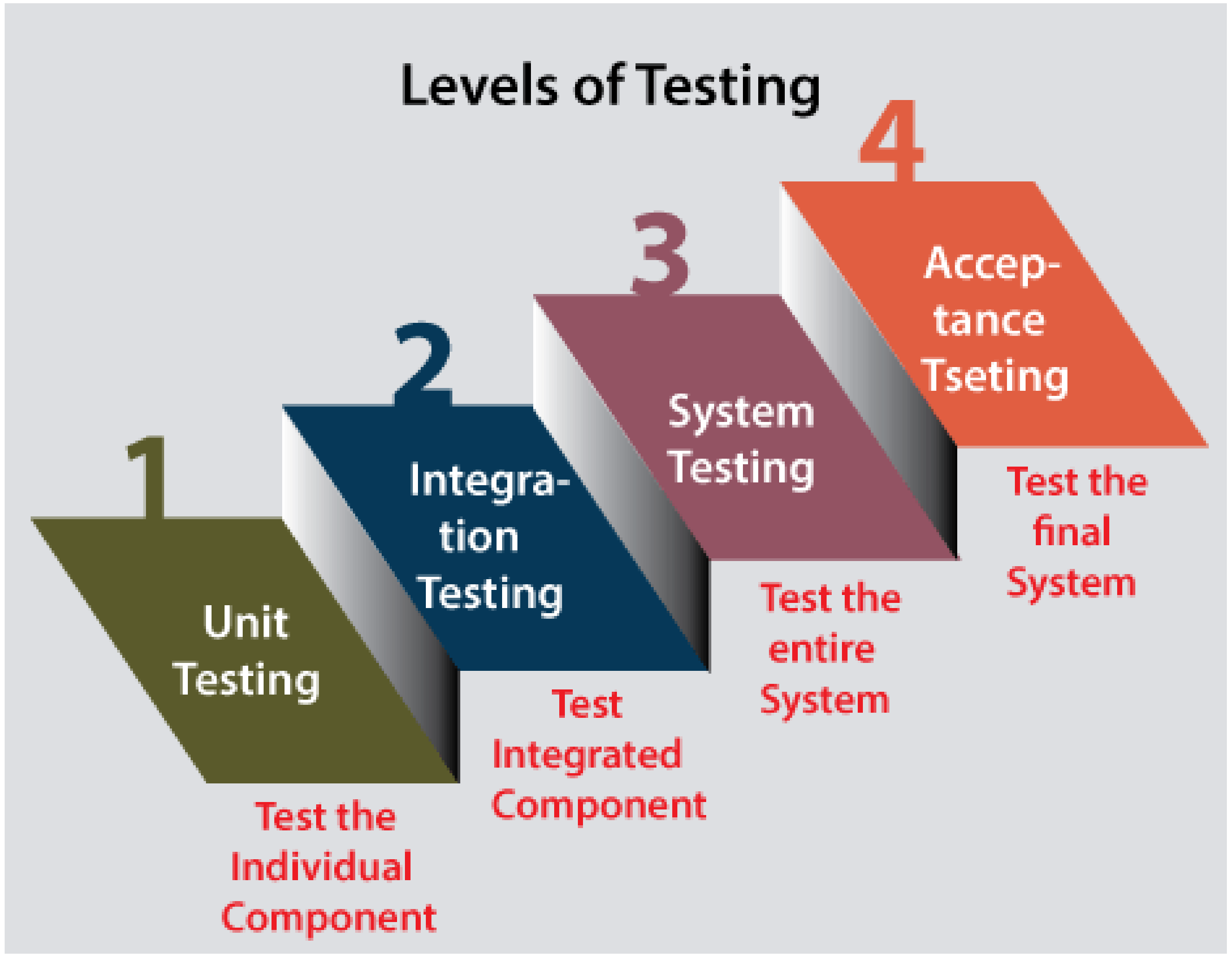
Manual Testing

➢Humans perform the test step by step, without test scripts
➢Used in complex testing where automation is very expensive
➢Slow and tedious
➢Hard to get test coverage
➢Examples: Acceptance testing, Black Box testing, White Box testing, Unit tests

Automated Testing

➢Involves tests which are executed without human assistance
➢Needs coding, test framework maintenance
➢Fast and repeatable
➢Most efficient for periodic and regular tests

## Software Engineering
## Levels of testing



## Levels of Testing

Unit Testing
Verifies proper functioning of the individual unit.

Integration testing
Focuses on finding interface errors between components
Bugs that are not identified during unit testing

System Testing
Assessment of the entire system behavior
Information helps direct product release
Discover bugs that cannot be attributed to single component

Acceptance Testing
Done by system providers/users/customers
Determines if system meets the needs

## Unit Testing

**Focus**: Test for coding/construction errors prior to quality engineering

Tests the smallest individually executable code units.

Done by programmers

Test cases for

- ➢Algorithms and logic
- ➢Data Structures
- ➢Interfaces
- ➢Independent paths
- ➢Boundary conditions
- ➢Error Handling

In OO environment, tests are at class level using constructors and destructors.

## Integration Testing

**Focus**: Test to verify the interface between components against a software design

Performed by programmers to isolate timing and resource contention problems

Approach

Abstract away unit issues and look for defects between units

**Strategies**

Big bang approach

Integrate everything and run to see if system functions as expected

Iterative approach

Software components are integrated iteratively until software works

Top down or Bottoms up

## System Testing

System testing tests a completely integrated system to verify compliance with Software Requirement Specifications

Detect defects both within “inter-assemblages” and within the system as a whole

Black box testing which involves testing end-to-end flow of an application

<u>Process</u>

- ➢Test Environment Setup
- ➢Create Test Case
- ➢Create Test Data
- ➢Execute Test Case
- ➢Defect Reporting and Logging

## System Testing – Types

Smoke and Sanity testing

- Ensure that the most important functions work
- Decide whether the build is fit for further testing

Software Performance testing

- To test responsiveness, scalability, stability and reliability of the software product

Functional and Non-functional testing

- Functional Testing: test features and functionality covering all scenarios including failure and boundary cases
- Non Functional Testing: Verify the attributes of the system such as performance or robustness

## System Testing – Types

<u>Destructive testing</u>

- ➢Application is made to fail in an uncontrolled manner to test robustness of the application and find point of failure

<u>Installation testing</u>

- ➢Verify if the software has been installed with all necessary components and application works as expected
- ➢Verify if all the components of the application have been removed during the process

<u>Usability testing</u>

- ➢Ease of use of the system from user's perspective
- ➢Level of skill required to learn/use the software and time to acquire the skill

## System Testing – Types

Localization testing

- Verify the quality of products localization for a particular target culture

Compliance testing

- Compliance with internal and external standards.

Scalability testing

- Determine capability to scale up or scale down

Boundary tests

- Black Box testing using boundary values

Regression testing

- Determine whether any changes have caused unintended side effects

## System Testing – Types …

Startup and Shutdown test

- ➢Shutdown testing ensures system has not left uncleared lock files, states of tables or inconsistent data
- ➢Startup ensures product starts in deterministic and consistent state

Platform tests

- ➢Cross platform tests evaluate behavior of application in different environments

Load tests

- ➢Determine behavior or robustness of system under varying load

## System Testing – Types …

Stress Test

- ➢System is pushed to maximum design load and beyond to test limits of the system

Security testing

- ➢Testing to uncover vulnerabilities and ensure data and resources are protected

Recovery tests

- ➢Testing to check restart capabilities following a disaster or unanticipated shutdown

Cloud testing

- ➢Software application is tested using cloud computing services
- ➢**Question**: Could performing cloud testing result in skipping few of the above tests?

## Acceptance Testing

Acceptance testing involves running a suite of tests on the completed system
Each test case exercises a particular operating condition of the users environment or feature

Tests are created through collaboration between customers, business analysts, testers and developers

Verifies the completeness of a user story during a sprint

Provides confidence that the delivered system meets the business requirements of sponsors and users

- Final quality gateway
- Question: What happens if there are undetected errors beyond this point?

## Software Engineering

## Acceptance Testing categorization

| | |
|---|---|
| **Acceptance / qualification testing** | Checks the system behavior against the customer's requirements, however these may have been expressed. |
| **Installation testing** | Verifies the installation in the target environment. May be identical to system testing in a new environment. |
| **Alpha and beta testing** | Before the software is released, it is sometimes given to a small, representative set of potential users for trial use, either in-house (*alpha* testing) or external (*beta* testing). … Alpha and beta use is often uncontrolled, and is not always referred to in a test plan. |
| **Performance testing** | Aimed at verifying that the software meets the specified performance requirements (capacity and response time, for instance). A specific kind of performance testing is volume testing, in which internal program or system limitations are tried. |

## Software Engineering
## Test planning generics

What is Software Test Planning?

Software test planning is the process of evolving a test plan which discusses what, when, how much and how testing has to be done to ensure quality expectations can be met

- The outcome also serves as a blueprint to conduct software testing activities as a defined process
- Developers, business managers and customers can understand the details of testing
- This plan is also to be used for monitoring and control.

## Test planning process overview

1. Ensuring Context and Scope of the project is understood
2. Establish test adequacy criteria
3. Evolve a test strategy which will be followed
4. Evolving list of deliverables
5. Creation of detailed test schedule
6. Planning, Identification and Allocation of resources
7. Identification of milestones
8. Risk Management
9. Establishment of measures and metrics

## Understanding and Determining the product testing scope

Understanding the context where the product is going to be used by

- ➢Reviewing use case scenario in the product deployment environment
- ➢Discussing with the designer
- ➢Discussing with the developer
- ➢Reviewing project/product documentation
- ➢Play around the product or perform product walk-through

Optimal amount of testing is based on

- ✓Customer requirement
- ✓Project schedule
- ✓Project budget
- ✓Product specification
- ✓Skills and talent of test team

## Test adequacy

Testing of the subset of possible combinations does not guarantee absence of issues

Fixing issues found during testing may not be feasible due to

- Volume or number of errors
- Schedule
- Resources
- Some issues are enhancements

Test Adequacy Criteria: Criteria to determine when to stop testing or consider testing complete for that iteration

- Examples
  - ❑Of the planned tests, x% of lines are executed and y% of branches
  - ❑All planned test cases are complete with no critically high priority issues
  - ❑Total number of severe defects is less than 5

## Testing strategy

Known as the test approach and defines how testing is carried out and deals with following

- Testing mindset or test models to be followed
- Test types which will be used
- Test environment
- Automation Strategy
- Tools
- Risk analysis with contingency planning

## Testing Strategy – Models/Mindsets

Demonstration

- ▪Make sure software runs and solves the problem
- ▪If software passes all tests, establishes satisfaction of specs
- ▪Might only test what succeeds

Preventive

- •Prevents faults in early phases through careful planning and design
- •Reviews and Test Driven Development

Destruction

- •Try and make the software fail to find as many faults
- •Good and effective test cases find faults
- •Difficult to decide when to stop testing

## Testing Strategy – Models/Mindsets ..

Evaluation

- Detects faults through the lifecycle phases
- Focuses on analysis and review techniques to detect faults in requirements and design documents

## Testing Strategy – Testing Types Chosen

**Each lifecycle phase has outcomes which can be tested**

- ➢Feasibility phase has acceptance test cases
- ➢Requirements phase has requirements specification
- ➢May have functional and system test cases
- ➢Architecture/Design phase has refined functional and system cases and integration test cases
- ➢Implementation phase has code unit test cases
- ➢Testing phases would involve reviews and execution of test cases
- ➢Maintenance phase would involve review and execution of regression and other tests

**Strategies in terms of**

- ➢Begin by “testing-in-the-small” and move towards “testing-in-the-large”
- ➢Top Down and Bottoms up
- ➢Positive and Negative testing
- ➢Dynamic and Heuristics based approach

## Testing Strategy – Test Execution Environment

Testing Environment/ Test Execution Environment/ Test Bed: setup of software and hardware for testing teams to execute

- Configured as per need of the application
- Could include test data, Database, front end, Operating System, Servers, Storage, and Network

Correct setup ensures success of testing or results in delay, cost escalations and so on

Environment management involves maintenance and upkeep of test bed, monitoring and modifying components

Challenges in terms of planning resource usage, remote environment, setup time, sharing of environment and setup of complex configurations.

## Testing Strategy – Automation

- Defining goals
- Planning the test approach
- Selection of Automation framework
- Selecting Test Tool
- Test Case Design and Execution
- Test Result Generation and Analysis
- Maintaining script

## Testing Strategy – Tools which will be used

- ❖The technology of the Software or application under test (AUT) will need to be compatible to the tools and drives the selection of it.
- ❖Testers will need to be comfortable with the tool, else it may not be effectively used
- ❖ Tools chosen need to be balanced in terms of the features offered, ability to generate reports/data needed for different stakeholders and the ease of use.
- ❖Cross platform support is an expectation as automated tests would/may need to run on different platforms.
- ❖Acceptability/Popularity/prevalence of the tool in the Industry is an indication of availability of support, quality documentation, technical forums and availability of trained personnel.
- ❖Cost
- ❖Opensource or Proprietary have different characteristics of cost, features and the support and needs to be balanced

## Testing Strategy – Risk Analysis with Contingency Planning

Risk: Probability of an unwanted incident during or towards testing

- Risks in strategy could be in terms of
    - Changes to the Business, Technology or competition directions
    - Resources
    - Quality of the software product being developed
    - The test models not being able to be used
    - Some type of testing chosen cannot be used
    - Test environment and its state
    - Automation or Tool issues

Any risks found in any of them would need to be planned for addressing as part of the mitigation and contingency.

## Test planning

<u>Evolving a list of deliverables</u>

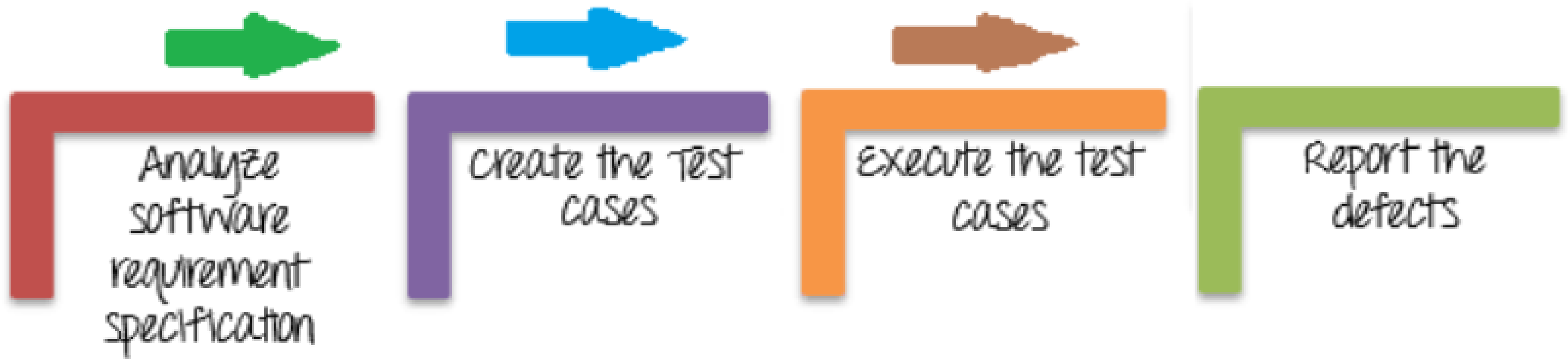
List of Activities and Deliverables will be identified which would need to be executed and delivered

Could include

- ➢Test specifications for each of the modules of the product
- ➢Test cases for different conditions planned

## Creation of Detailed Test Schedule

- Includes estimates for building test strategy/specification/test cases/test environment setup and test execution, test reporting, etc.
- Would include Work Breakdown Schedule and estimation using methods mentioned in project planning
- Calendarize the Work Breakdown estimates



## Planning, Identification and Allocation of Resources

The current step and the previous step of estimation are performed together or iteratively

- This ensures the schedule factors in characteristics of the planned resources

Initially involves

- Identifying the number and type of servers, storage, test tools and network resources
- Identifying the number and type of people to work on the project
    - Test Manager, Testers, Test Developers, Test Administrators, and SQA
- Identifying the test environment

Schedule is reworked after resource and skill identification

## Identification of the Milestones & Risks

Considering the expected deliverables, schedule and commitments, project milestones are identified

- Milestones track or monitor progress and control overruns
- Used to identify any risk triggers

Risks for completion of a task from schedule and quality perspective is identified and analyzed

- Mitigation plans and triggers for kick-off are identified

## Identify Measures and Metrics

Measurements are identified

- Examples: Number of Test Cases planned and Created, Number of test cases run, amount of time spent on creation, execution, number of errors found

The errors are classified into critical, serious, medium or low impact

Metrics

- Examples: Number of test cases executed/day, % of test cases executed, number of issues/KLoC, number of critical issues/KLoC

## Typical Test Plan contents

1. Introduction
    1. Scope
        1. In Scope
        2. Out of Scope
    2. Quality Objective
    3. Roles and Responsibilities
2. Test Methodology
    1. Overview
    2. Test Levels
    3. Bug Triage
    4. Suspension Criteria and Resumption Requirements
    5. Test Completeness
3. Test Deliverables

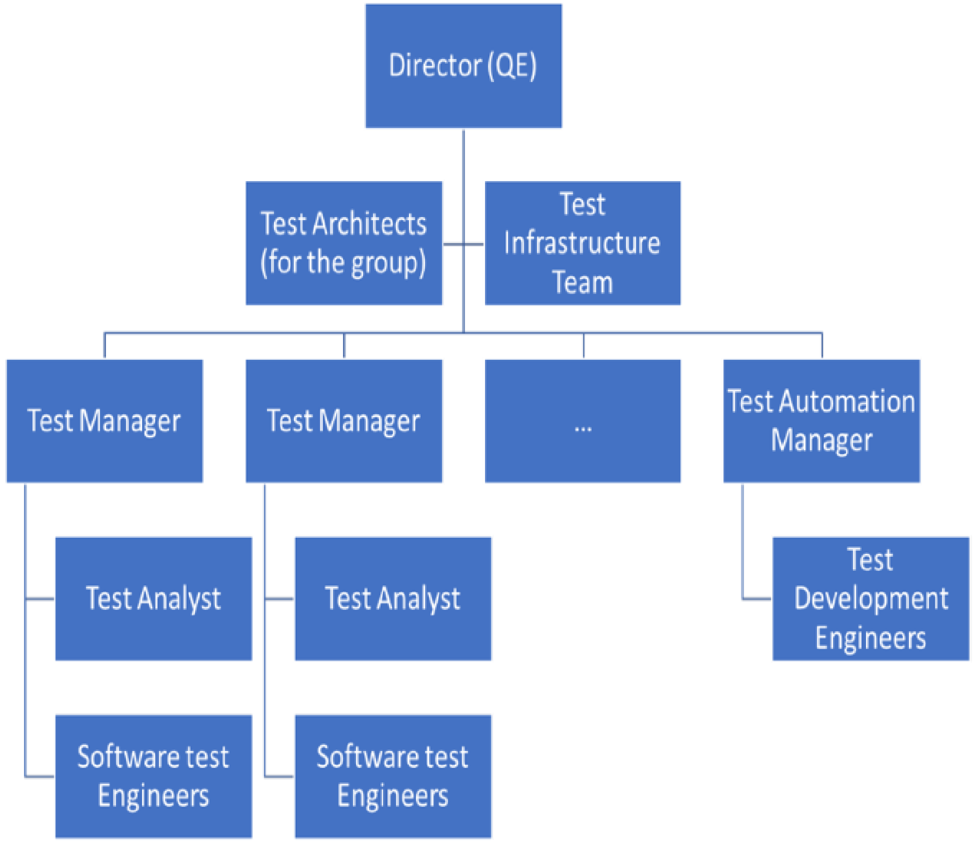
4. Resource & Environment Needs
    1. Testing Tools
    2. Test Environment

## Test plan document

A sample software test plan is here.

## Software Engineering
## Test organization

Software Testing is considered as a part of Quality Engineering
QE groups are typically organized as shown below (some roles may/may not exist)



## Roles and Responsibilities

**Test Director**
Provides oversight, co-ordination, strategic vision, customer and stake holder connect.

**Test Manager**
Prepare test strategy, plans for project, monitors and controls testing.

**Test Infrastructure Manager**
Manages all the infrastructure, capacity planning, maintenance, support and configurations.

**Test Automation Manager**
Manages the development of tools and scripts for automating.

**Test Architect**
Designs test infrastructure, picks appropriate tools, validates test strategy.

## Roles and Responsibilities …

**Test Analyst**
Mapping of customer environment and test conditions and features to testing conditions and documentation.

**Software Test Engineer**
Tests the product using appropriates testing techniques and tools

## Test process

**Planning and Control**

1. Involves identifying requirements
2. Analyze test requirements, product architecture and Interfaces
3. Creating Test Sufficiency criteria
4. Creating Test Strategy, Planning for resources and building a schedule
5. Setup review points, status reporting mechanisms, approval boards

**Design**

1. Identify test conditions, design tests and test environment

**Implementation and Execution**

1. Develop test cases, test data, test automation
2. Execute tests, collect metrics, log results and compare with expected values
3. Track Defects, Bug Fixing

## Test process …

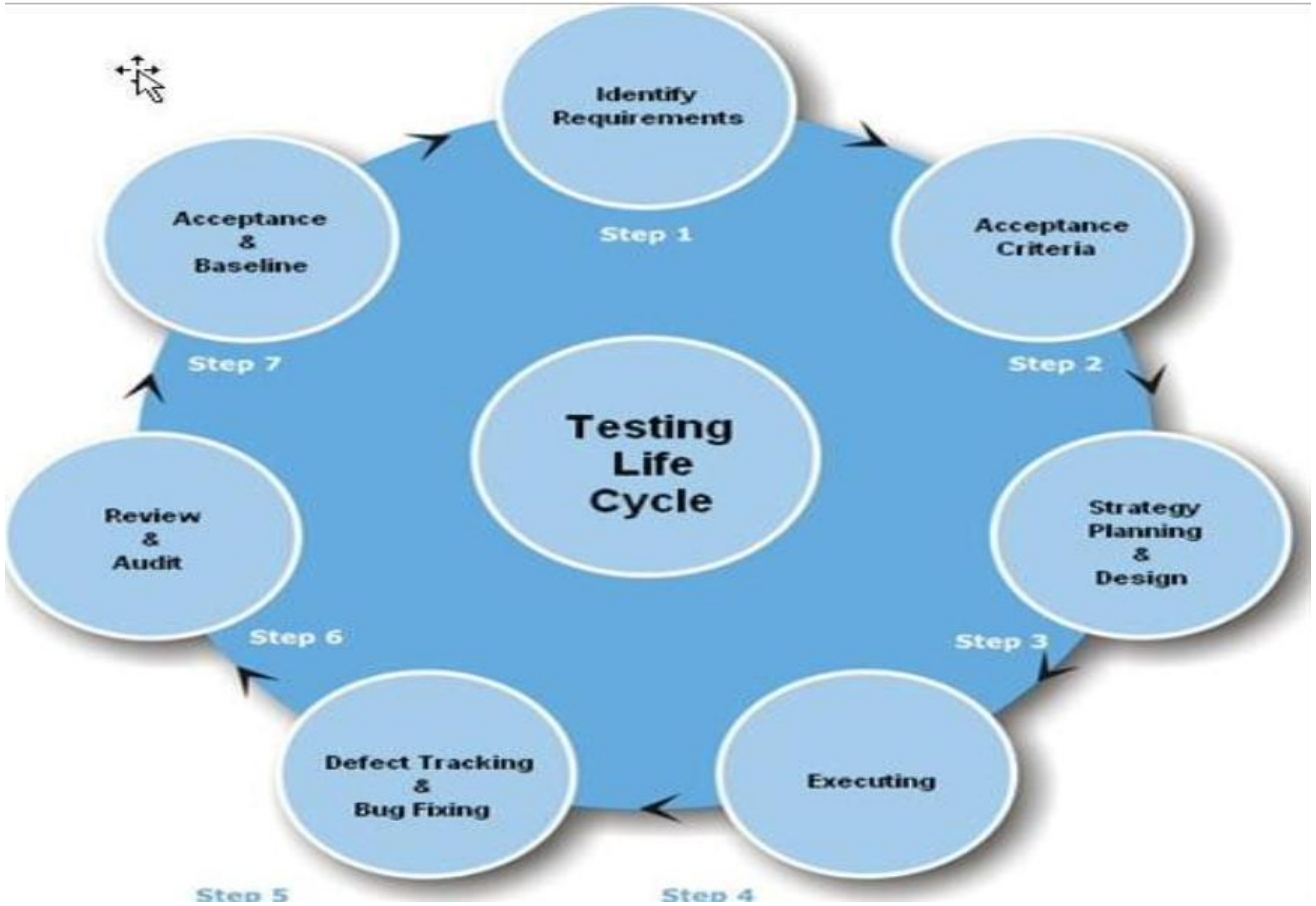
**Evaluate exit criteria and Reporting**

1. Evaluate test completion/stopping criteria based on application functionality
2. Eg: Test Cases, Pass Percentage, Bug rate, Deadlines, RTM

**Test Closure Activities**

1. When testing is complete or project is cancelled
2. Verify all planned deliverables
3. Archive test scripts, environment and close with reports
4. Perform a retrospective

## Software Engineering
### Test process …



## Software Test execution

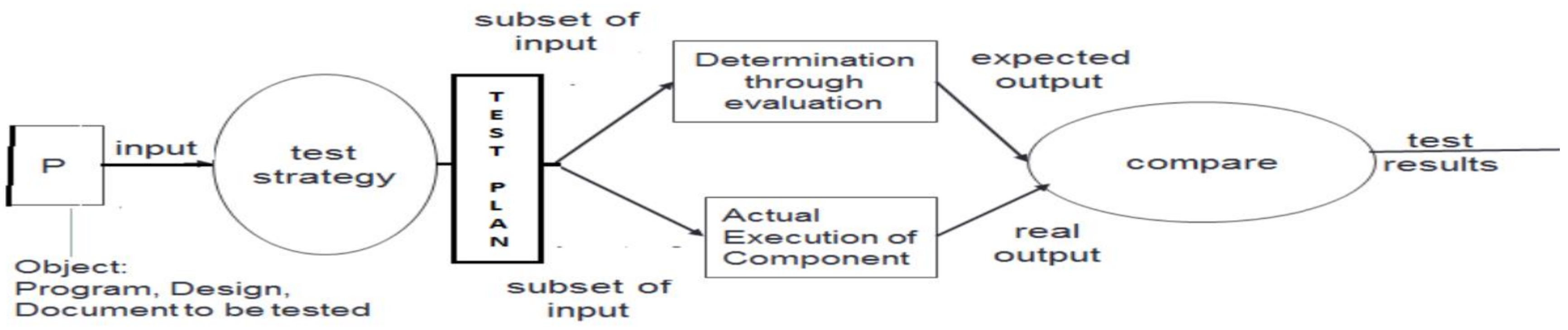
Process of executing code and comparing actual observation with predicted expected values

<u>Steps</u>

- Subset of test cases is selected for execution
    - Test cases are assigned to testers for execution
- Environment is setup-configured-test data is setup
- Steps for execution is noted and expected output is looked at
- Tests are executed, results logged, status captured and bugs logged
- If a road blocks occurs, it is resolved before continuing
- Results reported, measurements done, metrics and test results analyzed

## Software Engineering

## Software Test execution


subset of input
Determination through evaluation
expected output
P
input
test strategy
TEST PLAN
compare
test results
Actual Execution of Component
real output
Object: Program, Design, Document to be tested
subset of input

## Software Engineering
## Test cases

- Test cases are a key part of test documentation
- Tells the tester what to test, how to check it, and what the expected results are
- Once test is executed, determines if software satisfies requirement or function properly
- Act as the starting point for text execution
- Well written test cases enable anyone to step into tester role and determine functioning of software
- Ensure proper test coverage of application under test
- Writing test cases helps in thinking through the details and ensures testing is addressed from multiple angles

## Test cases

### Typical Test Case Parameters are

- Test Case ID
- Test Scenario
- Test Case Description
- Test Setup
- Test Steps
- Prerequisite if any
- Test Parameters
- Expected Result
- Actual Result
- Comments

**Sample**
**Test Case Id:** TC1
**Title**: Login Page – Authenticate Successfully on gmail.com
**Description:** A registered user should be able to successfully login at gmail.com.
**Precondition:** the user must already be registered with an email address and password.
**Assumption:** a supported browser is being used.
**Test Steps:**
1.Navigate to gmail.com
2.In the 'email' field, enter the email address of the registered user.
3.Click the 'Next' button.
4.Enter the password of the registered user
5.Click 'Sign In'
**Expected Result:** A page displaying the gmail user's inbox should load, showing any new message at the top of the page.

## Value proposition of writing test cases

Test Scenarios identify and isolate areas to be tested

- Broken down into detailed subsection and creates a frameworks for comprehensive and successful testing

Beneficial to future teammates

Good source of truth on how system and particular feature works

Provide following value

- Ensures good test coverage
- Allows tester to find different ways of validating feature
- Negative test cases are also documented
- Reusable for the future

## Types of test cases

Different types of test cases ensure assurance of quality
Work with a subset of tests that achieve the highest quality products

**Types**
**Positive**
Verify the software is doing what it’s supposed to do .
**Negative**
Verify the software is not doing what it is not supposed to do.
**Destructive**
Scenarios the software can handle before it breaks.

Example
Allow login when valid credentials are supplied:
Don’t allow login when invalid credentials are supplied

## Writing test cases

Keep the title short

Include a strong description

Be clear and concise

Include the Expected result

Make test case reusable

## Test suites

Test Suite is a container that has a set of tests which helps testers in executing and reporting the test execution status

Has three states

- ➢Active
- ➢In-Progress
- ➢Completed

Test case can be added to multiple test suites and test plans

Test suites are created based on the cycle or scope

Can contain any type of tests

- ➢Functional
- ➢Non Functional

## Software Test Related Measurements and Metrics

Software Test Metric: Quantitative Measure of testing process indicating progress, quality, productivity and degree to which system possesses a given attribute

Goal

- Improve efficiency and effectiveness of software testing process
- Make better decisions for future testing process using reliable data

**Measurements**: critical in performing SQA and testing

Support increasing effectiveness of testing process

- ➢When used as part of project planning and monitoring
- ➢Evaluate the quality of product under test
- ➢Helps in defect analysis

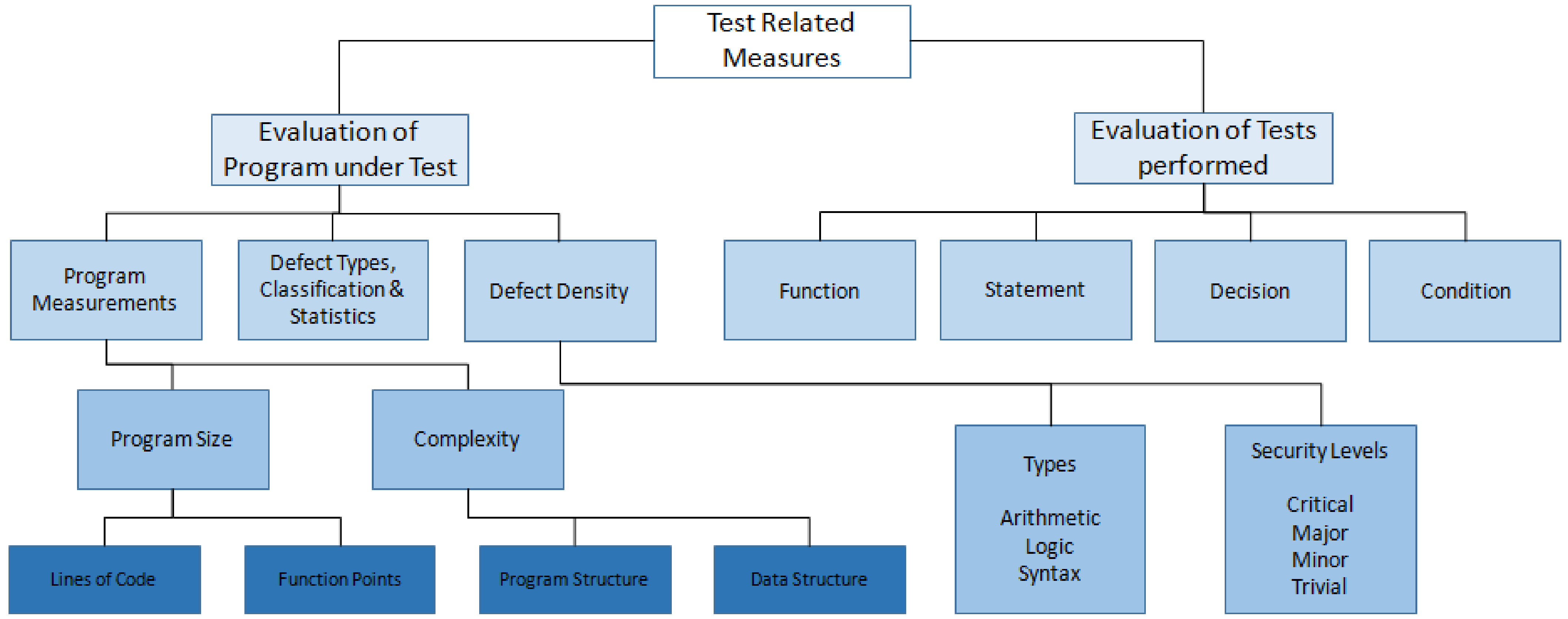
Measures can

- Represent the quality
- Represent test coverage

## Software Test Related Metric Characteristics

**Quantitative**
Expressed in Values
**Understandable**
Method of metric computation should be easily understood and clearly defined
**Applicability**
Should be applicable in initial phase of development
**Repeatable**
The results should be consistent when the same measures are used.
**Economical**
Computation should be economical
**Language Independent**
Should not depend on programming language

## Software Engineering
## Software Test Measurements



## Software Test Related Measures – Product Related

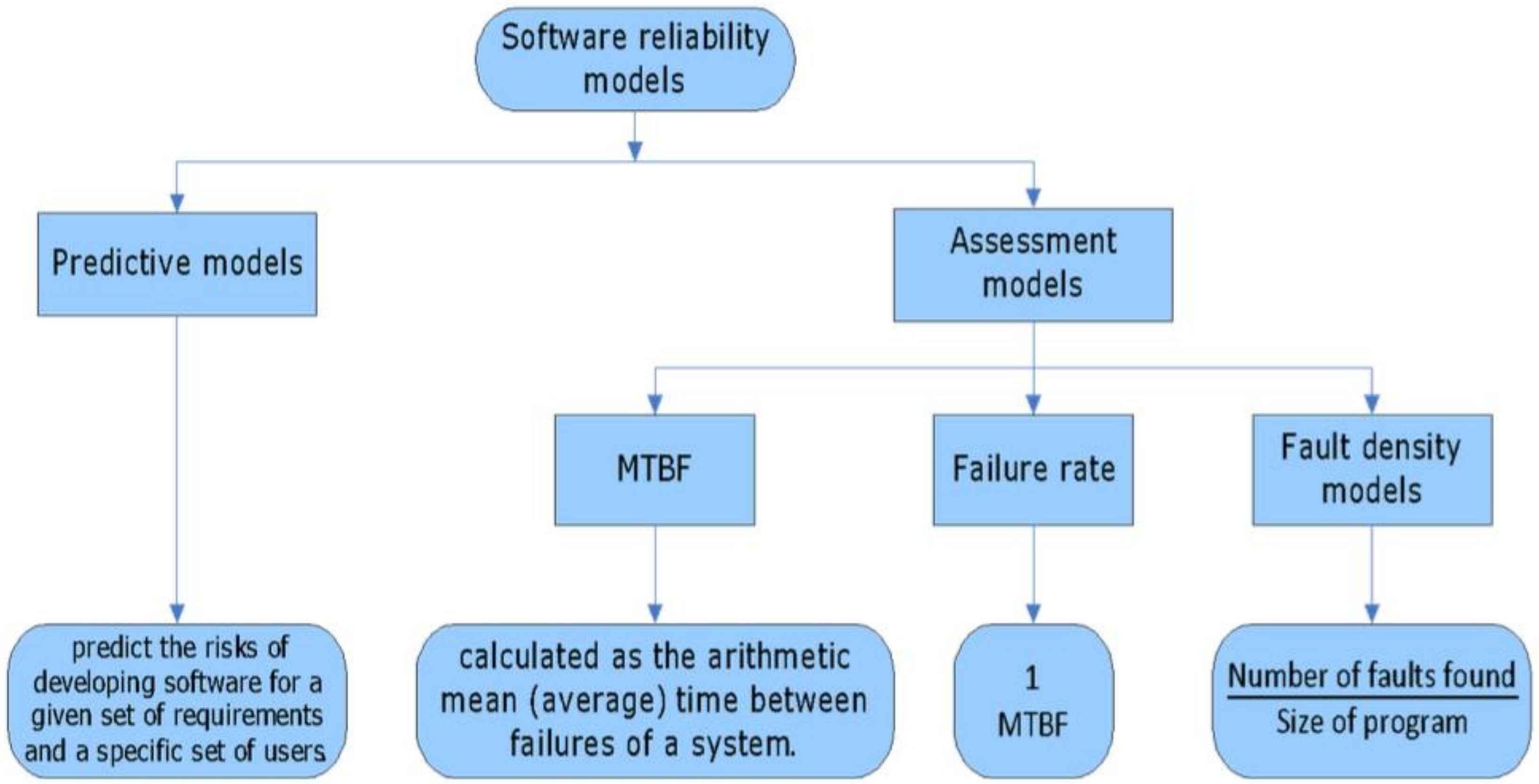
| Metrics | Description |
|---|---|
| SLOC | Size in Lines of Code |
| Fault Density | Ratio of number of faults found to size of the programs |
| MTBF | Mean Time Between Failure is based on statistical analysis to indicate probability of failure |
| Failure Rate | Inverse of MTBF |
| Defect Distribution | % of defects attributed to a specified phase in SDLC |
| Defect Density of Modules | Ratio of number of faults found in module to total faults found in product |
| Defect Leakage | Test Efficiency = [(Total number of defects found in UAT)/(Total number of defects found before UAT )] x 100 |

## Software Test Coverage Metrics – Product Related

| Metrics | Description |
|---|---|
| Percent Test Coverage | Includes: Statement Coverage, Branch/Decision Coverage, Condition Coverage, Loop Coverage, Path Coverage, Data Flow Coverage, etc. |
| Percent completed and<br>Percent Defects corrected | # of test cases executed and closed compared to total # of test cases<br>& # of defects fixed/closed compared to total # of defects |
| Defect Removal Effectiveness | % of defects found at the later phase of development. |
| Requirement Compliance Factor | Using the traceability matrix, the RCF measures the coverage provided by a test case to one or set of requirements. |
| Defect Discovery Rate | Number of defects found per line of code (LOC). |
| Defect Removal Cost | The cost associated with finding and fixing a defect. |
| Cost of Quality | Total cost of prevention, appraisal, rework/failure, to the total cost of the project. |
| Percent Injected Fault Discovered | Number of defects injected found, compared to total number of faults injected. |
| Mutation Score | Number of killed mutants to the total number of mutants. Again, any mutant left alive points to the weakness in the testing effort. |

## Software Reliability

**Software Reliability** is the probability of failure-free software operation for a specified period of time in a specified environment


Software reliability models
Predictive models
Assessment models
MTBF
Failure rate
Fault density models
predict the risks of developing software for a given set of requirements and a specific set of users
calculated as the arithmetic mean (average) time between failures of a system.
1 / MTBF
Number of faults found / Size of program

## Test driven development

Prevention Model

•Write the tests first, then do the design/implementation

•Could be done as part of some agile approaches like XP

❑Supported by tools, e.g. Junit

Steps Associated

1. Add a test
2. Run this and earlier tests, see if the system fails
3. Make a small change to make the test work
4. Continue incrementally until all planned tests run properly
5. Refactor system to improve design and reduce redundancies

## Software Engineering
## Key issues in Software Testing

- Test Selection
- Objectives for a Test
- Testing for Defect Identification
- Theoretical and Practical Limitations
- Infeasible Paths
- Testability

## Key Issues in Software Testing – Test Case Selection

Testers need to choose the most important tests to run.

**Common ways to prioritize tests**

- Test highest priority requirements first
- Test complex code first
  - On basis of McCabe’s Cyclomatic complexity
- Test Largest modules first
- Test the most often modified modules first

## Maintenance - Introduction

Software delivered to customers will and can go through a large number of modifications and enhancements due to

1. Fixing Defects reported from the field
2. Correcting requirement and design errors
3. Implementing enhancements
4. Improving performance
5. Adapting the system in response to the environment changes such as
   a. Hardware changes
   b. Software changes within programs that it interfaces with
   c. Network/Telecommunication changes
   d. Changes in standards
6. Retiring outdated systems
7. Interfacing with other software

## Maintenance - Introduction

Maintenance: Sustaining process of modifying a software system or component after delivery, to correct faults, improve performance or other attributes, or adapt to changed environment

- ➢Totality of activities required to provide cost-effective support to software
- ➢Modification of software product after delivery to correct faults, improve performance or other attributes, or to adapt to modified environment
- ➢Largest portion of the total life cycle as products are used for a long period of time

**Why**

To ensure software product continues to satisfy customer needs
Commitment to support system customers have invested in

**Who**

Maintainer could be an organization or person responsible for carrying out maintenance activities

## Maintenance - Activities expected and Actions which can help

Activities expected

1. Maintaining control over software's day to day functioning
2. Maintaining control over software modifications
3. Perfecting existing functions
4. Preventing software performance degrading to unacceptable levels

Actions which will support the activities

1. Understand the product which needs to be maintained by studying architecture, design, code, test cases and documentation
2. Discussion with architects and developers
3. Adequate and precise documentation, testable code, code management, instrumentation and debug code
4. Look at present and change request at hand to identify how to satisfy task
5. Look at future and analyze consequences of solution

## Maintenance - Costs

Maintenance consumes a major share of software lifecycle financial resources

- ➢Typically 70% of total effort spent would be on maintenance
- ➢Total costs also follows this trend

Factors affecting maintenance costs

- ➢Application type
- ➢Software novelty
- ➢Software maintenance staff availability
- ➢Software lifespan
- ➢Hardware Characteristics
- ➢Quality of Software Design, Construction, Documentation, and Testing

## Software Engineering
### Maintenance – Key issues

Technical

1. Code and Documentation Quality
2. Limited Understanding
3. Availability of test environment to reproduce problems
4. Impact analysis

Management

1. Staffing and retaining people with right skill levels
2. Alignment with economic objectives as maintenance does not have much RoI
3. Outsourcing issues
4. Protection of Intellectual Property
5. Control over development process and quality control
6. Learning curve for product
7. Scope of Maintenance

## Software Engineering

### Maintenance - Costs

Cost

1. Application type
2. Lifespan of the system
3. Quality of software artifacts
4. Staff availability

Predictability

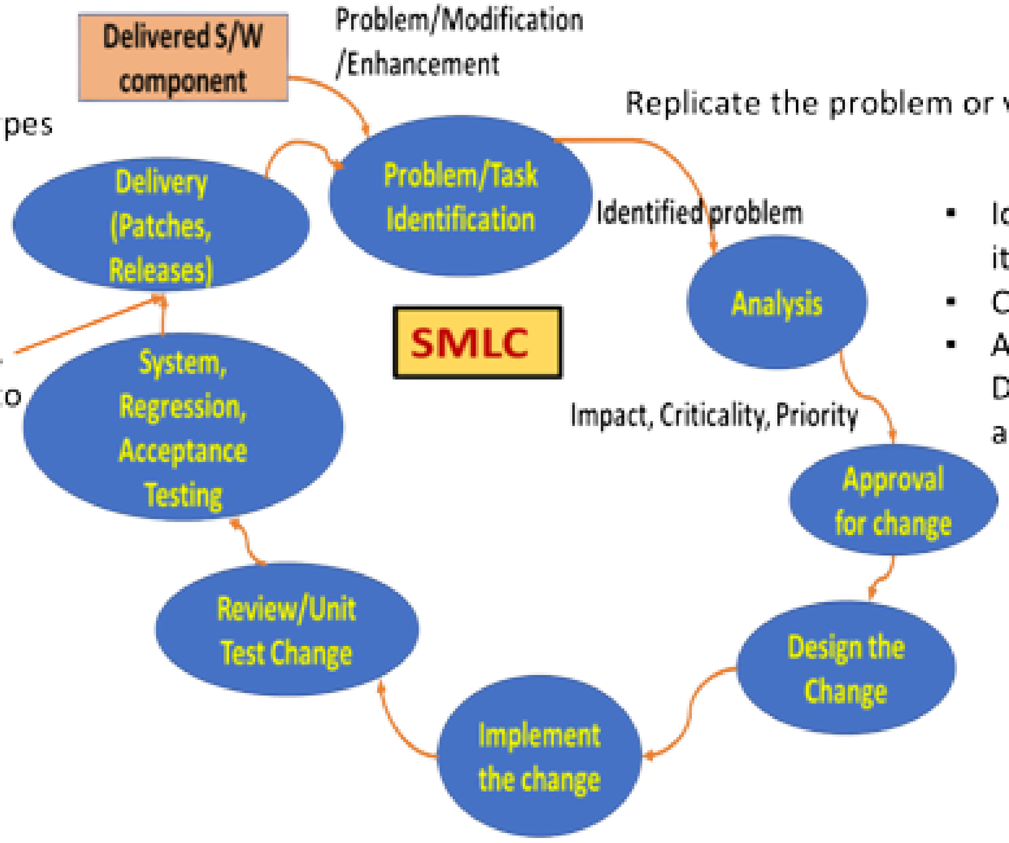
1. Measures related to schedule, time required for maintenance, Complexity and Quality of product

# Software Engineering

## Maintenance – Life Cycle

Different delivery mechanisms with Maintenance lifecycle
Patches – Different Types
Complete products
Application release bundles

Check in the approved, changed components to the CM system


Delivered S/W component
Problem/Modification /Enhancement
Problem/Task Identification
Identified problem
Analysis
Impact, Criticality, Priority
Approval for change
Design the Change
Implement the change
Review/Unit Test Change
System, Regression, Acceptance Testing
Delivery (Patches, Releases)
SMLC

Replicate the problem or verify its existence

- Identify all affected configuration items
- Consider modification options
- Analyse the impact of problem

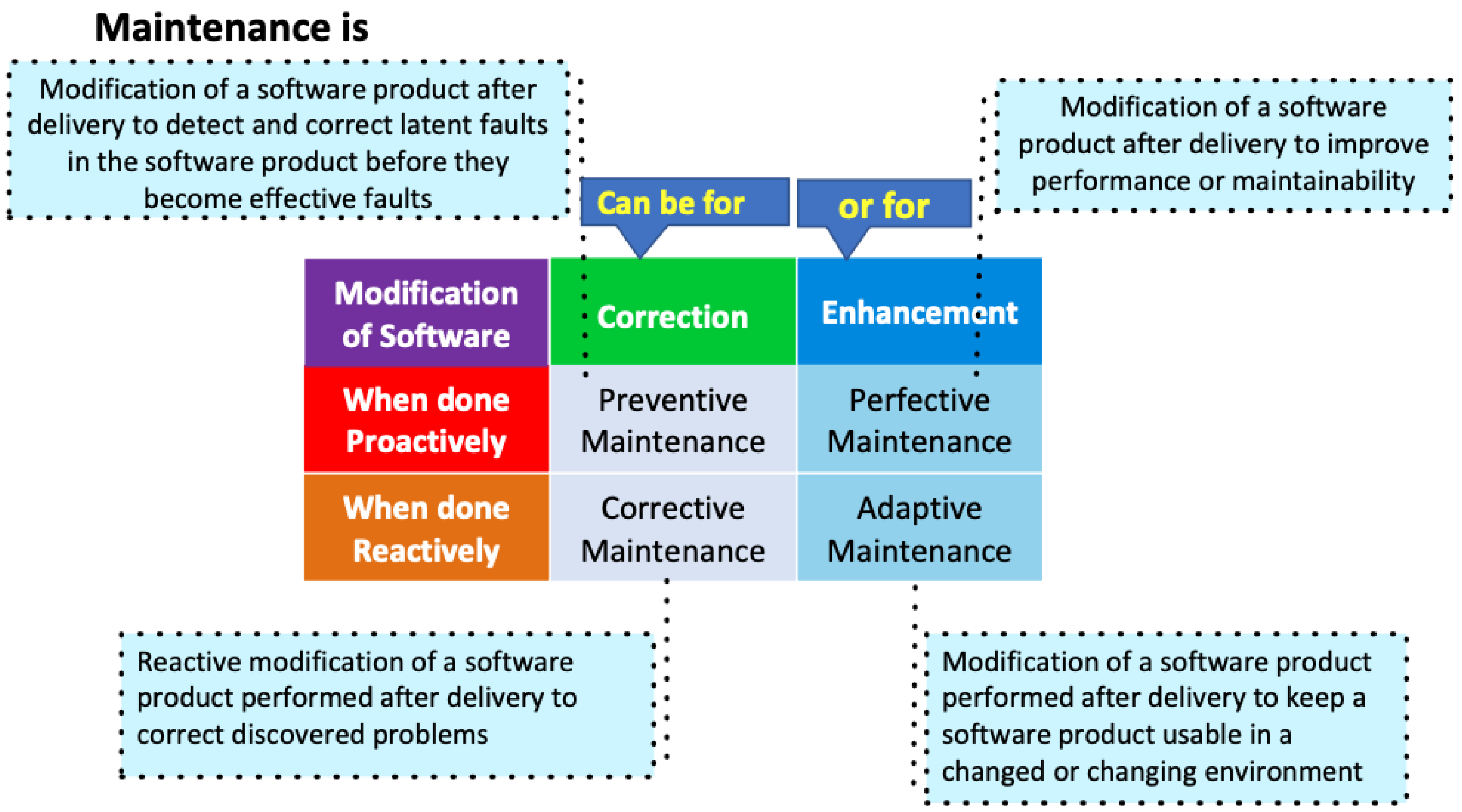
Document the request, the analysis, and the suggested options

# Software Engineering

## Maintenance - Tips

- Reproduce the problem.... Before you start fixing it
- Understand the stack traces
- Check your environment variables
- Write a test case that reproduces the problem .. Even better ..Convert the problem into an automated test. ...
- Know your error codes
- Don't assume things work the way they're meant to. ...
- Be clear in your mind about correct behavior. ...
- Pair program whenever possible .. Two pairs of eyes helps
- Guess and check
- Get your code to help you. ...
  - Print
  - Use logs
  - Comment out code and check
- Learn your debugger's capabilities
- Take breaks to get your mind off and restart
- Fix one problem at a time. ...
- Test, Test and Test
- Consider the bigger picture

## Categories of maintenance



**Maintenance is**

Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults

Modification of a software product after delivery to improve performance or maintainability

| Modification of Software | Correction | Enhancement |
|---|---|---|
| When done Proactively | Preventive Maintenance | Perfective Maintenance |
| When done Reactively | Corrective Maintenance | Adaptive Maintenance |

Reactive modification of a software product performed after delivery to correct discovered problems

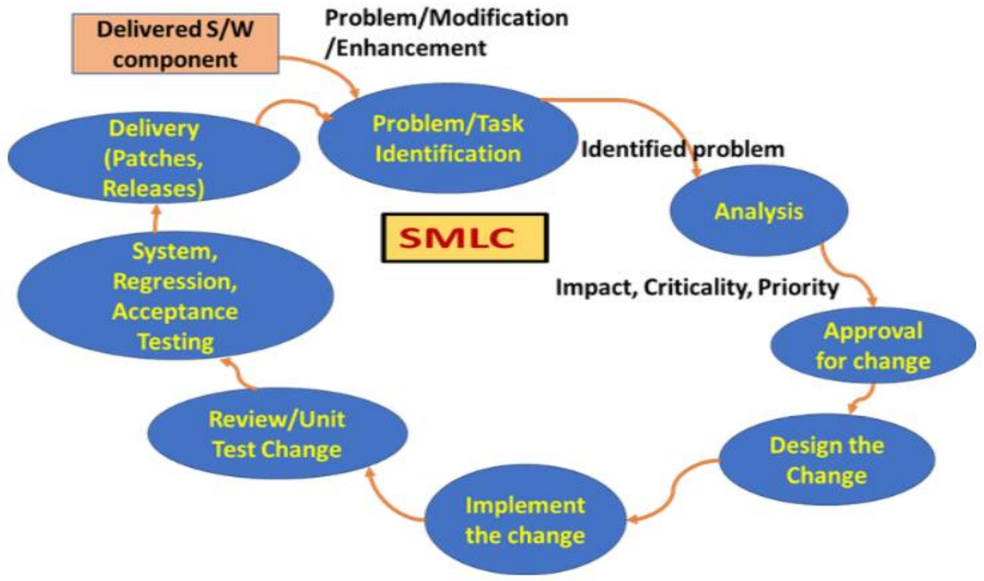
Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment

## Software Engineering
## Maintenance process activities

Implementation of Processes

1. Develop, Document and execute maintenance process plans including usage of tools as part of the process
2. Procedures for handling user reports and modification requests, tracking of problems, responding to users

Follow the Software Maintenance Life Cycle



Delivered S/W component
Problem/Modification /Enhancement
Problem/Task Identification
Identified problem
Analysis
Impact, Criticality, Priority
Approval for change
Design the Change
Implement the change
Review/Unit Test Change
System, Regression, Acceptance Testing
Delivery (Patches, Releases)
SMLC

## Maintenance process activities

Migration

1. Develop a migration plan for the fixed/new component and execute the plan
2. This could be
3. Notifying and replacing the product and restarting
4. Applying the path with restart/no-restart based on the product and patch
5. For a complex product, elaborate set of activities
6. Migration, verification of migration, support for older data

Software Retirement

1. Develop a retirement plan for system including timeline, communicate and execute that plan
2. Retire older version and running of new in parallel
3. Dispose superseded hardware and software

## Techniques or Approaches for Maintenance – Reverse Engineering

**Reverse Engineering**:
passive technique to understand a piece of software prior to re-engineering

- ▪Process of recovering specification and design information about the system from source code

Over time, application may have been corrected, adapted and enhanced

- •Application becomes complex, unstable and has unexpected and serious side effects

Identifies the components of a software product and the interrelationships between the components
Creates a representation of the software in a different format than existing code/documentation
Does not modify the product or result in a new product

Typical products of this process are graphs and control flow graphs

## Techniques or Approaches for Maintenance – Reverse Engineering

**Re-Engineering**:
process of modifying the software to make it easier to understand, change and extend

- Improving maintainability of system at reasonable cost

Considered from maintenance perspective, when software product is no longer viable to employ standard maintenance techniques to support with reasonable efficiency and productivity
May involve Refactoring
Re-Engineered code is expected to result in reduction of CPU Utilization, Readability, Usability and Performance
Steps

1. Identify current process
2. Document planned reconstruction
3. Reverse engineer
4. Refactor
5. Reconstruct code and data

PES UNIVERSITY ONLINE

## THANK YOU

**M S Anand**
Department of Computer Science Engineering

**anandms@yahoo.com**