Unit 1

Introduction

Software testing is a process of executing a program or application with the intent of finding the software bugs.

It can also be stated as the process of validating and verifying that a software program or application or product:

- o Meets the business and technical requirements that guided it's design and development
- o Works as expected
- o Can be implemented with the same characteristic.

Static Testing: It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing. For example: reviewing, walkthrough, inspection, etc.

Dynamic Testing: In dynamic testing the software code is executed to demonstrate the result of running tests. It's done during validation process. For example: unit testing, integration testing, system testing, etc.

Nature of Errors

Mainly Divided into three types

1. **Specifications are a common source of faults(Nature of Errors I)**. A software system has an overall specification, derived from requirements analysis. In addition, each component of the software ideally has an individual specification that is derived from architectural design.

The specification for a component can be:

- ambiguous (unclear)
- incomplete
- faulty.
- Misunderstood

Any such problems should, of course, be detected and remedied by verification of the specification prior to development of the component, but, of course, this verification cannot and will not be totally effective. So there are often problems with a component specification.

2. **During programming(Nature of Errors II),** the developer of a component may misunderstand the component specification. The next type of error is where a component contain faults so that it does not meet its specification. This may be due to **two kinds** of problem:

- errors in the logic of the code – an error of commission
- code that fails to meet all aspects of the specification – an error of omission.

This second type of error is where the programmer has failed to appreciate and correctly understand all the detail of the specification and has therefore omitted some necessary code.

3. **Finally, the kinds of errors that can arise in the coding of a component(Nature of Errors III)** are:

- data not initialised
- loops repeated an incorrect number of times.

- boundary value errors. Boundary values are values of the data at or near critical values.

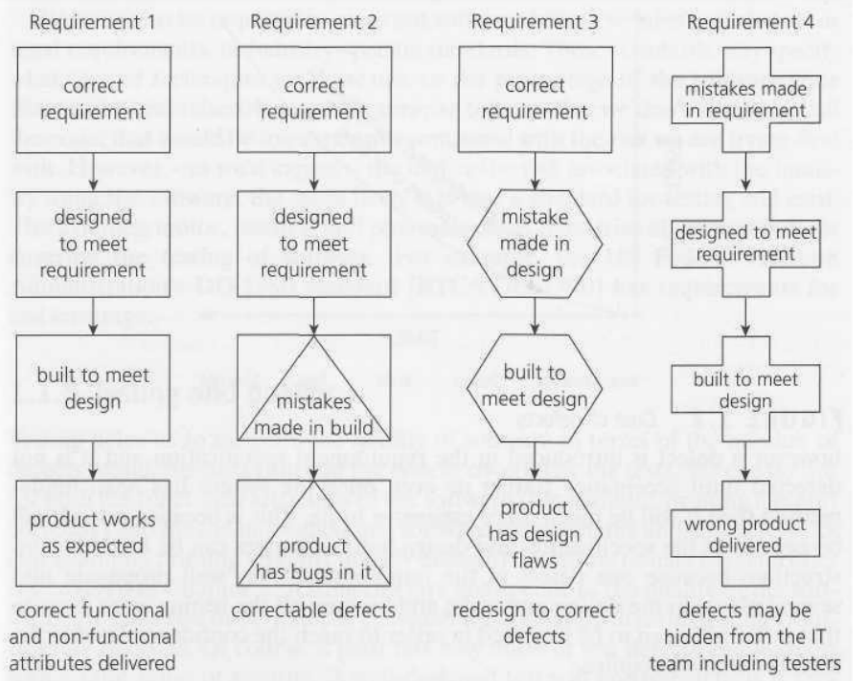


FIGURE 1.1 Types of error and defect

Some important Terms related to Software Testing

Quality management ensures that an organisation, product or service is consistent. It has four main components:

- Quality planning
- Quality assurance
- Quality control
- Quality improvement

Quality management is focused not only on product and service quality, but also on the means to achieve it. Quality management, therefore, uses **quality assurance** and control of processes as well as products to achieve more consistent quality.

Quality assurance (QA) is a way of preventing mistakes and defects in manufactured products and avoiding problems when delivering solutions or services to customers;

Quality control, or **QC** for short, is a process by which entities review the quality of all factors involved in production. This approach places an emphasis on three aspects

- Elements such as controls, job management, defined and well managed processes, performance and identification of records
- Competence, such as knowledge, skills, experience, and qualifications
- Soft elements, such as integrity, confidence, culture, motivation, team spirit, and quality relationships.

Software Quality Assurance (SQA)

- This consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards.
- SQA encompasses the entire software development process, which includes processes such as requirements definition, software design, coding, source code control, code reviews, software configuration management, testing, release management, and product integration.

- SQA is organised into goals, commitments, abilities, activities, measurements, and verifications.

Software Quality

A software quality factor is a
Non-functional requirement, but is a desirable requirement which enhances the quality of the software program.
None of these factors are binary.
Rather, they are characteristics that one seeks to maximize in one's software to optimize its quality.

- **Understandability-** Clarity of purpose
- **Completeness -**Presence of all constituent parts, with each part fully developed.
- **Conciseness-** Minimization of excessive or redundant information or processing.
- **Portability-** Ability to be run well and easily on multiple computer configurations.
  Consistency- Uniformity in notation, symbology, appearance, and terminology within itself.
- **Maintainability-** Propensity to facilitate updates to satisfy new requirements.
- **Testability-** Disposition to support acceptance criteria and evaluation of performance.
- **Usability-** Convenience and practicality of use.
- **Reliability-** Ability to be expected to perform its intended functions satisfactorily.
- **Efficiency-** Fulfilment of purpose without waste of resources.
- **Security-** Ability to protect data against unauthorized access

SDLC

As the Waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a **Linear-Sequential Life Cycle** Model.

Advantages of Waterfall Model

The advantage of waterfall development is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.
The waterfall model progresses through easily understandable and explainable phases and thus it is easy to use.

It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.

In this model, phases are processed and completed one at a time and they do not overlap. Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of Waterfall Model

It is difficult to estimate time and cost for each phase of the development process.

Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

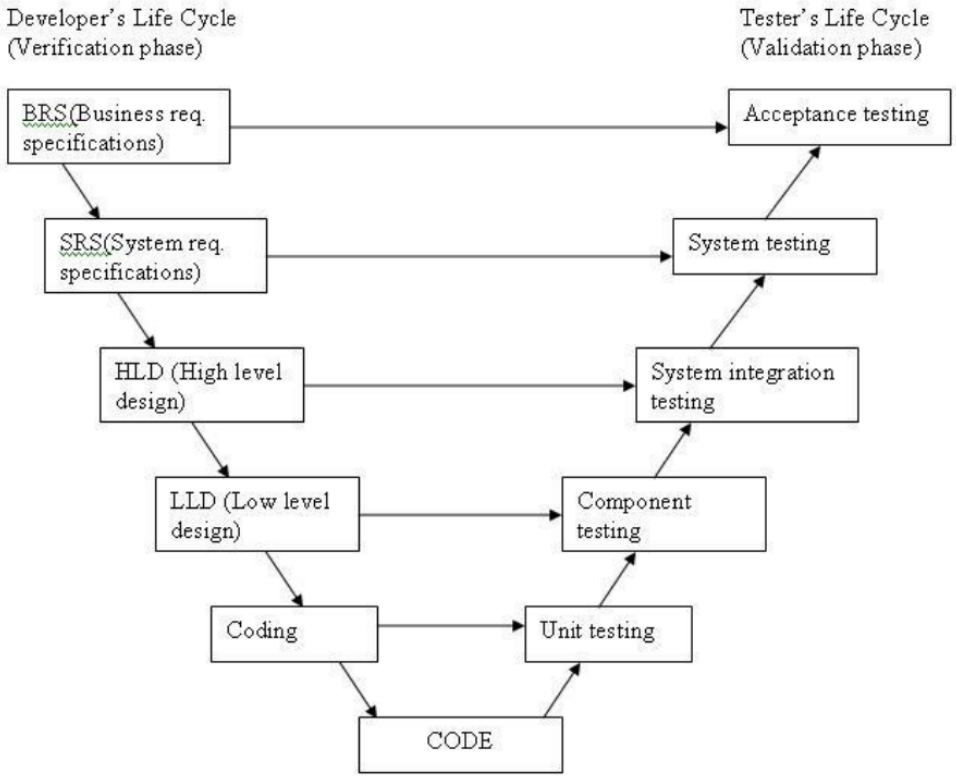
Not a good model for complex and object-oriented projects.

Not suitable for the projects where requirements are at a moderate to high risk of changing.

V model

V-Model :

The V-Model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as **Verification and Validation Model.** V-Model is an extension of the Waterfall Model and is based on association of a testing phase for each corresponding development stage.



Verification Phases

Business Requirement Analysis : In this first phase, the product requirements are understood from the customer perspective. The users are interviewed and a document called the **User Requirements Document** is generated. The user requirements document will typically describe the system's functional, interface, performance, data, security and other requirements as expected by the user.

System Design: In the phase, system developers analyze and understand the business of the proposed system by studying the user requirements document.
Architecture Design: This is also referred to as **High Level Design (HLD).** This phase focuses on system architecture and design. It provides overview of solution, platform, system, product and service/process. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken.
Module Design: The module design phase can also be referred to as **low-level design.** In this phase the actual software components are designed. It defines the actual logic for each and every component of the system.

Validation Phases
Unit Testing: Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage. A unit is the smallest entity which can independently exist, e.g. a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/units.
Component testing: searches for defects in and verifies the functioning of software components (e.g. modules, programs, objects, classes etc.) that are separately testable;
The Component Testing is like Unit Testing with the difference that all Stubs and Simulators are replaced with the real objects.
Integration Testing: Integration testing is associated with the architectural design phase. These tests verify that units created and tested independently can coexist and communicate among themselves within the system.
integration testing: tests interfaces between components, interactions to dif ferent parts of a system such as an operating system, file system and hard ware or interfaces between systems;
System Testing: System Tests Plans are developed during System Design Phase. System Test Plans are composed by client's business team. System Test ensures that expectations from application developed are met. The whole application is tested for its functionality, interdependency and communication. **User acceptance testing**: Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. UAT verifies that delivered system meets user's requirement and system is ready for use in real time.

Advantages
This is a highly disciplined model and Phases are completed one at a time.
Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
Simple and easy to understand and use.
Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
The implementation of testing starts right from the requirement phase, defects are found at early stage.
Works well for small projects where requirements are easily understood.

Disadvantages

Very rigid and least flexible so adjusting scope is difficult and expensive.

Software is developed during the implementation phase, so no early prototypes of the software are produced.

Not suitable for the projects where requirements are at a moderate to high risk of changing.

Verification and Validation

Verification is The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Verification is a **static practice** of verifying documents, design, code and program. It includes all the activities associated with producing high quality software: inspection, design analysis and specification analysis. It is a relatively objective process.

Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful. Verification is concerned with whether the system is well-engineered and error-free.

Methods of Verification : Static Testing

- Walkthrough
- Inspection
- Review

Validation is The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Validation is the process of evaluating the final product to check whether the software meets the customer expectations and requirements. It is a dynamic mechanism of validating and testing the actual product.

Methods of Validation : **Dynamic Testing**

- Testing
- End Users

Verification	**Validation**
1. **Verification** is a static practice of verifying documents, design, code and program.	1. **Validation** is a dynamic mechanism of validating and testing the actual product.

2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. **Verification** uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. **Validation** uses methods like black box (functional) testing and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. **Validation** is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.

Static Testing Techniques provide a powerful way to improve the quality and productivity of software development by assisting engineers to recognize and fix their own defects early in the software development process. In this software is tested without executing the code by doing **Review, Walk Through, Inspection or Analysis etc.**

Review :
Informal Review :
Informal reviews are applied at various times during the early stages in the life cycle of a document. A two-person team can conduct an informal review, as the author can ask a col-league to review a document or code.
Phases of a formal review In contrast to informal reviews, formal reviews follow a formal process. A typical formal review process consists of six main steps:

- Planning - The review process for a particular review begins with a 'request for review' by the author to the moderator (or inspection leader). A moderator is often assigned to take care of the scheduling (dates, time, place and invitation) of the review.
- Kick-off - An optional step in a review procedure is a kick-off meeting. The goal of this meeting is to get everybody on the same wavelength regarding the document under review and to commit to the time that will be spent on checking.

- Preparation - The participants work individually on the document under review using the related documents, procedures, rules and checklists provided.
- Review meeting - The meeting typically consists of the following elements (partly depending on the review type): logging phase, discussion phase and decision phase.
- Rework - Based on the defects detected, the author will improve the document under review step by step.
- Follow-up - The moderator is responsible for ensuring that satisfactory actions have been taken on all (logged) defects, process improvement suggestions and change requests.

Walkthrough

A walkthrough is characterized by the author of the document under review guiding the participants through the document and his or her thought processes, to achieve a common understanding and to gather feedback. This is especially useful if people from outside the software discipline are present, who are not used to, or cannot easily understand software development documents. The content of the document is explained step by step by the author, to reach consensus on changes or to gather information. Within a walkthrough the author does most of the preparation.

The participants, who are selected from different departments and backgrounds, are not required to do a detailed study of the documents in advance. Because of the way the meeting is structured, a large number of people can participate and this larger audience can bring a great number of diverse viewpoints regarding the contents of the document being reviewed as well as serving an educational purpose.

A walkthrough is especially useful for higher-level documents, such as requirement specifications and architectural documents.

The specific goals of a walkthrough depend on its role in the creation of the document. In general the following goals can be applicable:

to present the document to stakeholders both within and outside the soft ware discipline, in order to gather information regarding the topic under documentation;

to explain (knowledge transfer) and evaluate the contents of the docu ment;

to establish a common understanding of the document;

to examine and discuss the validity of proposed solutions and the viability of alternatives, establishing consensus.

Key characteristics of walkthroughs are:

The meeting is led by the authors; often a separate scribe is present.

Scenarios and dry runs may be used to validate the content.

Separate pre-meeting preparation for reviewers is optional.

Technical review

A technical review is a discussion meeting that focuses on achieving consensus about the technical content of a document. Compared to inspections, technical reviews are less formal. During technical reviews defects are found by experts, who focus on the content of the document. The experts that are needed for a technical review are, for example, architects, chief designers and key users. In practice, technical reviews vary from quite informal to very formal.

The goals of a technical review are to:

- assess the value of technical concepts and alternatives in the product and project environment;

- establish consistency in the use and representation of technical concepts;
- ensure, at an early stage, that technical concepts are used correctly;
- inform participants of the technical content of the document. Key characteristics of a technical review are:
- It is a documented defect-detection process that involves peers and technical experts.
- It is often performed as a peer review without management participation.
- Ideally it is led by a trained moderator, but possibly also by a technical expert.
- A separate preparation is carried out during which the product is examined and the defects are found.
- More formal characteristics such as the use of checklists and a logging list or issue log are optional.

Inspection

Inspection is the **most formal review type**. The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists. In the inspection meeting the defects found are logged and any discussion is postponed until the discussion phase. This makes the inspection meeting a very efficient meeting.

The generally accepted goals of inspection are to:

help the author to improve the quality of the document under inspection;
remove defects efficiently, as early as possible;
improve product quality, by producing documents with a higher level of quality;
create a common understanding by exchanging information among the inspection participants;
train new employees in the organisation's development process;
learn from defects found and improve processes in order to prevent recurrence of similar defects;

Key characteristics of an inspection are:

It is usually led by a trained moderator (certainly not by the author).
It uses defined roles during the process.
It involves peers to examine the product.
Rules and checklists are used during the preparation phase.
A separate preparation is carried out during which the product is examined and the defects are found.
The defects found are documented in a logging list or issue log.
A formal follow-up is carried out by the moderator applying exit criteria.
Optionally, a causal analysis step is introduced to address process improvement issues and learn from the defects found.
Metrics are gathered and analyzed to optimize the process.

Test Case Design Techniques :

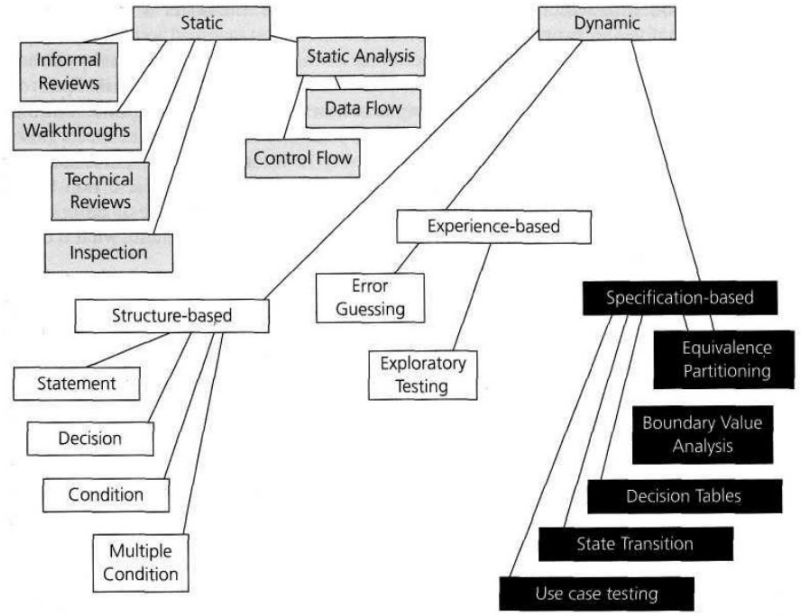


FIGURE 4.1 Testing techniques

BLACK BOX TESTING

SPECIFICATION-BASED OR BLACK-BOX TECHNIQUES

BLACK BOX TESTING, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or nonfunctional, though usually functional. This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- **Construction**: inappropriate use of #include statements cause construction errors.
- **Inadequate functionality**: These are errors caused by implicit assumptions in one part of a system that another part of the system would perform a function.However, in reality, the “other part” does not provide the expected functionality – intentionally or unintentionally by the programmer who coded the other part.
- **Location of Functionality**: Disagreement on or misunderstanding about the location of a functional capability within the software leads to this sort of error. The problem arises due to the design methodology, since these disputes should not occur at the code level. It is possible that inexperienced personnel contribute to the problem.
- **Changes in Functionality**: Changing one module without correctly adjusting for that change in other related modules affects the functionality of the program.
- **Added Functionality**: A completely new functional module, or capability, was added as a system modification. Any added functionality after the module is checked in to the version control system without a CR is considered to be an error.
- **Inadequate Error Processing**: A called module may return an error code to the calling module. However, the calling module may fail to handle the error properly.
- **Timing/Performance Problems** : These errors were caused by inadequate synchronisation among communicating processes.A race condition is an example of these kinds of error.

- **Initialization/Value Errors:**A failure to initialise, or assign, the appropriate value to a variable data structure leads to this kind of error.

Black Box Testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black-box testing method comes into use.

Techniques Following are some techniques that can be used for designing black box tests.

- **Equivalence Partitioning**: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- **Boundary Value Analysis:** It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- **Cause-Effect Graphing:** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

Advantages

• Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.

• Tester need not know programming languages or how the software has been implemented.

• Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developerbias.

• Test cases can be designed as soon as the specifications are complete.

Disadvantages

• Only a small number of possible inputs can be tested and many program paths will be left untested.

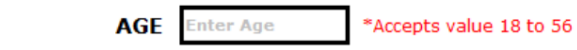
• Without clear specifications, which is the situation in many projects, test cases will be difficult to design.

• Tests can be redundant if the software designer/developer has already run a test case.

Equivalence partitioning (EP) is a good all-round specification-based blackbox technique. It can be applied at any level of testing and is often a good technique to use first. The idea behind the technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. Equivalence partitions are also known as equivalence classes - the two terms mean exactly the same thing. The equivalence-partitioning technique then requires that we need test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others.

Example 1:

Assume, we have to test a field which accepts Age 18 – 56



EQUIVALENCE PARTITIONING		
Invalid	**Valid**	**Invalid**
<=17	18-56	>=57

- Valid Input: 18 – 56
- Invalid Input:less than or equal to 17 (<=17), greater than or equal to 57 (>=57) Valid Class: 18 – 56 = Pick any one input test data from 18 – 56
- Invalid Class 1: <=17 = Pick any one input test data less than or equal to 17
- Invalid Class 2: >=57 = Pick any one input test data greater than or equal to 57 We have one valid and two invalid conditions here.

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.
• So these extreme ends like Start- End, Lower- Upper, MaximumMinimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
• The basic idea in boundary value testing is to select input variable values at their:
1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum

Examples 3: Input Box should accept the Number 1 to 10

Here we will see the Boundary Value Test Cases

Test Scenario Description	Expected Outcome
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept
Boundary Value = 9	System should accept
Boundary Value = 10	System should accept

Boundary Value = 11 System should NOT accept

Why Equivalence & Boundary Analysis Testing

- This testing is used to reduce very large number of test cases to manageable chunks.
- Very clear guidelines on determining test cases without compromising on the effectiveness of testing.

- Appropriate for calculation-intensive applications with large number of variables/inputs

Decision table testing

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs. However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface. The other two specification-based techniques, decision tables and state transition testing are more focused on business logic or business rules.

A **decision table** is a good way to deal with combinations of things (e.g. inputs). This technique is sometimes also referred to as a **'cause-effect'** table. The reason for this is that there is an associated logic diagramming technique called **'cause effect graphing'** which was sometimes used to help derive the decision table Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers. Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.

If you are a new customer opening a credit card account, you will get a 15% discount on all your purchases today. If you are an existing customer and you hold a loyalty card, you get a 10% discount. If you have a coupon, you can get 20% off today (but it can't be used with the 'new customer' discount). Discount amounts are added, if applicable.

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**	**Rule 7**	**Rule 8**
New customer (15%)	T	T	T	T	F	F	F	F
Loyalty card (10%)	T	T	F	F	T	T	F	F
Coupon (20%)	T	F	T	F	T	F	T	F
Actions								
Discount (%)	X	X	20	15	30	10	20	0

State transition testing is used where some aspect of the system can be described in what is called a **'finite state machine'**. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the 'machine'. This is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system. A finite state system is often shown as a **state diagram**

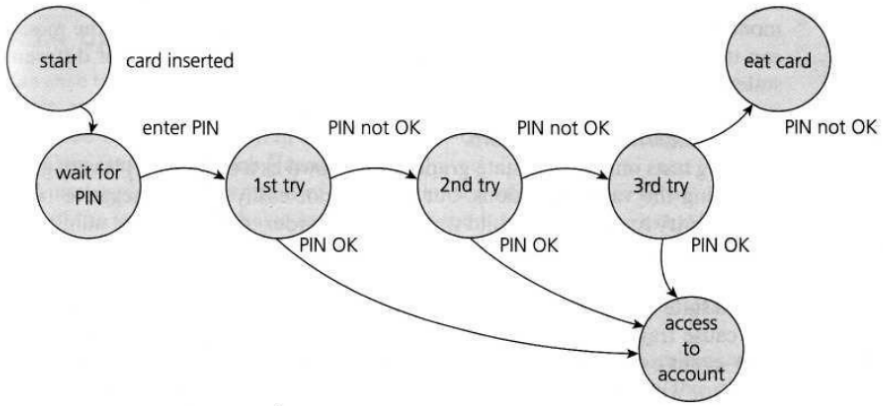


FIGURE 4.2 State diagram for PIN entry

Use case testing

Use case testing is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish. A use case is a description of a particular use of the system by an actor (a user of the system). Each use case describes the interactions the actor has with the system in order to achieve a specific task (or, at least, produce something of value to the user). Actors are generally people but they may also be other systems.

	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

FIGURE 4.3 Partial use case for PIN entry

White box TESTING

WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential.

Levels Applicable To White Box Testing method is applicable to the following levels of software testing: • Unit Testing: For testing paths within a unit. • Integration Testing: For testing paths between units. However, it is mainly applied to Unit Testing.

Advantages

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths.

Disadvantages

- Since tests can be very complex, highly skilled resources are required, with a thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation/platform may not be readily available.

1. Statement Coverage 2. Branch Coverage 3. Path Coverage Note that the statement, branch or path coverage does not identify any bug or defect that needs to be fixed. It only identifies those lines of code which are either never executed or remains untouched. Based on this further testing can be focused on.

Statement coverage: In a programming language, a statement is nothing but the line of code or instruction for the computer to understand and act accordingly. A statement becomes an executable statement when it gets compiled and converted into the object code and performs the action when the program is in a running mode. Hence “Statement Coverage”, as the name itself suggests, it is the method of validating whether each and every line of the code is executed at least once.

Example -

```
INPUT A & B
C = A + B
IF C>100
PRINT “ITS DONE”
```

For Statement Coverage – we would only need one test case to check all the lines of the code.

That means:

If I consider TestCase_01 to be (A=40 and B=70), then all the lines of code will be executed.

Now the question arises:

1. Is that sufficient?
2. What if I consider my Test case as A=33 and B=45?

Because Statement coverage will only cover the true side, for the pseudo code, only one test case would NOT be sufficient to test it. As a tester, we have to consider the negative cases as well.

Branch Coverage: “Branch” in a programming language is like the “IF statements”. An IF statement has two branches: True and False. So in Branch coverage (also

called Decision coverage), we validate whether each branch is executed at least once. In case of an “IF statement”, there will be two test conditions:
• One to validate the true branch and,
• Other to validate the false branch.
Hence, in theory, Branch Coverage is a testing method which is when executed ensures that each and every branch from each decision point is executed.

Hence for maximum coverage, we need to consider “Branch Coverage”, which will evaluate the “FALSE” conditions.
In the real world, you may add appropriate statements when the condition fails.
So now the pseudocode becomes:

```
INPUT A & B
C = A + B
IF C>100
PRINT “ITS DONE”
ELSE
PRINT “ITS PENDING”
```

Since Statement coverage is not sufficient to test the entire pseudo code, we would require Branch coverage to ensure maximum coverage.
So for Branch coverage, we would require two test cases to complete the testing of this pseudo code.
TestCase_01: A=33, B=45
TestCase_02: A=25, B=30
With this, we can see that each and every line of the code is executed at least once.
Here are the Conclusions that are derived so far:
• Branch Coverage ensures more coverage than Statement coverage.
• Branch coverage is more powerful than Statement coverage.
• 100% Branch coverage itself means 100% statement coverage.
• But 100 % statement coverage does not guarantee 100% branch coverage

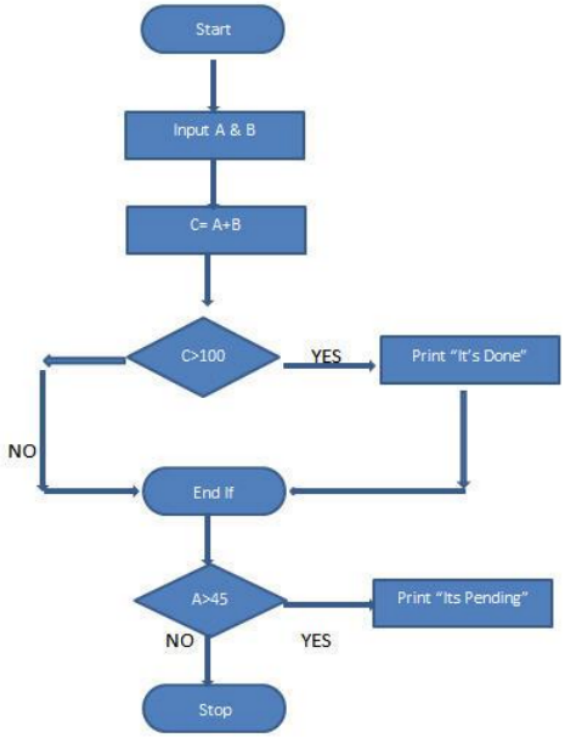
Path Coverage Path coverage tests all the paths of the program. This is a comprehensive technique which ensures that all the paths of the program are traversed at least once. Path Coverage is even more powerful than Branch coverage. This technique is useful for testing the complex programs.

Now let’s move on to Path Coverage:
As said earlier, Path coverage is used to test the complex code snippets, which basically involve loop statements or combination of loops and decision statements.
Consider this pseudocode:

```
INPUT A & B
C = A + B
```

IF C>100
PRINT “ITS DONE”
END IF
IF A>50
PRINT “ITS PENDING”
END IF
Now to ensure maximum coverage, we would require 4 test cases.
How? Simply – there are 2 decision statements, so for each decision statement, we would need two branches to test. One for true and the other for the false condition. So for 2 decision statements, we would require 2 test cases to test the true side and 2 test cases to test the false side, which makes a total of 4 test cases.

To simplify these let’s consider below flowchart of the pseudo code we have:



Path Coverage

In order to have the full coverage, we would need following test cases:

TestCase_01: A=50, B=60
TestCase_02: A=55, B=40
TestCase_03: A=40, B=65
TestCase_04: A=30, B=30

Criteria	Black Box Testing	White Box Testing
Definition	Testing without knowledge of internal structure/design/implementation	Testing with knowledge of internal structure/design/implementation
Levels applicable to	Mainly higher levels: Acceptance Testing, System Testing	Mainly lower levels: Unit Testing, Integration Testing
Responsibility	Independent Software Testers	Software Developers
Programming knowledge required	No	Yes
Implementation knowledge required	No	Yes
Basis for test cases	Requirements specifications	Detail design

Unit 2

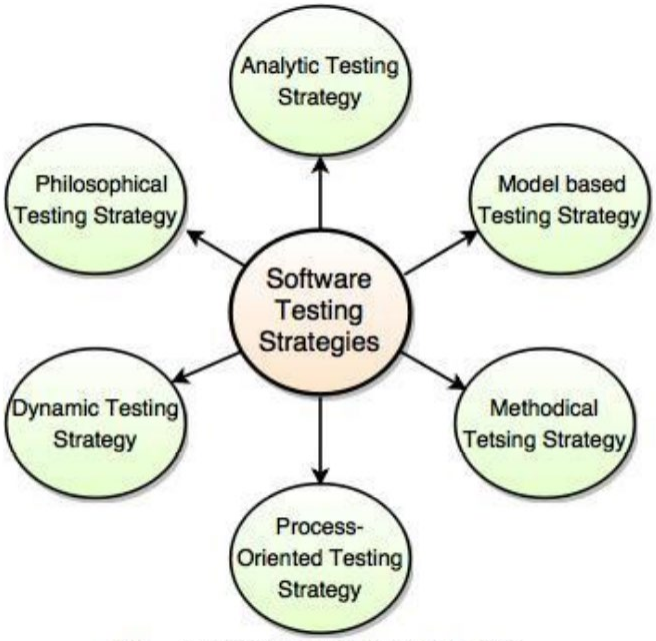


Fig. - Strategies of Software Testing

Analytic testing strategy

• Uses formal and informal techniques to access and prioritize risks that might occur during software testing.

• Takes a full overview of requirements, design and implementation of objects to determine the aim of testing.

• Collects complete information regarding software, target that is achieved and the data needed for testing the software.

ii) **Model-based testing strategy**
• Tests the functionality of the software.
• Identifies the domain of data and selects suitable test cases as per the probability of errors in that domain.
iii) **Methodical testing strategy**
• Tests the function and status of software according to checklist, based on the user requirements.
• This strategy is used to test the functionality, reliability, usability and performance of the software.
iv) **Process-oriented testing strategy**
• Tests the software from the existing standards i.e IEEE standards.
• Ensures the functionality of the software by using automated testing tools.
v) **Dynamic testing strategy**
• Tests the software after having common decision of the testing team.
• Gives information regarding the software, for e.g., the test cases used for testing the errors present in it.
vi) **Philosophical testing strategy**
• Tests the software by assuming that any component of the software terminates the functioning anytime.
• For testing the software it takes help from software developers, users and system analysis

Levels of Software Testing

1) **Unit testing:**
A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.
The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.
2) **Integration testing:**
Integration means combining. In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.
Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.
3) **System testing**:
System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.
System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the Testing.
4) **Acceptance testing:**
Acceptance testing is a test conducted to find if the requirements of a

specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.
Other Types of Testing:

REGRESSION TESTING is a type of software testing that intends to ensure that changes (enhancements or defect fixes) to the software have not adversely affected it.
The likelihood of any code change impacting functionalities that are not directly associated with the code is always there and it is essential that regression testing is conducted to make sure that fixing one thing has not broken another thing. During regression testing, new test cases are not created but previously created test cases are re-executed. Some tend to include Regression Testing as a separate level of software testing but that is a misconception. Regression Testing is, in fact, just a type of testing that can be performed at any of the four main levels.

Alpha Testing - Alpha testing is one of the most common software testing strategy used in software development. Its specially used by product development organisations.
This test takes place at the developer's site. Developers observe the users and note problems. Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing. Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers. Alpha testing is final testing before the software is released to the general public. It has two phases:

- In the first phase of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware assisted debuggers. The goal is to catch bugs quickly.
- In the second phase of alpha testing, the software is handed over to the software QA staff, for additional testing in an environment that is similar to the intended use.

Beta Testing - Beta Testing is also known as field testing. It takes place at customer's site. It sends the system/software to users who install it and use it under real-world working conditions.
A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. (Beta is the second letter of the Greek alphabet.) Originally, the term alpha testing meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing.
The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user's perspective that you would not want to have in your final, released version of the application. Example: Microsoft and many other organizations release beta versions of their products to be tested by users.

Software Metrics
• A measure of some property of a piece of software or its specifications
• They are all measurable, that is they can be quantified.
Some common software metrics are:-

1. LOC
2. Cyclomatic complexity, is used to measure code complexity.
3. Function point analysis (FPA), is used to measure the size (functions) of software.
4. Bugs per lines of code.
5. Code coverage - Code lines that are executed for a given set of software tests.
6. Cohesion - how well the source code in a given module work together to provide a single function.
7. Coupling - how well two software components are data related, i.e. how independent they are.

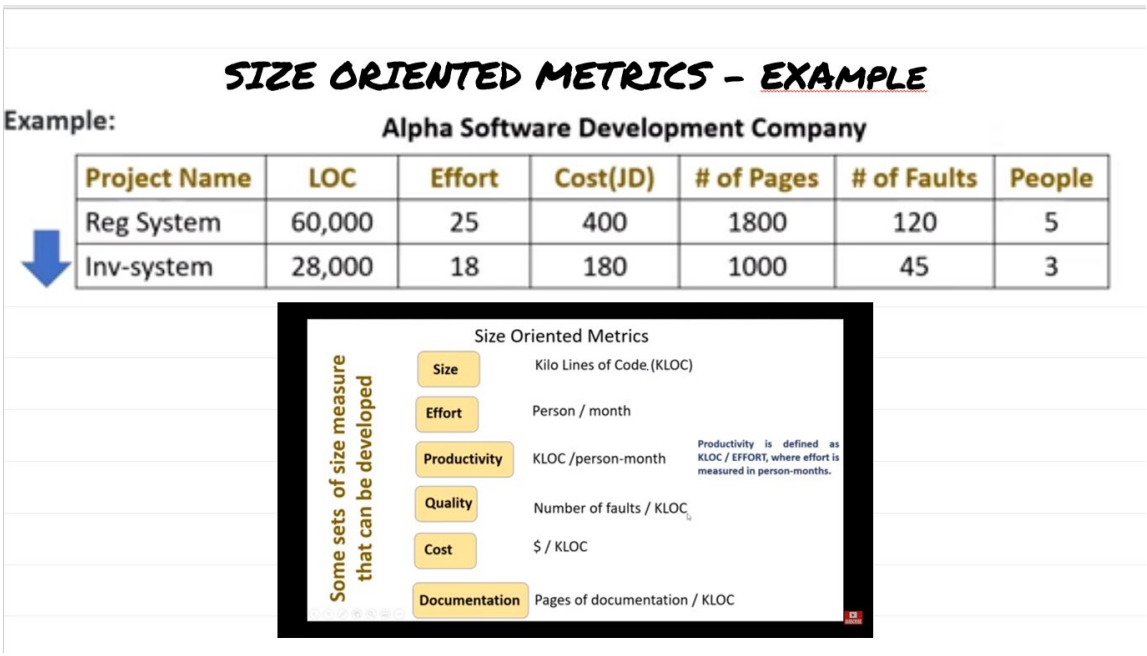
Size-Oriented Metrics

Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced. If a software organization maintains simple records, a table of sizeoriented measures, such as

OR

Size-oriented metrics are a category of software metrics used to measure and evaluate software products based on their size or the volume of code. These metrics focus on the physical or logical size of the software, such as lines of code, function points, or object points. Size-oriented metrics are valuable for estimating software development efforts, assessing productivity, and managing project progress. Some common size-oriented metrics include:

Lines of Code (LOC): LOC is one of the most basic and widely used size-oriented metrics. It measures the size of a software program by counting the number of lines in the source code. LOC can be categorized into physical LOC (including comments and whitespace) and logical LOC (excluding comments and whitespace).



Function-Oriented Metrics

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalisation value. Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity. Function points are computed [IFP94] by completing the table shown in Figure. Five information domain characteristics are determined and counts are provided in the appropriate table location. Information domain values are defined in the following manner:

Number of user inputs. Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.
Number of user outputs. Each user output that provides application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
Number of user inquiries. An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an online output. Each distinct inquiry is counted.
Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.
Number of external interfaces. All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.
Once these data have been collected, a complexity value is associated with each count. Organisations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

To compute function points (FP), the following relationship is used:
FP = count total * (0.65 + 0.01 * sum of (Fi))
where count total is the sum of all FP entries
The Fi (i = 1 to 14) are "complexity adjustment values" based on responses to the following questions

			Weighting factor			
Measurement parameter	**Count**		**Simple**	**Average**	**Complex**	
Number of user inputs		×	3	4	6	=
Number of user outputs		×	4	5	7	=
Number of user inquiries		×	3	4	6	=
Number of files		×	7	10	15	=
Number of external interfaces		×	5	7	10	=
Count total →						

RECONCILING DIFFERENT METRICS APPROACHES

The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design. A number of studies have attempted to relate FP and LOC Measures.

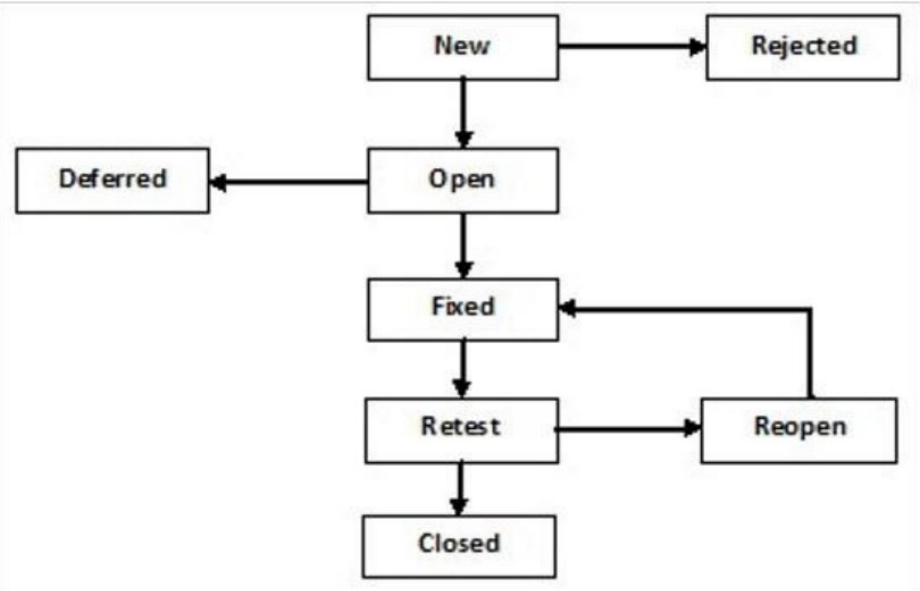
What is Defect?

A Defect, in simple terms, is a flaw or an error in an application that is restricting the normal flow of an application by mismatching the expected behavior of an application with the actual one. The defect occurs when any mistake is made by a developer during the designing or building of an application and when this flaw is found by a tester, it is termed as a defect. It is the responsibility of a tester to do a thorough testing of an application with an intention to find as many defects as possible so as to ensure that a quality product will reach the customer. It is important to understand about defect life cycle before moving to the workflow and different states of the defect.

Defect Life Cycle in Detail

A Defect life cycle, also known as a Bug life cycle, is a cycle of a defect from which it goes through covering the different states in its entire life. This starts as soon as any new defect is found by a tester and comes to an end when a tester closes that defect assuring that it won't get reproduced again

Defect Workflow:



Defect States:

1. New: This is the first state of a defect in the defect life cycle. When any new defect is found, it falls in a 'New' state and validations and testing are performed on this defect in the later stages of the defect life cycle.
2. Assigned: In this stage, a newly created defect is assigned to the development team for working on the defect. This is assigned by the project lead or the manager of the testing team to a developer.
3. Open: Here, the developer starts the process of analyzing the defect and works on fixing it, if required. If the developer feels that the defect is not appropriate then it may get transferred to any of the below four states namely Duplicate, Deferred, Rejected or Not a Bug-based upon the specific reason. I will discuss these four states in a while.
4. Fixed: When the developer finishes the task of fixing a defect by making the required changes then he can mark the status of the defect as 'Fixed'.

5. Pending Retest: After fixing the defect, the developer assigns the defect to the tester for retesting the defect at their end and till the tester works on retesting the defect, the state of the defect remains in ‘Pending Retest’.
6. Retest: At this point, the tester starts the task of working on the retesting of the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.
7. Reopen: If any issue still persists in the defect then it will be assigned to the developer again for testing and the status of the defect gets changed to ‘Reopen’.
8. Verified: If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to ‘Verified’.
9. Closed: When the defect does not exist any longer then the tester changes the status of the defect to ‘Closed’. Few More:
10. Rejected: If the defect is not considered as a genuine defect by the developer then it is marked as ‘Rejected’ by the developer.
11. Duplicate: If the developer finds the defect as same as any other defect or if the concept of the defect matches with any other defect then the status of the defect is changed to ‘Duplicate’ by the developer.
12. Deferred: If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, he can change the status of the defect as ‘Deferred’.
13. Not a Bug: If the defect does not have an impact on the functionality of the application then the status of the defect gets changed to ‘Not a Bug’. The mandatory fields when a tester logs any new bug are Build version, Submit On, Product, Module, Severity, Synopsis and Description to Reproduce

Defect management process is explained below in detail.

#1) **Defect Prevention:**

Defect Prevention is the best method to eliminate the defects in the early stage of testing instead of finding the defects in the later stage and then fixing it. This method is also cost effective as the cost required for fixing the defects found in the early stages of testing is very low.

However, it is not possible to remove all the defects but at least you can minimize the impact of the defect and cost to fix the same.

The major steps involved in Defect Prevention are as follow:

Identify Critical Risk: Identify the critical risks in the system which will impact more if occurred during testing or in the later stage.

Estimate Expected Impact: For each critical risk, calculate how much would be the financial impact if the risk actually encountered.

Minimize expected impact: Once you identify all critical risks, take the topmost risks which may be harmful to the system if encountered and try to minimize or eliminate the risk. For risks which cannot be eliminated, it reduces the probability of occurrence and its financial impact.

#2) **Deliverable Baseline:**

When a deliverable (system, product or document) reaches its pre-defined milestone then you can say a deliverable is a baseline. In this process, the product or the deliverable moves from one stage to another and as the deliverable moves from one stage to another, the existing defects in the system also gets carried forward to the next milestone or stage.

For Example, consider a scenario of coding, unit testing and then system testing. If a developer performs coding and unit testing then system testing is carried

out by the testing team. Here coding and Unit Testing is one milestone and System Testing is another milestone.
So during unit testing, if the developer finds some issues then it is not called as a defect as these issues are identified before the meeting of the milestone deadline. Once the coding and unit testing have been completed, the developer hand-overs the code for system testing and then you can say that the code is “baselined” and ready for next milestone, here, in this case, it is “system testing”.
Now, if the issues are identified during testing then it is called as the defect as it is identified after the completion of the earlier milestone i.e. coding and unit testing.
Basically, the deliverables are baselined when the changes in the deliverables are finalized and all possible defects are identified and fixed. Then the same deliverable passes on to the next group who will work on it.

#3) **Defect Discovery:**

It is almost impossible to remove all the defects from the system and make a system as a defect-free one. But you can identify the defects early before they become costlier to the project. We can say that the defect discovered means it is formally brought to the attention of the development team and after analysis of that the defect development team also accepted it as a defect.
Steps involved in Defect Discovery are as follows:

- Find a Defect: Identify defects before they become a major problem to the system.
- Report Defect: As soon as the testing team finds a defect, their responsibility is to make the development team aware that there is an issue identified which needs to be analyzed and fixed.
- Acknowledge Defect: Once the testing team assigns the defect to the development team, its the development team’s responsibility to acknowledge the defect and continue further to fix it if it is a valid defect.

#4) **Defect Resolution:**

In the above process, the testing team has identified the defect and reported to the development team. Now here the development team needs to proceed for the resolution of the defect.
The steps involved in the defect resolution are as follows:

- Prioritize the risk: Development team analyzes the defect and prioritizes the fixing of the defect. If a defect has more impact on the system then they make the fixing of the defect on a high priority.
- Fix the defect: Based on the priority, the development team fixes the defect, higher priority defects are resolved first and lower priority defects are fixed at the end.
- Report the Resolution: Its the development team’s responsibility to ensure that the testing team is aware when the defects are going for a fix and how the defect has been fixed i.e. by changing one of the configuration files or making some code changes. This will be helpful for the testing team to understand the cause of the defect.

#5) Process Improvement:

Though in the defect resolution process the defects are prioritized and fixed, from a process perspective, it does not mean that lower priority defects are not important and are not impacting much to the system. From process improvement point of view, all defects identified are same as a critical defect. Even these minor defects give an opportunity to learn how to improve the

process and prevent the occurrences of any defect which may impact system failure in the future. Identification of a defect having a lower impact on the system may not be a big deal but the occurrences of such defect in the system itself is a big deal.
For process improvement, everyone in the project needs to look back and check from where the defect was originated. Based on that you can make changes in the validation process, base-lining document, review process which may catch the defects early in the process which are less expensive.

Unit 3

Quality concepts

Variation control

All engineered and manufactured parts exhibit variation. The variation between samples may not be obvious without the aid of precise equipment to measure the geometry, electrical characteristics, or other attributes of the parts. However, with sufficiently sensitive instruments, we will likely come to the conclusion that no two samples of any item are exactly alike.
Variation control is the heart of quality control. A manufacturer wants to minimize the variation among the products that are produced, even when doing something relatively simple like duplicating diskettes. From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time.

When we examine an item based on its measurable characteristics, two kinds of quality may be encountered: quality of design and quality of conformance. **Quality of design** refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design. As higher-grade materials are used, tighter tolerances and greater levels of performance are specified, the design quality of a product increases, if the product is manufactured according to specifications.
Quality of conformance is the degree to which the design specifications are followed during manufacturing. Again, the greater the degree of conformance, the higher is the level of quality of conformance.

Quality Control

Variation control may be equated to quality control. But how do we achieve quality control? Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications. This approach views quality control as part of the manufacturing process. Quality control activities may be fully automated, entirely manual, or a combination of both.

Quality Assurance

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.

Cost of Quality

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities. Cost of quality studies are conducted to provide a base- line for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison. The basis of normalization is almost always dollars. Once we have normalized quality costs on a dollar basis, we have the necessary data to evaluate where the opportunities lie to improve our processes. Furthermore, we can evaluate the effect of changes in dollar-based terms.

Quality costs may be divided into costs associated with prevention, appraisal, and failure. Prevention costs include

- quality planning
- formal technical reviews
- test equipment
- training

Appraisal costs include activities to gain insight into product condition the “first time through” each process. Examples of appraisal costs include

- in-process and interprocess inspection
- equipment calibration and maintenance
- testing

Failure costs are those that would disappear if no defects appeared before shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs.

Internal failure costs are incurred when we detect a defect in our product prior to shipment. Internal failure costs include

- rework
- repair
- failure mode analysis

External failure costs are associated with defects found after the product has been shipped to the customer. Examples of external failure costs are

- complaint resolution
- product return and replacement
- help line support
- warranty work

Six Sigma Rules

Write a note on Six Sigma for software engineering. (Repeat: 5)

Six Sigma is the most widely used strategy for statistical quality assurance in industry today. Originally

popularized by Motorola in the 1980s, the Six Sigma strategy “is a rigorous and disciplined methodology

that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes". The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences— implying an extremely high quality standard.
The Six Sigma methodology defines three core steps:

- Define customer requirements and deliverables and project goals via welldefined methods of customer communication.
- Measure the existing process and its output to determine current quality performance (collect defect metrics).
- Analyse defect metrics and determine the vital few causes.

If an existing software process is in place, but improvement is required, Six Sigma suggests two
additional steps:

- Improve the process by eliminating the root causes of defects.
- Control the process to ensure that future work does not reintroduce the causes of defects.These core and additional steps are sometimes referred to as the DMAIC (define, measure, analyze, improve, and control) method.

If an organization is developing a software process (rather than improving an existing process), the core
steps are augmented as follows:
Design the process to (1) avoid the root causes of defects and (2) to meet customer requirements.
Verify that the process model will, in fact, avoid defects and meet customer requirements

Q-1	**Attempt any THREE Questions :**	**(21)**
a)	Explain any 7 desirable qualities of a Software project	(7)
b)	Write at least 7 differences between Black box Technique and Whitebox Technique	(7)
c)	Explain Boundary Value Analysis with at least 2 examples	(7)
d)	Consider the process of Hotel Room Reservation System. Enumerate the list of states and draw a state transition Diagram	(7)

Q-2	**Attempt any THREE Questions :**	**(21)**
a)	How does Function oriented metric work ? What are the different factors considered in this metric and write the formula and	(7)
b)	Explain Defect Life Cycle with a neat diagram	(7)
c)	How is Cyclomatic Complexity calculated ? Explain it with the help of a coding example. Discuss its limitations	(7)
d)	What is Halstead metric used for ? In a programming module, the number of unique operators present are 10, the number of unique operands are 5. Total number of operator occurrences and operand occurrences are 18 and 13. Calculate the Vocabulary, Program volume and Program Difficulty.	(7)

Q-3 Attempt any THREE Questions : **(21)**

a) What is the role of SQA group ? Explain the activities performed by the group. (7)

b) List and explain at least 7 Review Guidelines (7)

c) Explain Cause-Effect Diagram and Create a Cause-Effect Diagram for a Defective Product Delivery Problem (Hint : Different Errors may be the main cause). (7)

d) Elaborate on the Six Sigma process of Software Engineering (7)

Q-4 Answer any THREE questions **(12)**

a) Observe the table below and frame the conditions that are mentioned in the table. **(4)**

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Over 23?	F	T	T	T
Clean driving record?	Don't care	F	T	T
On business?	Don't care	Don't care	F	T
Actions				
Supply rental car?	F	F	T	T
Premium charge?	F	F	F	T

b) Write at least 4 principles of ISO:9000 standard **(4)**

c) Write at least 4 advantages of static testing technique **(4)**

d) Given the following, compute the FP when all complexity adjustment factors and weighted factors have average range (4, 5, 4, 10, 7). **(4)**
User i/p – 40, User o/p – 30, User inquiries – 25, Files – 4 External interfaces – 6

f) Explain Cause-Effect Diagram with a good example.

g) Write a note on ISO 9000 Quality Standards.

h) List out at least 5 causes of errors that can be traced in Statistical SQA.

a) Write a note on the nature of errors that can occur in SDLC.

b) Observe the following code and Perform the following.

- Create a Control Flow Graph
- Create Test cases for Statement Coverage, Branch Coverage

```
if(extras >= 3)
        addon_discount = 10;
else if (extras >= 5)
        addon_discount = 15;
else addon_discount = 0;
if (discount > addon_discount)
        addon_discount = discount;
result = baseprice/100.0*(100-addon_discount);
return(result);
```

c) Write at least 5 differences between Blackbox Technique and Whitebox Technique.

d) Write a note on Walkthrough technique.

e) Explain Boundary Value Analysis with at least 2 examples.

...separated by a comma.

Attempt **any four** questions: **20**

a) What are the different types of errors that can occur during SDLC. Explain them with a diagram **5**

b) Write a note on Quality Management. **5**

c) Explain the V-Model with a neat diagram. **5**

d) Write a note on Technical Review. **5**

e) Explain the technique of Boundary value Analysis with Atleast 2 examples. **5**

f) What is White-Box Testing? Explain any two types of White-Box Testing. **5**

Attempt **any four** questions: **20**

a) Explain the Analytic Testing strategy and Model Based Testing Strategy of Software Testing. **5**

b) Write a note on the following terms of testing. Take a scenario and fit the terms in the scenario.

   i) Alpha Testing

   ii) Beta Testing

   iii) Regression Testing

c) Write a note on Size-Oriented Metrics.

d) Write any Five complexity Adjustment Factors considered in Function Oriented Metrics.

e) What is a defect? What are the metrics related to defects?

f) Explain Defect Life Cycle with a neat diagram.

Attempt **any four** questions:

a) Write a note on the following elements of SQA

   i) Standards

   ii) Reviews and Audits

   iii) Vendor Management

b) What are the steps in Statistical SQA?

c) i) Define Software Reliability

   ii) Suppose that a certain software product has a mean time between failure of 40,000 hours and a mean time to repair of 80 hours, what is its availability?

d) Write a note on Six sigma for software engineering.

e) What are the SQA activities recommended by software Engineering Institute.

f) Explain Pareto Diagram with a good example.