UNIT III

3 Test Case Design Techniques

Syllabus

Software Testing Methodologies : *White Box Testing, Black Box Testing, Grey Box Testing. Test Case Design Techniques : Static Techniques : Informal Reviews, Walkthroughs, Technical Reviews, Inspection. Dynamic Techniques: Structural Techniques : Statement Coverage Testing, Branch Coverage Testing, Path Coverage Testing, Conditional Coverage Testing, Loop Coverage Testing Black Box Techniques : Boundary Value Analysis, Equivalence Class Partition, State Transition Technique, Cause Effective Graph, Decision Table, Use Case Testing, Experienced Based Techniques : Error guessing, Exploratory testing.*

Levels of Testing : *Functional Testing : Unit Testing, Integration Testing, System Testing, User Acceptance Testing, Sanity/Smoke Testing, Regression Test, Retest. Non-Functional Testing : Performance Testing, Memory Test, Scalability Testing, Compatibility Testing, Security Testing, Cookies Testing, Session Testing, Recovery Testing, Installation Testing, Adhoc Testing, Risk Based Testing, I18N Testing, L1ON Testing, Compliance Testing.*

Contents

3.1 Software Testing Methodologies

SPPU : Dec.-19

- Software testing methodologies are the various strategies or approaches used to test an application to ensure it behaves and looks as expected. These encompass everything from front to back-end testing, including unit and system testing.

3.1.1 White Box Testing

- White box testing builds upon the internals of a software component. It is the detailed investigation of internal logic and structure of the code. In white box testing it is necessary for a tester to have full knowledge of source code. Fig. 3.1.1 shows white box testing.

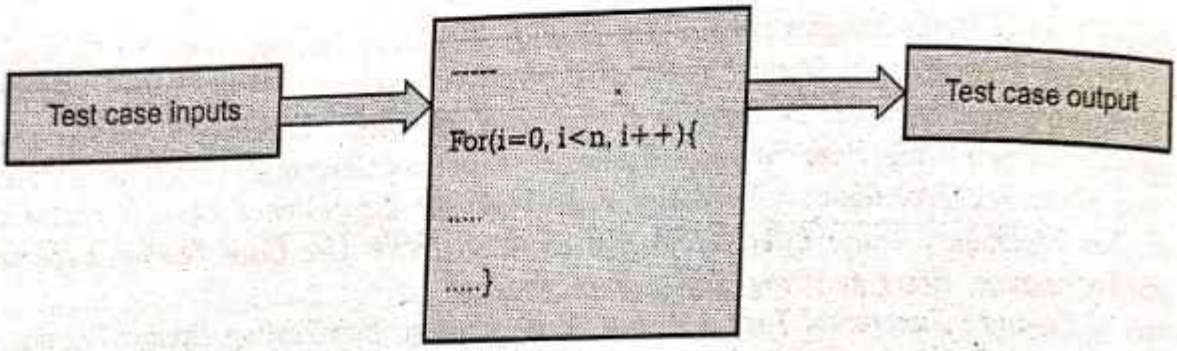

Fig. 3.1.1 White box testing

- White box testing is a test case design method that uses the control structure of the procedural design to derive test cases. White box testing can uncover implementation errors such as poor key management by analyzing internal workings and structure of a piece of software. White box testing is applicable at integration, unit and system levels of the software testing process.
- White box testing strategies use the source code of software components as the basic information for building test suites.
- For both black box testing and white box testing, the software components under test are routines. A routine can range from a very simple function to a complete program.
- **Advantages of white box testing :**
  1. Can be done by development team
  2. Does not require a completed GUI
  3. Highly systematic / easy to reproduce
  4. Thorough - Can test every possible path
  5. Can uncover errors early in the development process

- **Disadvantages of white box testing :**
  1. Time required of developers
  2. Code updates can invalidate test cases
  3. Hard to separate the program and the testing environment.

3.1.2 Black Box Testing

- This testing methodology looks at what are the available inputs for an application and what the expected outputs are that should result from each input.
- Black box testing is also called **functional testing.** The main ideas are simple : Define initial component state, input and expected output for the test. Set the component in the required state. Give the defined input and observe the output and compare to the expected output. Fig. 3.1.2 shows black box testing.

Fig. 3.1.2 Black box testing

- No knowledge about the internals of the code is required. Test cases are designed based on specifications. In black box testing, we ignore the internals of the system and focus on relationship between inputs and outputs.
- An example of a black-box system would be a search engine. You enter text that you want to search for in the search bar, press "search" and results are returned to you. In such a case, you do not know or see the specific process that is being employed to obtain your search results, you simply see that you provide an input, a search term and you receive an output, your search results.
- Black-box testing is testing from a functional or behavioral perspective to ensure a program meets its specification. Black-box testing is focused on results.
- **Advantages :**
  1. Easy to use.
  2. Quicker test case development.
  3. Efficient for large code segment.
  4. Tester perception is very simple.

- **Disadvantages :**
  1. Only a selected number of test scenarios are actually performed. As a result, there is only limited coverage.
  2. Without clear specification test cases are difficult to design.
  3. Inefficient testing.

3.1.3 Difference between Black Box and White Box Testing

Black box testing	White box testing
Also called as functional testing techniques.	Also called as structural testing technique.
Black box testing is mainly higher level, as in system and acceptance testing, so implementation comes later in the development cycle.	White box testing usually begins early in the development cycle. It is conducted at lower levels and includes unit and integration testing.
Programming knowledge is not required.	Programming knowledge is required.
May not have access to the source code.	May have access to the source code.
Black box testing techniques are equivalence partitioning, boundary value analysis, orthogonal array testing, all pair testing, state transition testing.	White box testing techniques are control flow testing, branch testing, basis path testing, data flow testing and loop testing.
Mainly applicable to higher level of techniques.	Mainly applicable to lower level of techniques.

3.1.4 Grey Box Testing

- This testing technique is a combination of black box testing and white box testing.
- In black box testing, the tester does not have any knowledge about the code. They have information for what will be the output for the given input. In white box testing, the tester has complete knowledge about the code.
- Grey box tester has knowledge of the code, but not completely.
- In gray box testing, the tester is not required to design test cases. Instead, test cases are created based on algorithms that evaluate internal states, program behavior and application architecture knowledge. The tester then carries out and interprets the results of these tests.
- The purpose of Gray box testing is to improve the quality of the product for which it covers both functional and non-functional testing altogether, which saves time and the lengthy process to test the application.
- The techniques used for gray box testing are matrix testing, regression testing, orthogonal array testing and pattern testing.

- Matrix testing is a technique that examines all variables in an application.
- Regression testing is a technique that enables user to verify whether application changes or bug fixes have caused errors to appear in existing components.
- Pattern testing is a technique that evaluates past defects to identify patterns that lead to defects.
- Orthogonal array testing is a technique user can use when our application has only a few inputs that are too complex or large for extensive testing. This technique enables user to perform test case optimization, where the quality and number of tests performed balance test coverage with effort.
- **Advantages and disadvantages of gray box testing :**

1. Advantages of gray box testing

- Testers do not need to have a programming expertise
- It can provide the benefits of both black and white box testing
- It can eliminate conflicts between developers and testers
- It is cheaper than integration testing.

2. Disadvantages of gray box testing

- Code path traversals are limited due to restricted access to internal application structure.
- It cannot be used for algorithm testing.
- Test cases can be difficult to design.

Review Question

1. *What is white box testing and black box testing ? What are the different methods for white box and black box testing ? Explain any one methods of each in detail.*

SPPU : Dec.-19, End Sem, Marks 4

3.2 Test Case Design Techniques : Static Techniques

- A test design technique is used to select a good set of tests from all possible tests for a given system.
- Static testing do not execute code. It manually checks work documents to find errors in early stage. It define checking the software product and related artifacts without executing them.

- Static testing of software is where methods are used to detect defects in the software but the software is not actually executed. It uses white box testing method. It may include reviews, walkthroughs, inspections and audits.
- A static technique is so named because it involves no execution of code, product, documentation etc.
- The main goal of static testing is to check if the coding and the algorithm of software works properly. Static testing helps to indicate code errors at the beginning of testing process and to make further testing more efficient and productive.
- **Some static testing methods are :**
  1. Code inspections, where software code is manually reviewed for defects
  2. Code analyzers, where code is analyzed by a tool that will check the code for things such a type errors, memory leaks, syntax errors, etc.
  3. Compiling, compiling code using the source compiler, or interpreter.
  4. Modeling techniques, where software requirements can be modeled to verify that they are correct.
- Static testing finds bugs before you compile. The earlier you detect bugs the cheaper they are to fix.

3.2.1 Advantages and Disadvantages of Static Testing

1. Advantages of Static Testing

- Since static testing can start early in the life cycle, early feedback on quality issues can be established.
- By detecting defects at an early stage, rework costs are most often relatively low.
- Since rework effort is substantially reduced, development productivity figures are likely to increase.
- The evaluation by a team has the additional advantage that there is an exchange of information between the participants.
- Static tests contribute to an increased awareness of quality issues.
- Highest probability of finding defects.

2. Disadvantages of Static Testing

- Time-consuming
- Cannot test data dependencies
- High skill levels required.

3.2.2 Informal Reviews

- 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 colleague to review a document or code.
- This normally involves peers of the author, who try to find defects in the document under review and discuss these defects in a review meeting.
- The goal is to help the author and to improve the quality of the document. Informal reviews come in various shapes and forms, but all have one characteristic in common - they are not documented.
- The material may be as informal as a computer listing or hand-written documentation.

3.2.3 Walkthroughs

- It is a semi-formal type of review. Reviews of products by groups of people mostly without preparation.
- For example a requirements traceability review is a walkthrough. It involves tracing a requirement from customer requirements to the test procedures. All issues found during these reviews are documented on CAR forms.
- A form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product and the participants ask questions and make comments about possible errors, violation of development standards and other problems.
- The producer is the review leader, therefore the utility of a walkthrough depends on the real goal of the producer.
- Walkthroughs are most effective at the source code level and on other small work product.

3.4 Inspections

- Software inspections are a disciplined engineering practice for detecting and correcting defects in software artifacts and preventing their leakage into field operations.

An inspection is a more formal process than a walkthrough used to collect metrics or statistics about the software process. Walkthrough is a more informal version of an inspection

- Inspections are a method to reduce variability and tighten process control. Inspections address three major tasks of process management : Planning, measurement, control.
- Inspections are used to collect quantitative quality data at defined points in the development process. This can be used to give feedback to the developers, feed-forward to future development and feed-into future steps of process. It can also provide data on effectiveness of inspection techniques.
- Inspections can be held a various points in development process. Michael Fagan recommended inspections on : Detailed design, cleanly compiled code and completion of unit test.
- At a minimum a formal inspection includes designated moderator, author of the work, at least one peer inspector. Walkthroughs generally do not include designated moderator and are often led by the author of the software.
- **Steps of inspection** : Planning, overview, preparation, meeting, rework and follow-up.

3.2.5 Comparison between Walkthrough and Inspection

Sr. No.	Parameters	Walkthrough	Inspection
1.	Focus	Improve product	Find defects
2.	Activities	• Find defects. • Examine alternatives. • Forum for learning. • Discussion.	• Find defects. • Only defect explanation allowed. • Learning through defects and inspection.
3.	Process	Informal	Formal
4.	Quality	Variable; personalities can modify outcome.	Repeatable with fixed process.
5.	Time	Preparation ad-hoc, less formal.	Preparation required, efficient use of time.

3.2.6 Technical Review

- A technical review involves reviewing the technical approach used during the development process. It is more of a peer review activity and less formal as compared to audit and inspection.
- In technical review, the product is examined and the defects are found which are mainly technical ones.

- No management participation is there in technical review. The full report is prepared to have a list of issues addressed.
- Defects are found by the experts (such as architects, designers, key users) who focus on the content of the document.
- In practice, technical reviews vary from quite informal to very formal.
- **The goals of the technical review are :**
  1. To ensure that an early stage the technical concepts are used correctly
  2. To access the value of technical concepts and alternatives in the product
  3. To have consistency in the use and representation of technical concepts
  4. To inform participants about the technical content of the document.

3.3 Dynamic Techniques

- Dynamic testing can take place only after compilation and linking. Dynamic testing detects fewer errors than static testing, but it does detect some that static testing misses.
- If timescales are tight, use of dynamic testing tools might be omitted, but tool-supported static testing should never be omitted. Dynamic testing takes place when the program itself is used for the first time.
- **Dynamic testing** is a term used to describe the testing of the dynamic behavior of code.
- Dynamic analysis refers to the examination of the physical response from the system to variables that are not constant and change with time.
- The software must actually be compiled and run;
- Dynamic testing involves working with the software, giving input values and checking if the output is as expected.
- These are the validation activities. unit tests, integration tests, system tests and acceptance tests are few of the dynamic testing methodologies. Dynamic testing means testing based on specific test cases by execution of the test object or running programs.

3.3.1 Difference between Static and Dynamic Testing

Sr. No.	Static Testing	Dynamic Testing
1.	Static testing is a form of software testing where the software isn't actually used.	In dynamic testing the software must actually be compiled and run.

2.	It is generally not detailed testing, but checks mainly for the sanity of the code, algorithm, or document. It is primarily syntax checking of the code or and manually reading of the code or document to find errors.	Dynamic analysis refers to the examination of the physical response from the system to variables that are not constant and change with time.
3.	This type of testing can be used by the developer who wrote the code, in isolation. Code reviews, inspections and walkthroughs are also used.	Some of dynamic testing methodologies include unit testing, integration testing, system testing and acceptance testing.
4.	This is the **verification** portion of verification and validation.	Dynamic testing is the **validation** portion of verification and validation.
5.	Reviews, walkthroughs, or inspections are considered as static testing.	Unit tests, integration tests, system tests and acceptance tests are types of the dynamic testing.

3.4 Structural Testing Techniques

SPPU : Dec.-19

- This type of testing is based on the code's structure. Structural testing ensures sufficient checking of artifacts including requirements, architectures, design and implementation, by finding test data that will force sufficient coverage in terms of statements, conditions and paths.
- For example, if a code is meant to calculate the average of even numbers in an array, then structure-based testing would be interested in the 'steps that lead to the average being calculated', rather than whether the final output is a correct numerical value.
- Suppose we have to check whether we have defined the code that differentiates even numbers from odd numbers. We may have a conditional statement here, like, if an array element is divisible by two without a remainder, if (arr[i] % 2 === 0) then the number can be said as an even number.
- Structural testing is carried out by the same people who write the code as they understand it best.
- Some of the test tools that are used for white box or structural testing are JBehave, Cucumber, Junit and Cfix.
- Structural testing is divided into four different categories : Mutation testing, data flow testing, control flow testing and slice-based testing.

3.4.1 Structural Testing at System Level

- Structural system testing aims to verify whether the developed system or program works correctly or not. This testing technique is implemented for ensuring accuracy/correctness of the design and functionality of the system.
- Structural system testing process is to determine that the technology used to build the product has been properly implemented. Also this technique aids in verifying that all components work as a single unit. The purpose of structural system testing is to ensure that it is designed correctly.
- Data for system testing needs to be generated using requirements/design specification or one may use client supplied data. Client supplied data is the best options as it is a live data but may not be feasible in all conditions.
- Structural testing can thus be implemented at all levels or at any phase throughout the software development life cycle.

3.4.2 Advantages and Disadvantages of Structural Testing

1. Advantages of structural testing

- The logic of the software's structure can be tested.
- Provides a more thorough testing of the software.
- Helps finding out defects at an early stage.
- Helps in eliminating dead code.
- Not time consuming as it is mostly automated.

2. Disadvantages of structural testing

- Its tests do not ensure that user requirements have been met.
- Its tests may not mimic real-world situations.
- Requires knowledge of the code.
- Requires training in the tool used for testing.
- It is expensive.

3.4.3 Mutation Testing

- Mutation testing is a structural testing method, i.e. we use the structure of the code to guide the test process. A mutation is a small change in a program. Such small changes are intended to model low level defects that arise in the process of coding systems. Ideally mutations should model low-level defect creation.
- Mutation testing is used to check the capability of test program and test cases to find defects. Test cases are designed and executed to find defects.

- The original test data are then run through the mutants. If test data detect all differences in mutants, then the mutants are said to be dead and the test set is adequate.

Original Program	Mutant

int index = 0;
while (...)
{
...;
index++;
if (index==10)
break;
}
	int index = 0;
	while (...)
	{
	...;
	index++;
	if (index>=10)
	break;
	}

Kinds of Mutation

1. **Value Mutations :** These mutations involve changing the values of constants or parameters (by adding or subtracting values etc), e.g. loop bounds, being one out on the start or finish is a very common error.
2. **Decision Mutations :** This involves modifying conditions to reflect potential slips and errors in the coding of conditions in programs, e.g. a typical mutation might be replacing a > by a < in a comparison.
3. **Statement Mutations :** These might involve deleting certain lines to reflect omissions in coding or swapping the order of lines of code. There are other operations, e.g. changing operations in arithmetic expressions. A typical omission might be to omit the increment on some variable in a while loop.

3.4.4 Statement Coverage Testing

- Statement coverage is a white box testing technique. It involves the execution of all the statements at least once in the source code. It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed.
- It aims to test all the statements present in the program. It is a good measure of testing each part in terms of statements but it is not a good technique for testing the control flow. For example : One node can be reached from multiple other nodes, but since the aim is to cover just the statements, those conditions are not covered.
- Statement coverage testing is put into practice in the software development life cycle as a part of code validation, during the build and development phases.

- Statement coverage statement is calculated as follows :

$$\text{Statement coverage (\%)} = \frac{\text{Number of statements executed as a part application's code}}{\text{Total number of statements in the application's source code}} \times 100$$

- Where the total number of statements represents the mentions from the requirement document and the number of statements executed represents the statements present in the source code.

3.4.5 Branch Coverage Testing

- Branch coverage is a requirement that, for each branch in the program (e.g., if statements, loops), each branch have been executed at least once during testing.
- Branch coverage is a metric that indicates whether all branches in a code base are exercised by tests. A "branch" is one of the possible execution paths the code can take after a decision statement, e.g., an if statement gets evaluated.
- Branch coverage is also known as decision coverage testing. It aims to test all the branches or edges at least once in the test suite or to test each branch from a decision point at least once. It provides solution for the problem faced in statement coverage
- Branch coverage testing helps in validating of all the branches in the code and making sure that no branching leads to abnormal behavior of the application.
- Branch coverage is an important metric in that it can help a team or organization assess whether an application has been tested to completion. A low branch coverage shows that there are scenarios in the application lacking testing. Such scenarios might contain defects that will only manifest in edge cases when the application makes it to production.
- For example, executing an "if ..... then" statement (no else) when the tested condition is true, tests only one of two branches in the flow graph. Branch testing seeks to ensure that every branch has been executed.

Pros of branch coverage testing :

a) To approve that all the branches in the code are reached.

b) To guarantee that no branches prompt any irregularity of the program's operation.

c) It kills issues that happen with statement coverage testing.

3.4.6 Path Coverage Testing

- It aims to test the different path from entry to the exit of the program, which is a combination of different decisions taken in the sequence. The paths can be too many to be considered for testing, for example, a loop can go on and on. It can be avoided using cyclomatic complexity, which help in finding out the redundant test cases.
- In this the test case is executed in such a way that every path is executed at least once.
- All possible control paths taken, including all loop paths taken zero, once and multiple (ideally, maximum) items in path coverage technique, the test cases are prepared based on the logical complexity measure of a procedural design. In this type of testing every statement in the program is guaranteed to be executed at least one time.
- Flow graph, cyclomatic complexity and graph metrics are used to arrive at basis path.
- Testing all paths does not mean that programmer will find all bugs in a program. Many bugs arise because programmers have forgotten to include some processing in their code, so there are no paths to execute. Some bugs are also related to the order in which code segments are executed.
- For example, if segments a, b and c are executed <a, b, c>, then the program may work properly. However, if they are executed <a, c, b> then an error may arise. It is practically impossible to test all orders of processing as well as all program paths.
- The objective of path testing is to ensure that each independent path through the program is executed at least once. An independent program path is one that traverses at least one new edge in the flow graph. In program terms, this means exercising one or more new conditions. Both the true and false branches of all conditions must be executed.

3.4.7 Conditional Coverage Testing

- Condition coverage aims to test individual conditions with possible different combination of Boolean input for the expression.
- With condition coverage the possible outcomes of ("true" or "false") for each condition are tested at least once. This means that each individual condition is one time true and false.

- In other words we cover all conditions, hence condition coverage. The outcome of the decision point is only relevant for checking the conditions. Also the combinations of conditions are not relevant

3.4.8 Loop Coverage Testing

- Loop testing is an example of white-box testing. This technic is used to test software loops and it is one type of control structure testing.
- The goal of loop testing is to test while-do, do-while and any other loops in a program thoroughly - By trying to ensure that each is executed at minimal, typical, and maximal values - and to try to "break" the program, by trying to have a loop executed with a fewer than minimum, as well as a larger than maximal, number of iterations.
- **Loop test is performed because of following reasons :**
  a) Testing can help to resolve loop repetition observation.
  b) Loop testing can estimate conditions in performance and facility.
  c) The loop's uninitialized variables can be detected by testing loops.
  d) It aids in the classification of loop starting issues.
  e) Testing can return the loop to routine issues.
  f) Loops testing can reveal production/capability bottlenecks.
  g) It serves to classify the loop's beginning queries.
- **Three types of loops will be tested :**
  a) Simple loops are loops whose loop bodies contain no other loops.
  b) Nested loops are combinations of loops such that each is contained inside the loop body of the next.
  c) Concatenated loops are loops such that each follows the next in the code - That is, the execution of the next loop begins after the previous terminates.

eview Question

1. *What is mutation testing ? Give an example.* **SPPU : Dec.-19, End Sem, Marks 3**

.5 Black Box Techniques

- Here we discuss some of the black box testing techniques.
- Test cases are based on the specification of the item's external behavior.
- Analysis of the input/output domain of the program : Leads to a logical partitioning of the input/output domain into 'interesting' subsets.

- Analysis of the observable black-box behaviour : Leads to a flow-graph-like model which enables application of techniques from the white-box world.
- Test design technique is a process to identify few test cases out of many with the likelihood of identifying defects. It helps to achieve high test coverage.
- **Types of black box testing techniques are,**
  1) Equivalence classes
  2) Boundary value analysis
  3) Decision table testing
  4) State transition testing
  5) Use case testing.
- A black box test passes input to a system, records the actual output and compares it to the expected output.

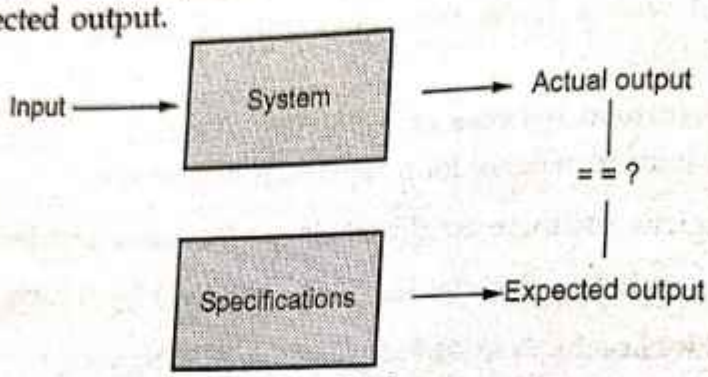

Fig. 3.5.1

Result :

- If actual output == expected output
  - Test passed
- else
  - Test failed

3.5.1 Boundary Value Analysis

- Boundary value is defined as a data value that corresponds to a minimum or maximum input, internal, or output value specified for a system or component. Programmers often make mistakes on the boundaries of the equivalence classes/input domain.
- Testing boundary conditions of equivalence classes is more effective i.e. values directly on, above and beneath edges of equivalence classes. Choose input boundary values as tests in input equivalence classes instead of, or additional to arbitrary values. Choose also inputs that invoke output boundary values.
- **Example strategy as extension of equivalence partitioning :**
  1. Choose one (n) arbitrary value in each equivalence class
  2. Choose values exactly on lower and upper boundaries of equivalence class
  3. Choose values immediately below and above each boundary (if applicable)

3.5.2 Equivalence Class Partition

- It reduce the number of test cases that need to be developed. We don't want to write several test cases that test the same aspect of our program.
- Equivalence partitioning divides the input domain of a program into classes. For each of these equivalence classes, the set of data should be treated the same by the module under test and should produce the same answer. Test cases should be designed so the inputs lie within these equivalence classes.
- **Example :** Test a program that computes the sum of the first value integers as long as this sum is less than maxint. Otherwise an error should be reported. If value is negative, then it takes the absolute value. Given integer inputs maxint and value compute result,

$$\text{Result} = \sum_{k=0}^{|value|} k$$

if this <= maxint, error otherwise

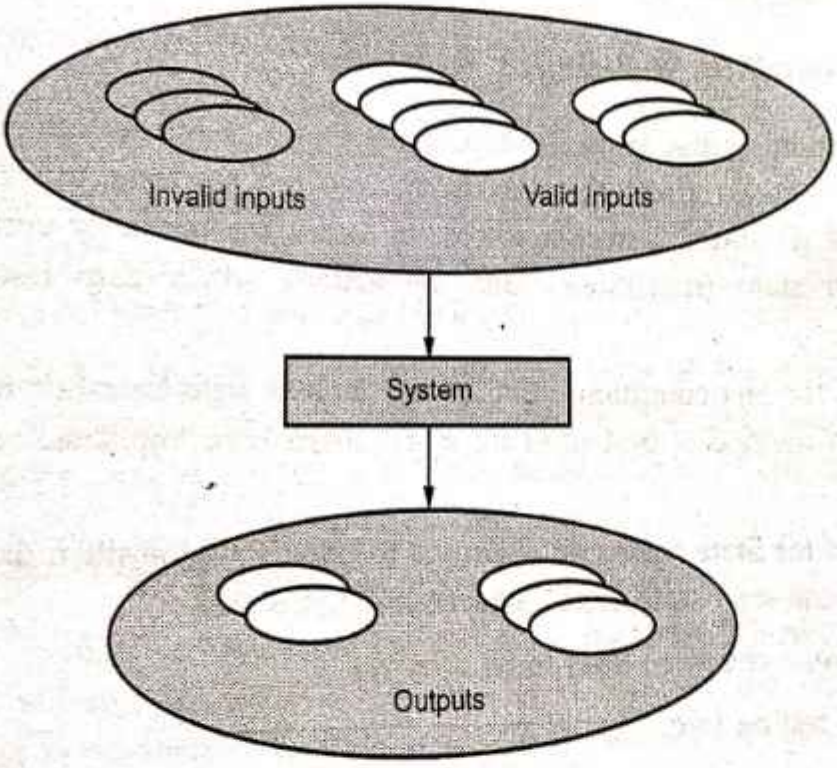

Fig. 3.5.2

Equivalence classes :

Condition	Valid equivalence classes	Invalid equivalence classes
Number of inputs	2	< 2, > 2
Type of input	int int	int no-int, no-int int

Absolute value	Value < 0, value ≥ 0
maxint	maxint Σk ≤ maxint, Σk > maxint

Test Cases :

	maxint	value	result
valid	100	10	55
	100	-10	55
	10	10	error
invalid	10	-	error
	10, 20	30	error
	"XYZ"	10	error
	100	9.1E4	error

3.5.3 State Transition Technique

- State transition testing is used where some aspect of the system can be described in what is called a **'finite state machine'**. It allows the tester to view the software in terms of its states, transitions between states, the inputs or events that trigger changes in state (transitions) and the actions which may result from those transitions.
- Identifying the test conditions from a state table is state transition testing. This test case design method is best used for applications with implanted workflow within them.
- The process for State transition testing is to draw state transition diagram,
  i. Determine start state, input, output and finish state
  ii. Determine coverage level to be achieved
  iii. Draw testing tree
  iv. Define tests.

3.5.4 Cause Effective Graph

- Black-box testing technique to analyze combinations of input condition. Make Boolean graph linking causes and effects. Annotate impossible combinations of causes and effects.

- Develop decision table from graph with in each column a particular combination of inputs and outputs. Transform each column into test case.
- Systematic method for generating test cases representing combinations of conditions. Combinatorial explosion of number of possible combinations. Some heuristics to reduce this combinatorial explosion. Starting point is effects outputs then working 'backwards'

3.5.5 Decision Table

- Decision table is a tabular representation of inputs versus rules/cases/test conditions.
- Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form.
- Many systems provide outputs based on a set of conditions. Testers can then identify "rules" which are a combination of conditions, identify the outcome of each rule and design a test case for each rule.
- For example, a health insurance company may provide different premium based on the age of the insured person (under 50 or over 50) and whether they are a drinker or not. This generates a decision table with rules and outcomes.
- Decision table technique is appropriate for the functions that have a logical relationship between two and more than two inputs.
- This technique is related to the correct combination of inputs and determines the result of various combinations of input. To design the test cases by decision table technique, we need to consider conditions as input and actions as output.

3.5.6 Use Case Testing

- Use case testing is a technique that helps to understand business conditions and flows. It is used to identify test cases that exercise the whole system on a transaction by transaction basis from start to finish.
- Use case testing describes interactions between actors, including users and the system, which produce a result of value to a system user. Use cases are a sequence of steps that describe the interactions between the actor and the system.
- The method of deriving the test condition from use case is known as **use case testing.** The test cases designed as per this type of method execute the different business scenarios and user functionalities.

- Different use cases are used in use case testing. Use case is a document which describes the end to end behavior of the system. A use case is a sequence of actions that an actor performs within a system to achieve a particular goal
- Use cases are a type of process flow model, so they are similar in concept, to classic flow models like data flow diagrams and cause-effect graphs.
- Use case testing offers a new perspective, and identifies test cases which other techniques have difficulty seeing. The value of use cases is that they focus attention on the user, rather than on the actions the system performs.
- Doing use case testing, we do and test the end to end flow of all the transactions (positive test case) and also cover the alternate flows (negative test case) based on the user action and system action. User action is the steps which the user does (Request) and system action is the output of user action (response).

3.6 Experienced Based Techniques

- It is the technique of executing testing activities with the help of experience gained through several years of churning. Basically, a tester verifies and validates the software product quality using his/her past experience of testing the similar type of product in the respective domain.
- In experience-based techniques, people's knowledge, skills and background are of prime importance to the test conditions and test cases.
- Experience-based techniques go together with specification-based and structure-based techniques and are also used when there is no specification, or if the specification is inadequate or out of date.
- When experience based resting is required :
  a) Non-availability of requirements and specifications.
  b) Limited knowledge of the software product.
  c) Inadequate specification
  d) Restricted amount of time, to perform testing.
- Types of experienced based testing are error guessing and exploratory testing. This technique is used for low risk system. This kind of testing is done even when there is no specifications or have inadequate specification list.

1 Error Guessing

In error guessing, tester applies his/her experience to guess the areas in the application that are prone to error.

- Error guessing is a technique that makes use of tester's skills, intuition and experience to anticipate the occurrence of errors, defects and failures that may not be easily captured by formal techniques like boundary value analysis and equivalence partitioning.
- The success of error guessing is very much dependent on the skill of the tester, as good testers know where the defects are most likely to be.
- Every application has a history, which is based on the people that are working on it and also the functionality or integration points that the application has. An experienced tester would know the type of errors that have occurred in the past.
- For example, there is an e-commerce application where the discount logic has often failed with the application of a random discount code. In such cases, the tester will try different combinations of valid and invalid discounts and see that the logic is working fine. This insight comes only when the tester has enough experience in the application to know the historical issues.

3.6.2 Exploratory Testing

- Exploratory testing is a testing technique and simultaneously, a progressive learning approach to perform maximum testing with the minimal planning.
- During the course of exploratory testing, a tester constantly studies and analyzes the software product and accordingly applies his/her skills, traits and experience to develop strategy and test cases to perform and carry out the necessary testing.
- Exploratory tests are all about testers exploring an application to identify and document potential bugs. Testers embark on a process of investigation and discovery in order to effectively test a product. That's what exploratory testing is all about.
- Exploratory testing is often used in Agile models because it links up with the concepts of discovering, investigating and acquiring a greater understanding of the code undergoing testing.
- Exploratory testing is valuable to perform at the point when new testers join the team, as they bring their knowledge and expertise as well as a fresh perspective to use as a lens for viewing the process.
- Exploratory testing best use may be seen in the event of inadequate specifications and requirement and severely limited time.

3.7 Levels of Testing

- Software design uses divide and conquer method. Software is divided into units (components) in design phase. Development of software becomes easy. This rule

is applicable to software testing also. Instead of testing whole system, unit-wise testing method is used.

- A practical relationship exists between levels of testing versus specification based and code based testing.
- Fig. 3.7.1 shows level of testing.

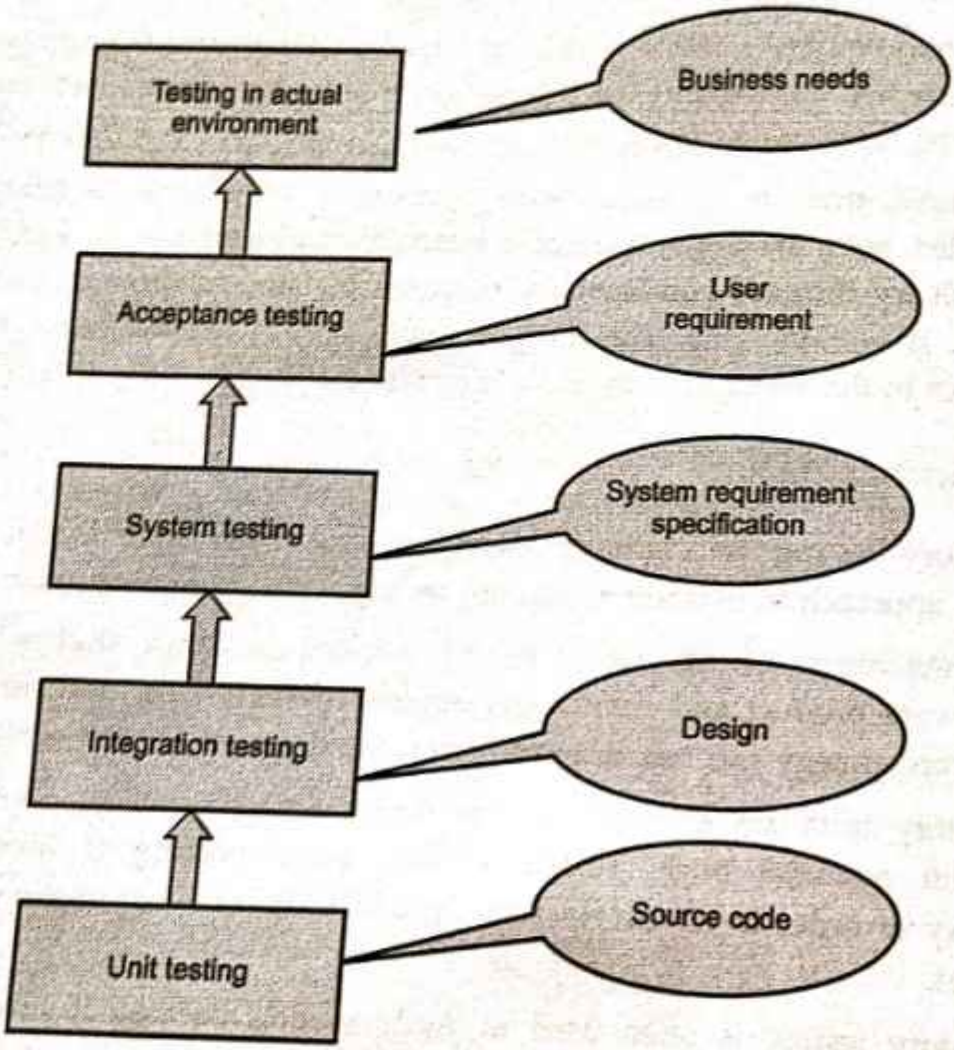

Fig. 3.7.1 Level of testing

- It is necessary to divide the testing process into different levels and each level is tested differently. Software project is divided into different modules and each modules are testing separately. After testing all modules, combines together and system is built and again tested.
- **Unit testing :** Checking the behavior of single modules.
- **Integration testing :** Checking the behavior of module cooperation.
- **Acceptance testing :** The software behavior is compared with end user requirements.

- **System testing :** The software behavior is compared with the requirements specifications.
- **Regression testing :** To check the behavior of new releases.

3.7.1 Functional Testing

- Functional testing addresses the overall behavior of the program by testing transaction flows, input validation and functional completeness. Functional testing often includes testing portions of the underlying code.
- Functional testing is considered black-box testing because no knowledge of the internal logic of the system is used to develop test cases. System testing, regression testing and user acceptance testing are types of functional testing.
- Both methods together validate the entire system. For example, a functional test case might be taken from the documentation description of how to perform a certain function, such as accepting bar code input.
- Comparing actual outputs against expected behaviors provides a clearer overall picture than testing individual modules in isolation. Interactions between modules are frequently the points where errors occur.

1. Advantages of Functional Testing

- Simulates actual system usage.
- Makes no system structure assumptions.

2. Disadvantages of Functional Testing

- Potential of missing logical errors in software.
- Possibility of redundant testing.

3.7.2 Unit Testing

- A unit is smallest testable piece of software. Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed.
- Unit testing aims at testing each of the components that a system are built upon. As long as each of them work as they are defined to, then the system as a whole has a better chance of working together.
- Unit testing is the first level of testing and is performed prior to integration Testing. Unit testing is normally performed by software developers themselves.

Advantages :

1. Codes are more reusable
2. Unit testing increases confidence in changing/ maintaining code
3. Development is faster
4. Debugging is easy
5. The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.

3.7.3 Integration Testing

- Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.
- The entire system is viewed as a collection of subsystems determined during the system and object design. The order in which the subsystems are selected for testing and integration determines the testing strategy.
- **Example :** Three layer call hierarchy

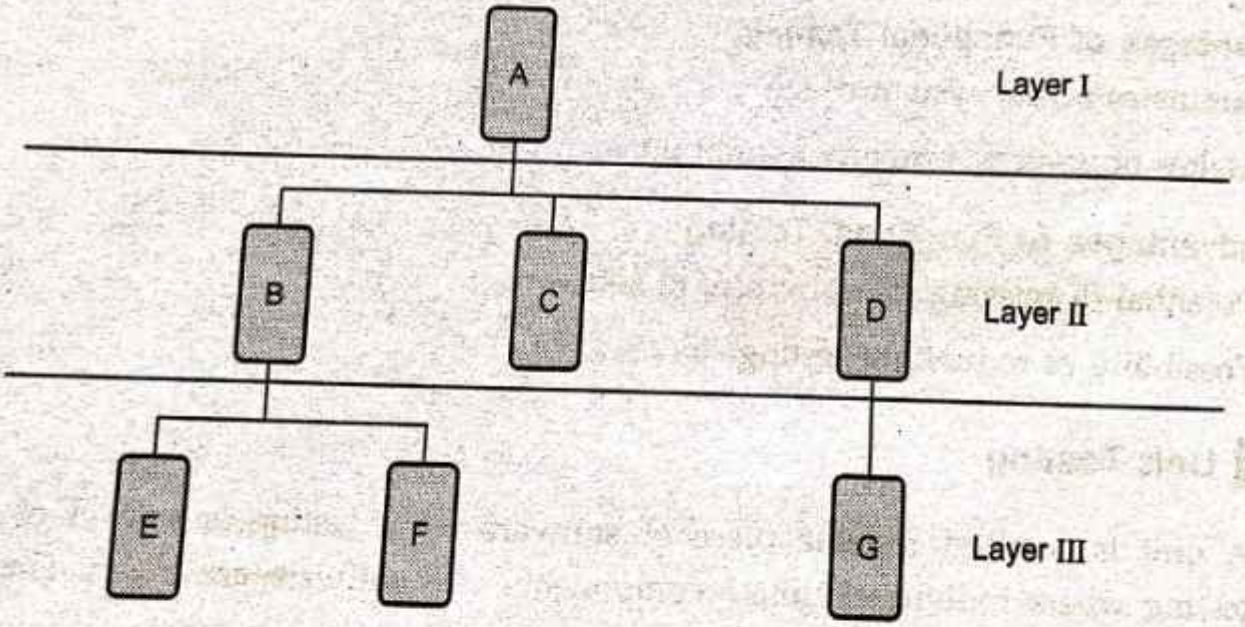

Fig. 3.7.2

1. Based on the integration strategy, select a component to be tested. Unit test all the classes in the component.
2. Put selected **component** together; do any preliminary fix-up necessary to make the integration test operational (drivers, stubs)
3. Do functional testing : Define test cases that exercise all **uses cases** with the selected component.

4. Do structural testing : Define test cases that exercise the selected component.
5. Execute performance tests.
6. Keep records of the test cases and testing activities.
7. Repeat steps 1 to 7 until the full system is tested.

- The primary goal of integration testing is to identify errors in the (current) component configuration.
- Different approaches to integration testing are bottom-up, top-down, big-bang and sandwich.
- Who does integration testing and when is it done ?
  1. Done by developers/testers
  2. Test cases written when detailed specification is ready
  3. Test continuous throughout project.
- Where is it done ? Done on programmer's workbench.
- Why is it done ? Discover inconsistencies in the combination of units.

3.7.4 System Testing

- **System testing :** The process of testing an integrated system to verify that it meets specified requirements.
- Test of overall interaction of components. Find disparities between implementation and specification.
- System testing is performed after integration testing and before acceptance testing. Normally, independent testers perform system testing.
- System testing has to be tested for its non-functional requirements, such as performance requirement, portability, security, safety, interoperability of hardware/software.
- Identifies new faults that may have been introduced as current one are being corrected. Verifies a new version or release still performs the same functions in the same manner as an older version or release.

3.7.5 Acceptance Testing

- Acceptance testing is a formal testing conducted to determine whether a system satisfies its acceptance criteria. There are two categories of acceptance testing : User acceptance testing and business acceptance testing.

- User acceptance testing is conducted by the customer to ensure that system satisfies the contractual acceptance criteria before being signed-off as meeting user needs.
- Business acceptance testing is undertaken within the development organization of the supplier to ensure that the system will eventually pass the user acceptance testing.
- Acceptance testing can be done either at the client office or developers office.
- **Three major objectives of acceptance testing :**
  1. Confirm that the system meets the agreed upon criteria
  2. Identify and resolve discrepancies, if there is any
  3. Determine the readiness of the system for cut-over to live operations.
- Acceptance tests are usually created by business customers and expressed in a business domain language. These are high level tests to test the completeness of a user story or stories 'played' during any sprint/iteration. These tests are created ideally through collaboration between business customers, business analysts, testers and developers, however the business customers (product owners) are the primary owners of these tests.
- As the user stories pass their acceptance criteria, the business owners can be sure of the fact that the developers are progressing in the right direction about how the application was envisaged to work and so it's essential that these tests include both business logic tests as well as UI validation elements.

3.7.6 Sanity/Smoke Testing

- Software components already translated into code are integrated into a build. A series of tests designed to expose errors that will keep the build from performing its functions are created.
- The build is integrated with the other builds and the entire product is smoke tested daily using either top-down or bottom integration.
- Smoke testing, also known as **"build verification testing"**, is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work.
- The purpose of smoke testing is to determine whether the new software build is stable or not so that the build could be used for detailed testing by the QA team and further work by the development team.
- Smoke testing is generally done by the QA team but in certain situations, can be done by the development team. Smoke testing is usually documented or scripted. This testing exercises the entire system from end to end.

3.7.7 Regression Test

- Testing activities occur after software changes. Regression testing usually refers to testing activities during software maintenance phase.
- Software regression test models are needed to support the definition of software regression test strategy, test cases and coverage criteria.
- Also referred to as verification testing, regression testing is initiated after a designer has attempted to fix a recognized problem or has added source code to a program that may have inadvertently introduced errors. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.
- Regression testing can be performed during any level of testing (Unit, integration, system, or acceptance) but it is mostly relevant during system testing.
- During regression testing, new test cases are not created but previously created test cases are re-executed.

3.8 Non-functional Testing

- Non-functional testing is done to verify the non-functional requirement of the application like performance, usability, etc. non-functional testing assesses application properties that aren't critical to functionality but contribute to the end-user experience.
- Performance and reliability under load aren't functional components of a software system but can certainly make or break the user experience. Something that fails a non-functional test doesn't always cause an issue that users would notice, but it can indicate a problem in the system especially at scale.
- **Advantages of a non-functional testing :**
  1. It covers the testing which cannot be covered in functional testing.
  2. It ensures that the application runs efficiently and is reliable enough.
  3. It ensures the security of the application.
- **Disadvantages of non-functional testing :**
  1. Every time the software is updated, non-functional tests are performed again.
  2. Due to software updates, user have to pay to re-examine the software; thus software becomes very expensive.

3.8.1 Performance Testing

- Performance testing is conducted after the completion of functional testing. It is usually conducted for web applications. Main objective of performance testing is to get information with respect to response time, throughput and utilization under a given load.
- Performance testing is normally used with load testing and stress testing.
- Poor latency destroys a user's experience and well-written performance tests often catch problems before they become evident to users.
- Performance testing is the testing technique used to determine the performance components of a particular system/application in a specific situation. It determines how much resources are used and how much product/system is scalable, reliable, responsible and speedy during test.
- This testing does not give pass or fail result, used to set the benchmark and standard of the application/system against concurrency/throughput, server response time, latency, render response time etc.
- This testing is the subset of performance engineering and it is used to determine the performance issues come in the design and architecture of software product.
- It is very extensive, contains : Load testing, stress testing, capacity testing, volume testing, endurance testing, spike testing, scalability testing and reliability testing etc.
- **Performance testing :** This measures the response time of an application with an expected number of users. The aim of this is to get a baseline and an indication of how an application behaves under normal conditions. Does it meet the required response time or not ?
- **Load testing :** This measures the response time when the application is subjected to more than usual load. The application will be slower under heavy load, but the aim of load testing is to see whether the application can sustain the increased load on the server or will it crash and kill the servers.
- Load is more about characterizing / simulating your actual workload. Load testing is usually started as low numbers and gradually increased over a given period of time until it reaches the desired load on the system and then it ramps down.
- The goal of a load test is to prove that a system can handle the expected volume with minimal to acceptable performance degradation. A load test ensures that a web system is capable of handling an expected volume of traffic and therefore is sometimes referred to as volume testing.

3.8.2 Memory Test

- The purpose of a memory test is to confirm that each storage location in a memory device is working.
- Memory leak is an unmanaged application refers to memory, or pointers to memory addresses, becoming unreachable or unusable once released (orphaned), but it more commonly is used to talk about many types of memory-centric problems.
- A memory leak occurs when program doesn't free up memory that is used.
- Catastrophic Failure is a memory problem that occurs due to physical and electrical damage, it is uncommon and easily detectable.
- Stuck bits : Sometimes specific bits in memory modules can get stuck to a 1 or 0. This is obviously a problem as when the memory needs to write the opposite to that space, it won't change.

3.8.3 Scalability Testing

- In scalability testing, the performance of a software application, system, network or process is tested in terms of its capability to scale up or scale down the number of user request load or other such performance attributes.
- It can be carried out at a hardware, software or database level. Scalability testing is defined as the ability of a network, system, application, product or a process to perform the function correctly when changes are made in the size or volume of the system to meet a growing need.
- It ensures that a software product can manage the scheduled increase in user traffic, data volume, transaction counts frequency and many other things. It tests the system, processes or databases ability to meet a growing need.
- The objective of scalability testing is to determine the software application's effectiveness in scaling up to support an increase in user load.
- Scalability testing is to measure at what point the software product or the system stops scaling and identify the reason behind it. The parameters used for this testing differs from one application to another.
- Reasons for using the scalability testing is as follows :
  1. In case of any modification in the software lead us to its failure ?
  2. After the enhancement, the software is working correctly and efficiently in meeting the user requirements and expectations.
  3. Whether the software can produce and improve as per extended needs ?

- For example, an e-commerce site may be able to handle orders for up to 1000 users at a time but scalability testing can be performed to check if it will be able to handle higher loads during peak shopping seasons.

3.8.4 Compatibility Testing

- Compatibility testing is a type of software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or mobile devices.
- Evaluates that the application is compatible with other hardware/software with minimum and maximum configuration.
- Why compatibility testing is important ?
  1. It ensures complete customer satisfaction.
  2. It provides service across multiple platforms.
  3. Identifying bugs during development process.
- Compatibility testing are of two types : Backward compatibility and forward compatibility
  1. **Backward compatibility testing** is a technique to verify the behavior and compatibility of the developed hardware or software with their older versions of the hardware or software. Backward compatibility testing is much predictable as all the changes from the previous versions are known.
  2. **Forward compatibility testing** is a process to verify the behavior and compatibility of the developed hardware or software with the newer versions of the hardware or software. Forward compatibility testing is a bit hard to predict as the changes that will be made in the newer versions are not known.

3.8.5 Security Testing

- Security testing is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders. Security testing depends heavily on expertise and experience.
- Software security is about making software behave correctly in the presence of a malicious attack. The difference between software safety and software security is therefore the presence of an intelligent adversary bent on breaking the system.
- Different types of security testing are as follows :
  1. **Network security** : This involves looking for vulnerabilities in the network infrastructure (resources and policies).
  2. **System software security** : This involves assessing weaknesses in the various software (OS and database system) the application depends on.

3. **Client-side application security** : This deals with ensuring that the client cannot be manipulated.
4. **Server-side application security** : This involves making sure that the server code and its technologies are robust enough to fend off any intrusion.

3.8.6 Cookies Testing

- Cookie is a file used to store the session content and other session related information inside browser. Some sites use cookies for payment gateway and some sites use it for authentication session.
- Cookie testing is performed to test the cookies that are saved in the web browser whenever a web application is being accessed by making use of its server.
- Cookie testing is all about testing the cookies that present at client side in local machines or computers or browsers. Before going through the methods of cookie testing, a basic understanding of the cookie would be useful in clearly understanding the concept of cookie testing.
- **Session cookies** : This type of cookie gets active on the call made by the web-browser and automatically gets deleted on the termination of the session by that web-browser.
- **Persistent cookies** : These cookies reside permanently on the user's machine and have a specific time-period for the expiration, which may last for few months or year.

3.8.7 Session Testing

- Session-based test management is an effective way to manage testing by focusing on activities testers perform in test sessions. Testing activity is broken up into three task categories : Test execution, bug investigation and reporting and setup and administration.
- A big aspect of session-based test management is that testers have the freedom to add and remove charters as needed to be successful.
- It is very similar to that of exploratory testing, except the tests, are conducted during time-boxed sessions. Session-based testing combines exploratory testing, accountability and control.
- This method helps testers to structure the tests and perform them during uninterrupted sessions. In this approach, testers create test charters.
- This charter contains details like what to test and test reports which allow testers to document what they discover during a test. It will, in turn, help in getting rid of hidden bugs and defects in the product.

3.8.8 Recovery Testing

- Recovery is the ability to restart operations after the integrity of the application has been lost. The process normally involves reverting to a point where the integrity of the system is known and then reprocessing transactions up until the point of failure.
- Recovery testing is non-functional testing that determines the capability of the software to recover from failures such as software/hardware crashes or any network failures.
- **Objective of recovery testing are as follows :**
  a) Adequate backup data is preserved.
  b) Backup data is stored in a secure location.
  c) Recovery procedures are documented.
  d) Recovery personnel have been assigned and trained.
  e) Recovery tools have been developed.
- Recovery test is conducted in two modes. They should assess the procedures, methods, tools and techniques. Then, after the system has been developed, they should introduce a failure into the system and test the ability to recover.
- **Example :** Restart the system while a browser has a definite number of sessions and check whether the browser is able to recover all of them or not.

1. Advantages of recovery testing

- To detect bugs and fix them. This improves the system's quality.
- Recovery testing also helps in eliminating unnecessary risks
- Performance-related issues can be easily identified through recovery testing.
- It helps in stabilizing the system/application.
- In case of failure, backup data is always maintained.

2. Disadvantages of recovery testing

- It is a time-consuming process.
- There is a need for a skilled person who knows the system inside out.
- There are few cases where the potential risks and flaws are uncertain.
- Not all bugs can be found even after rigorous testing.
- Recovery testing is expensive to perform

3.8.9 Installation Testing

- It is a type of quality assurance work in the software industry that converges on what customers will need to do to install and set up the new software successfully.
- It checks if the software application is successfully installed and is working as expected. The testing process may involve full, partial or upgrades install/uninstall processes.
- Installation testing is performed to check if the software has been correctly installed with all the inherent features and that the product is working as per expectations. Also known as implementation testing, it is done in the last phase of testing before the end user has his/her first interaction with the product.

3.8.10 Adhoc Testing

- Ad hoc testing is unstructured, random and requires no documentation. Ad hoc testing is also referred to as random testing and monkey testing.
- **Characteristics :**
  1. Random testing, done on any part of the application.
  2. It is a black box testing technique.
  3. It detect uncover bugs and loopholes that may be missed during automated testing.
  4. Requires a software tester with in-depth knowledge of the testing system
  5. It is unplanned and unstructured.
  6. It doesn't require any documentation or formal test case execution.
- **Types of Adhoc testing :**

1. Buddy Testing

- Buddy testing is done with at least two members. One member is from the testing team, and another one is from the development team.
- When the unit testing is performed on the application, then only we can perform buddy testing. This type of testing helps the developer team and testing team to do their jobs.

2. Pair Testing

- In this testing, two testers work together on a module with the same test setup shared between them. One tester can perform testing and the other tester can observe and analyze the testing process. Both can share the work of testing and make necessary documentation of all observations made.

Advantages of Adhoc Testing

1. Adhoc testing cannot follow any process.
2. Tester can find the number of defects than in traditional testing because of the various innovative methods they can apply to test the software.
3. When the in-depth testing is required in less time, adhoc testing can be performed and also deliver the quality product on time.

Disadvantages of Adhoc Testing

1. Adhoc testing is dependent on the test engineer's product knowledge because he/she knows the flow of the application.
2. We cannot trace the requirements or check requirement/test coverage, as there is no traceability matrix or any documentation for that matter.

3.8.11 Difference between Adhoc Testing and Exploratory Testing

Sr. No.	Adhoc testing	Exploratory testing
1.	Adhoc testing is performed without planning or goal.	Exploratory testing is performed with clear plan and structured.
2.	A tester needs to have complete knowledge of the software prior to working on it.	There is no such compulsion to have prior knowledge of the program. The tester can do testing with or without any knowledge of the subject under test.
3.	There is no target or goal of the process, everything is done randomly, i. e. without any structure.	In this, targets and goals can be set while still giving the tester the ability to explore the application and think freely.
4.	There is no need for any documentation, none of the steps performed are documented.	Documentation or recording of the action performed can be done in exploratory testing. So, the bugs identified can be easily reproduced.

3.8.12 Risk Based Testing

- Risk-based testing is an approach to software testing that relies on risk assessments to determine the best allocation of testing resources. Risk is the probability of occurrence of an undesirable outcome. This outcome is also associated with an impact.
- Risk-based testing is a ranking of tests and subtests, for functionality. Tools and techniques such as equivalence partitioning, state transition tables, decision tables, boundary-value analysis, path flow testing, all-pairs testing etc. help assess the most risk-prone areas.

- Risk-based testing is done on a project at the very early stages. This helps identify the project risks that disclose the project quality. This information, in turn, leads to the testing of the planning, specifications, preparations and execution.
- **Processes involved in risk-based testing are as follows :**
  1. Identify every requirement with regard to the risks associated with the project.
  2. When it comes to assessing risks, prioritize the needs.
  3. Prepare and outline the tests based on requirement prioritization.
  4. Perform the test as per prioritization as well as approval criteria.
- **Advantages of risk based testing :**
  a) As all the critical functions of the application are tested, it improves the overall quality of the product.
  b) Early detection of the potential problem areas. Effective preventive measures can be taken to overcome these problems.
  c) Improved market opportunity (Time to market) and on time delivery.
  d) Improved service performance.
  e) Helps make testing a better planned and organized activity.

3.8.13 I18N Testing

- I18N is also called **Internationalization testing.** There are 18 characters between I and N. Internationalization testing is a part of globalization testing. The process of internationalization testing ensures that we develop applications that are compliant with languages and the user interface design is adjusted accordingly.
- Localization is defined as making a product, application or document content adaptable to meet the cultural, lingual and other requirements of a specific region or a local. In localization testing the tester checks to see if the application looks and behaves as expected after localizing the application.

Internationalization is the process of designing and developing a product, application or document content such that it enables localization for any given culture, region, or language.

The main objective is to perform the internationalization testing, and this testing concentrates on the multiple testing such as functional testing, integration testing, usability testing, user interface testing, compatibility testing, installation testing, and validation testing.

Fig. 3.8.1 shows architecture of Internationalization testing.

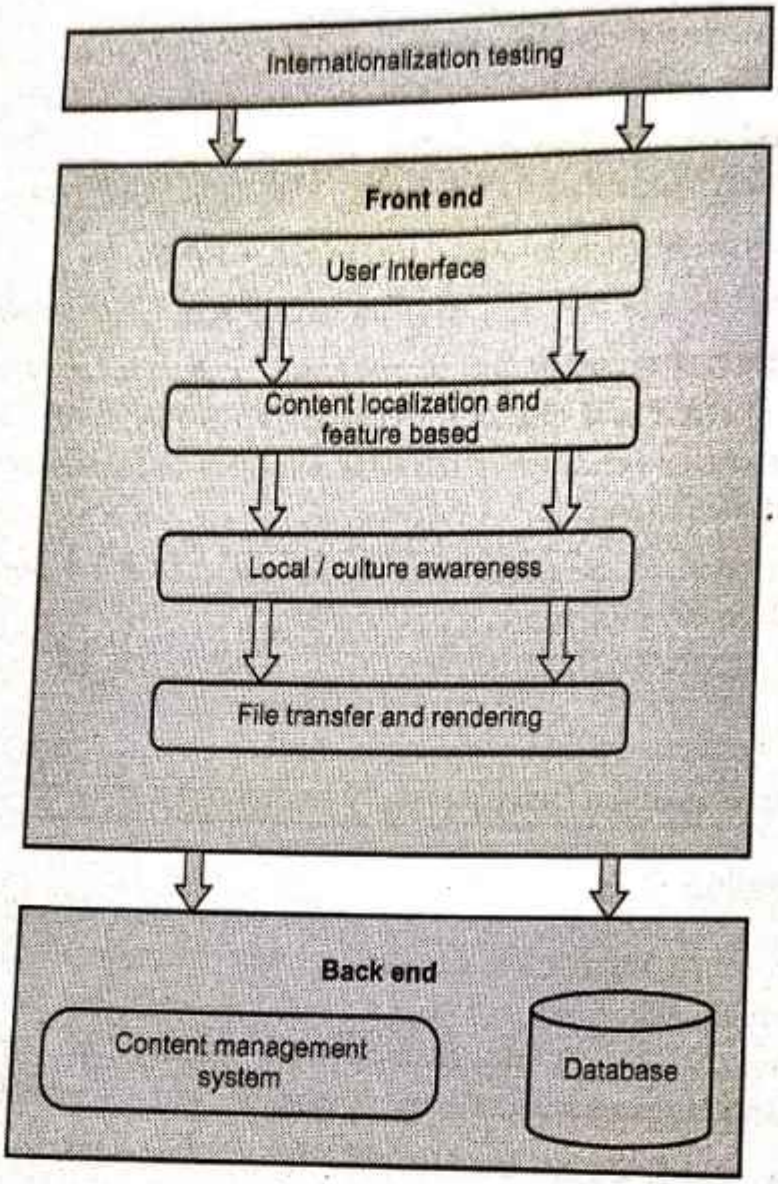

Fig. 3.8.1 Architecture of Internationalization testing

- Internationalization testing at back end :
- This process involves enabling the back end of a website to handle character encoding, form data submission, different languages and currencies and site search.
- Testing at the back end requires an understanding of Content Management System (CMS) that is used to store, author and publish content on the websites. Thorough understanding of the database is highly important for testing at the back end.
- Recent version of databases and content management system are already internationalized.

- Internationalization also checks if the product code can handle the international load and provide necessary support without distorting any function, losing data, or informational integrity. Ideally, software should be able to adapt to different cultures and languages without too many changes to source code.
- Internationalization essentially focuses on the design and development process of building a product that is built for a global audience. Therefore, Selenium internationalization testing can be extensively used for testing a global product at a large scale.
- In a globalized product, a code is separated from the messages or information. With the help of globalization, it enables software to be used with different languages without having to redesign the complete software.
- I18N testing can only be done for applications which supports multiple languages.
- Challenges of Internationalization Testing :
  1. Identifying audience
  2. Accessing the infrastructure.
- An example of internationalization is a website that makes the content available in various languages.
- **Advantages of Internationalization :**
  a) Higher quality software that meets the technical and cultural needs of multiple locales
  b) Reduced time, cost and effort for localization
  c) Single source code for all languages of the product
  d) Simpler, easier maintenance for future iterations of the product
  e) Greater in-country customer acceptance and satisfaction.

3.8.14 L1ON Testing

- Localization is a term in software that refers to support any language, any location culture. Localization testing is the software testing process for checking the localized version of a product for that particular culture or locale settings. The areas affected by localization testing are UI and content.
- L10N is also called **localization testing.** There are 10 characters between L and N.
- Localization is the process of adapting software which has already been internationalized to local or specific versions, translating the text to meet the standards of the target customer and adapting and customizing the non-textual parts of the software.

- Location culture means, the culture format of any geographical location such as date format, time format, currency format, number format etc. This is a vital issue when we need to deploy a world-ready software. It is also known as **L10N** testing.
- It is the aspect of development and testing relating to the translation of the software and its presentation to the end user. It includes translating the program, choosing the appropriate icons and graphics and other cultural considerations. It also may include the translation of help files and documentation.
- One important fact is that the quality of the international software is entirely dependent on how good the localization has been carried out. Localization therefore plays a key role in the quality assurance of international software. The quality of the product for a specific country or culture is established or achieved by applying software localization testing.
- **Example :** If the project is designed for Maharashtra State in India, the designed project should be in Marathi language and Marathi virtual keyboard should be present, etc.

1. Advantages of Localization Testing

- Overall testing cost reduce.
- Overall support cost reduce.
- Helps in reducing the time for testing.
- It has more flexibility and scalability.

2. Disadvantages of Localization Testing

- Requires a domain expert.
- Hiring local translator often makes the process expensive.
- A tester may face schedule challenges.

3.8.15 Difference between Localization and Internationalization Testing

Sr. No.	Localization Testing	Internationalization Testing
1.	Localization is referred as L10N.	Internationalization is referred as I18N.
2.	Localization itself means a specific local language for any given region.	Application code is independent of language.
3.	Localization testing allows checking out how well the product is adapted to the needs and culture of a certain target audience.	Internationalization testing represents designing and creating of a product and its documentation with the use of techniques that simplify localization of the app.

4.	Localization focuses on online help, GUI context, dialog boxes, error messages.	Internationalization focuses on compatibility testing, functionality testing, interoperability testing, usability testing.
5.	Optimize the product for one language or location.	It makes products easy to adapt.
6.	Localization is the process of adapting software for specific languages or regions.	Internationalization is the process of making software portable between languages or regions.
7.	Localization testing focuses on testing the product so that it is usable by users of a particular region.	Internationalization testing focuses on testing the product's functionalities and capabilities that are built for a global audience.

3.8.16 Compliance Testing

- Compliance testing, also known as **conformance testing**, is a type of software testing to determine whether a software product, process, computer program, or system meets a defined set of internal or external standards before it's released into production.
- In compliance testing, the primary goal is to provide a compliance report with details of any violations or missed procedures. The development team then uses this report to identify root causes and fix the identified gaps.
- These standards may get determined internally by the organization or externally, such as by clients or industry regulations. Performing compliance tests on software during their development allows the company to identify issues and make adjustments as needed to ensure compliance and quality.
- For example, a software developer may practice informal compliance testing when developing and testing their code by comparing their work against the specifications set for the project. This testing allows them to identify areas where their code does not align with expectations and make necessary adjustments to ensure compliance.
- **Process of compliance testing is as follows :**
  a) First, gather all the detailed rules, regulations, standards, and norms that your team needs to comply with.
  b) Secondly, record all the collected data in the first step accurately.
  c) Prepare a verification checklist for all developmental stages and modules. Make it available to the developers so that they know it before they begin development.

d) After each development stage, verify the checklist and prepare a report. Ensure that there is no conflict of interest during the verification process.

e) After your developers correct the errors, re-verify them to validate the affected areas.

f) If possible, apply for a compliance certificate.

3.8.17 Difference between Functional and Non-functional Testing

Sr. No.	Functional Testing	Non-functional Testing
1.	Functional tests confirm that the code is doing the right things.	Non-functional tests validate that the code is doing things the right way.
2.	Functional testing focus on user requirement.	Non-functional testing focus on user expectations.
3.	It can be performed both manually and with automation tools.	It requires automation tools for effective testing.
4.	Functional testing defines all the process and this is executed first.	Non-functional testing execution takes place only after the execution of functional testing because only if the system is defined, user can proceed further and customize it accordingly.
5.	Easy to execute black-box test cases.	Easy to execute white-box test cases.
6.	Types : Unit testing, integration, system testing.	Types : Performance, compatibility testing and usability testing.
7.	Business requirements are the inputs of functional testing.	Parameters like speed, scalability are the inputs of non-functional testing.

❑❑❑

UNIT IV

4 Software Quality Assurance and Quality Control

Syllabus

Software Quality Assurance : *Introduction, Constraints of Software Product Quality Assessment, Quality and Productivity Relationship, Requirements of a Product, Characteristics of Software, Software Development Process, Types of Products, Schemes of Criticality Definitions, Software Quality Management, Why Software Has Defects ? Processes Related to Software Quality, Quality Management System Structure, Pillars of Quality Management System, Important Aspects of Quality Management.*

Software Quality Control : *Software quality models, Quality measurement and metrics, Quality plan, implementation and documentation, Quality tools including CASE tools, Quality control and reliability of quality process, Quality management system models, Complexity metrics and Customer Satisfaction, International quality standards – ISO, CMM*

Contents

4.1 Introduction of SQA

- Software Quality Assurance (SQA) is a process that ensures that developed software meets and complies with defined or standardized quality specifications. SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets desired quality measures.
- **IEEE definition of "Software Quality Assurance" :** A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. A set of activities designed to evaluate the process by which the products are developed or manufactured
- **The various objectives of SQA are as follows :**
  1 Quality management approach.
  2 Measurement and reporting mechanisms.
  3 Effective software-engineering technology.
  4 A procedure to assure compliance with software-development standards where applicable.
  5 A multi-testing strategy is drawn.
  6 Formal technical reviews that are applied throughout the software process.
- **The major goals of SQA are as follows :**
  1 SQA activities are planned.
  2 Non-compliance issues that cannot be resolved within the software project are addressed by senior management.
  3 Adherence of software products and activities to the applicable standards, procedures and requirements is verified objectively.
  4 Affected groups and individuals are informed of SQA activities and results.

4.1.1 Differences between SQA and Software Testing

SQA	Software testing
SQA is about engineering process that ensures quality	Software Testing is to test a product for problems before the product goes live
QA ensures that the product design and functions meet the end user's expectations	Testing ensures that the product is defect-free and secure to use
SQA is proactive measure.	Testing is reactive-measure.

SQA is preventive technique	Testing is corrective technique.
It is subset of SDLC	It is subset of quality control.
It is a method to manage the quality-verification	It is a method to verify the quality-validation
It is a low-level activity, it can identify an error and mistakes which QC cannot	It is a high-level activity, it can identify an error that QA cannot

4.1.2 Constraints of Software Product Quality Assessment

- System analysts and business analysts prepare the requirement specification. Software tester may or may not have direct access to the customer and may get information through requirement statement, queries answered etc.
- Following are the some of the limitations for product quality assessment :
  1. Software is virtual in nature.
  2. Large communication gap between software user and developer team.
  3. Software is a product which is unique in nature. Similarities between any two products are superficial ones.
  4. Software program executes in the same way very time when it is executing some instructions.

4.2 Quality and Productivity Relationship

- Productivity is the relationship between a given amount of output and the amount of input needed to produce it. Quality affects productivity.
- Productivity is a tool of measurement that determines the efficiency of the organization in terms of the ratio of output produced with respect to inputs used. Various factors like technology, plant layouts, equipment, and machinery affect productivity.
- Productivity can be either measured as total productivity or as partial productivity where single variable or multiple variables are considered.
- Measuring productivity in production organizations is relatively easy. But measuring productivity for knowledge workers and in the service organizations is difficult.
- Maintaining time sheets to determine the time spent on each task and the quantity of work done is one of the ways of measuring productivity in the services industry.

- Quality is one of the key issues, which defines an organization's competitive position in the market. Quality is conformance to requirements. A well designed and properly produced product without any error may not be perceived as a quality product by the customers if it does not satisfy their requirements.

1. Improvement is quality directly leads to improved productivity.
2. Cost reduction is possible by improved quality.
3. Employee involvement is required for improved quality.
4. Proper communication between management and employee is essential.
5. Productivity is improved if scarp and rework is reduced.
6. Employees share responsibility for innovation and quality improvement.
7. Employee participate and contribute in quality improvement process.

4.2.1 Difference between Invention and Innovation

Invention	Innovation
Invention is unplanned activity.	Planned activity leading to changes.
Breakthrough changes are possible due to inventions	Changes in small steps are possible.
May not be directly applied to everyday work.	Easily applied to everyday work.
It may lead to scrapping old technologies.	It may lead to rearrangement of things but there may not be a fundamental change.

4.3 Requirements of a Product

- Requirement are divided into different types :

Types of Requirement	Remarks
Implied Requirement	• Functional and non-functional requirement specified by the customer. • No documented as a part of requirement. • Development team or test team must understand the requirements.
Specific Requirement	• Some requirement are generic in nature. • Many times generic requirement are considered as implied
Present/Future requirement	• Present requirement is required when an application is used in present circumstance. • Future requirements are for future needs which may be required after some time span. • Customer or system analysts specifies present and future requirement for projects.

Primary Requirements	• Also called must or must not requirements. • 'Must' requirement are primary requirement and customer pay for this requirement. • 'Must' requirement have highest priority in implementation. • 'Must not' requirement which must be absent in the product. • These requirement are denoted by priority P1.
Secondary Requirement	• 'Should be' and 'should not be' the secondary requirements. • 'Should be' requirement are appreciated by the customer if they present/absent. • These requirements are denoted by priority P2.
Tertiary Requirement	• 'Could be' and Could not be' requirement are tertiary requirements. • These are lowest priority requirements. • These requirement give a product identity in the market. • These requirements are denoted by priority P3

4.4 Characteristics of Software

- The quality of software has improved significantly over the past two decades. One reason for this is that companies have used new technologies in their software development process such as object-oriented development, CASE tools, etc.
- Software product is different than other product. Following is some of difference between them :
  1. There are different kinds of software product and their performance, capabilities etc vary considerably from each other.
  2. Every condition defined by the software program gets executed in the same way every time when it gets executed.
  3. Software cannot be sensed by common methods of inspection or testing.

4.5 Software Development Process

- A software development methodology in software engineering is a framework that is used to structure, plan, and control the process of developing an information system. A wide variety of such frameworks have evolved over the years, each with its own recognized strengths and weaknesses. One development methodology is not necessarily suitable for use by all projects.
- Each of the available methodologies is best suited to specific kinds of projects, based on various technical, organizational, project and team considerations.
- Following methods are discussed in this section.
  1. The Software Development Life Cycle model.
  2. The prototyping model.

3. The spiral model.
4. The object-oriented model.

4.5.1 Software Development Life Cycle Model

- The Software Development Life Cycle (SDLC) model is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software.
- The methodology within the SDLC process can vary across industries and organizations, but standards such as ISO/IEC 12207 represent processes that establish a lifecycle for software, and provide a mode for the development, acquisition and configuration of software systems.
- SDLC is classical model and still used in the industry. The most common example of the SDLC model is the **waterfall model.**
- **Waterfall model** is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete.
- The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. Fig. 4.5.1 shows waterfall model.

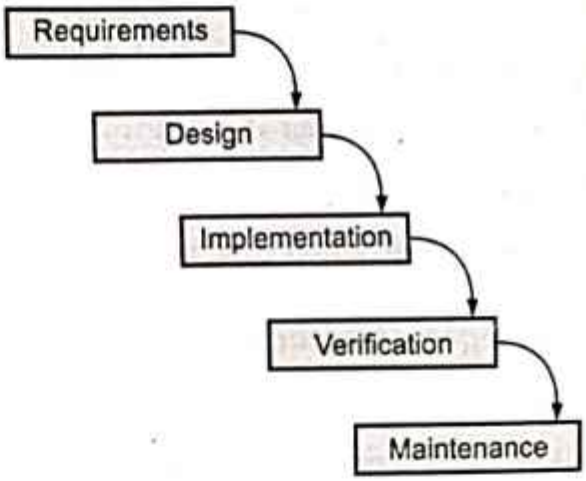

Fig. 4.5.1 Waterfall model

Basic principles :

1. Project is divided into sequential phases, with some overlap and splash back acceptable between phases.
2. Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
3. Tight control is maintained over the life of the project through the use of extensive written documentation, as well as through formal reviews and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase.

- **Requirements** analysis and specification is a study aiming to discover and document the exact requirements for the software system to be constructed.
- **Design** is an activity to construct a system, at a high level, to meet the system requirements.
- **Implementation** is where the design specification is transformed into a program written in a specific programming language, such as Pascal, C, or Java.
- **Verification and testing** is a way to detect potential faults in the program by running the program with test cases.
- **Deliver and maintenance** is where the ultimate system is delivered to the customer for operation, and is modified either to fix the existing faults when they occur during operation or to meet the new requirements.

Advantages of waterfall model :

1. Easy to understand, easy to use.
2. Provides structure to inexperienced staff.
3. Milestones are well understood.
4. Sets requirements stability.
5. Good for management control (plan, staff, track).
6. Works well when quality is more important than cost or schedule.

Disadvantages of waterfall model :

1. All requirements must be known upfront.
2. Difficult to respond to changes.
3. Can give a false impression of progress.
4. Problems are often not discovered until system testing.
5. Integration is one big bang at the end.
6. Little opportunity for customer to preview the system.

When to use the waterfall model ?

1. Requirements are very well known.
2. Product definition is stable.
3. Technology is understood.
4. New version of an existing product.
5. Porting an existing product to a new platform.

4.5.2 Prototyping Model

Basic principle :

1. Not a standalone, complete development methodology, but rather an approach to handling selected portions of a larger, more traditional development methodology.
2. Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process .
3. User is involved throughout the process, which increases the like-hood of user acceptance of the final implementation.
4. Small-scale mock-ups of the system are developed using an iterative modification process until the prototype evolves to meet the users' requirements.
5. While most prototypes are developed with the expectation that they will be discarded it is possible in some cases to evolve from prototype to working system.

- Prototype model is **iterative in nature.** A prototype of the actual product is preferred in situations such as user requirements are not complete and technical issues are not clear. Fig. 4.5.2 shows prototyping model.

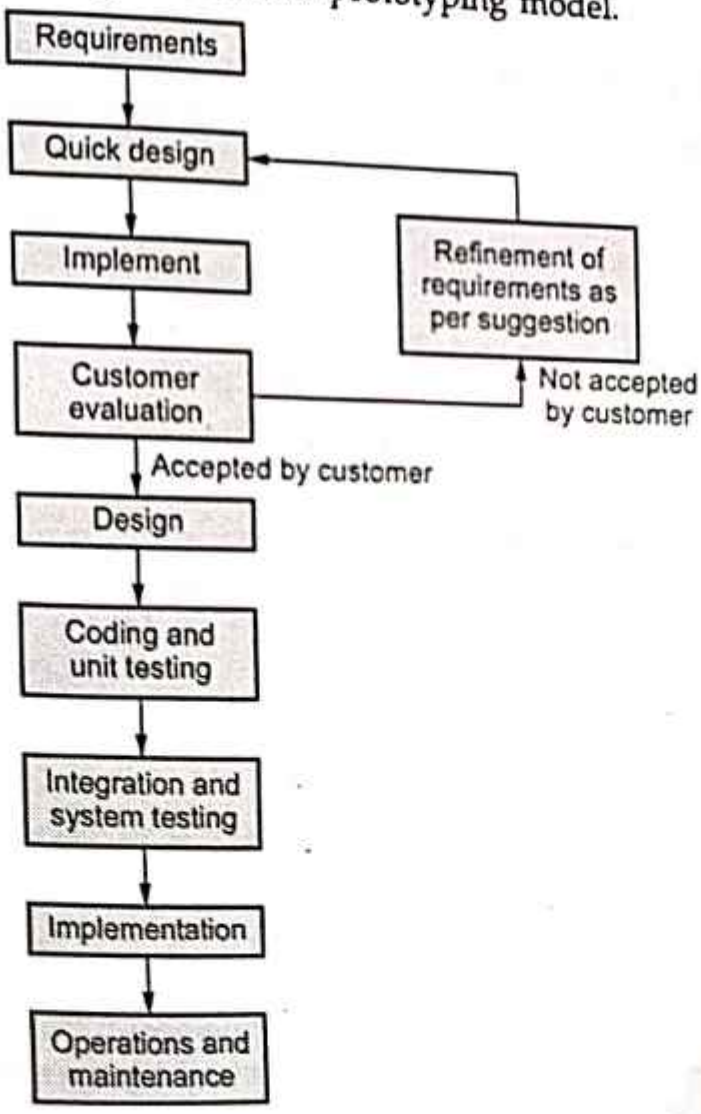

Fig. 4.5.2 Prototyping model

- Prototype is a scaled-down version of the actual model. The basic idea of prototyping model is that instead of freezing the requirements before any design or coding can proceed, a throw away prototype is built to help understand the requirements. This prototype is developed based on the currently known requirements.
- Development of prototype obviously undergoes design, coding and testing, but each of these phases are not done very formally or thoroughly. By using this prototype, the client can get an actual feel of the system. This results in more stable requirements that change less frequently.
- When applying the prototyping methodology, future users of the system are required to comment on the various versions of the software prototypes prepared by the developers. In response to customer and user comments, the developers correct the prototype and add parts to the system on the way to presenting the next generation of the software for user evaluation.
- Prototyping model is mainly used for small- to medium-sized software development projects.

Advantages of prototyping model :

1. Suitable for large system for which there is no manual process to define the.
2. It provides better understanding of the customer's needs.
3. Helps to reduce the cost and time.
4. Improves communication.
5. Helps to detect errors early.
6. Lets the developers have greater control over the problem.

Disadvantages of prototyping model :

1. Does not eliminate the time gap, between the definitions of requirements and final delivery of the application.
2. Does not stress the need for anticipating changes.
3. Suffers from bad documentation.
4. Developing the system may become a never-ending process.
5. Prototype has to be discarded. So at times the cost involved is higher.

4.5.3 The Spiral Model

- In 1988 Boehm developed the spiral model as an iterative model which includes risk analysis and risk management.

- **Basic principles :** Focus is on risk assessment and on minimizing project risk by breaking a project into smaller segments.
- Spiral model is combination Linear and Iterative model. The Spiral Model (Barry Boehm) is a risk-centered development model where each spiral includes major risk activities / assessments.
- The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on.
- Fig. 4.5.3 shows spiral model.

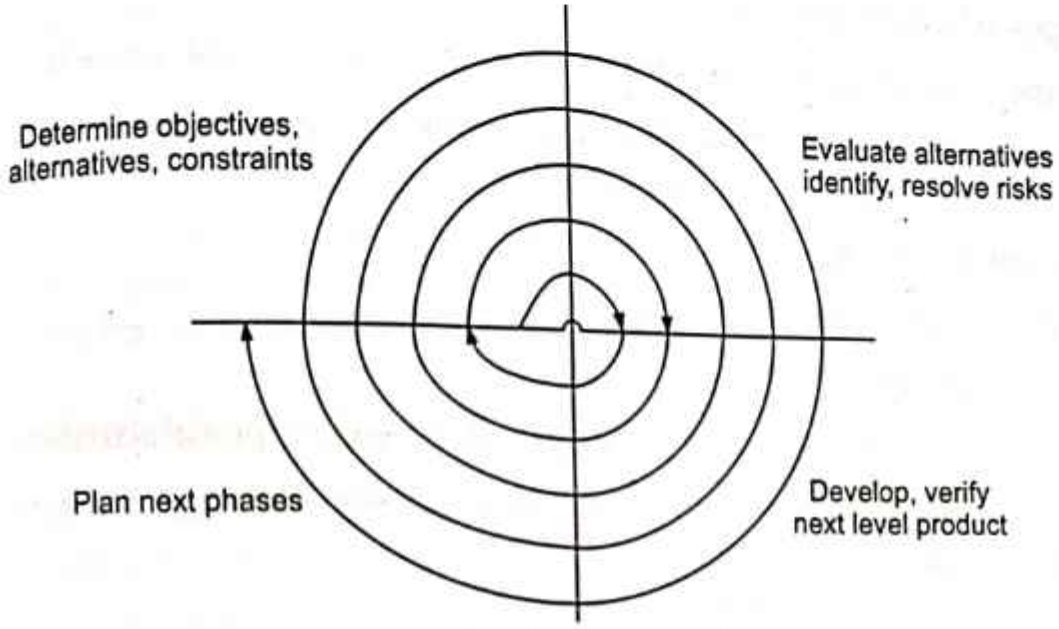

Fig. 4.5.3 Spiral model

- Each cycle of the spiral begins with the identification of :
  1. The objectives of the portion of the product being elaborated (performance, functionality, ability to accommodate change, etc.);
  2. The alternative means of implementing this portion of the product (design A, design B, reuse, buy, etc.); and
  3. The constraints imposed on the application of the alternatives (cost, schedule, inter-face, etc.).
- Following activity are performed by spiral model :
  1. Planning
  2. Risk analysis and resolution.
  3. Engineering activities according to the stage of the project : design, coding, testing, installation and release.
  4. Customer evaluation, including comments, changes and additional requirements, etc.

Advantages of spiral model :

1. Provides early indication of insurmountable risks, without much cost.
2. Users see the system early because of rapid prototyping tools.
3. Critical high-risk functions are developed first.
4. The design does not have to be perfect.
5. Users can be closely tied to all lifecycle steps.
6. Early and frequent feedback from users.
7. Cumulative costs assessed frequently.

Disadvantages of spiral model :

1. Time spent for evaluating risks too large for small or low-risk projects.
2. Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive.
3. The model is complex.
4. Risk assessment expertise is required.
5. Spiral may continue indefinitely.
6. Developers must be reassigned during non-development phase activities.
7. May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration.

- Spiral model is used in following situations :

1. Real-time or safety-critical systems.
2. Risk avoidance is a high priority.
3. Minimizing resource consumption is not an absolute priority.
4. Project manager is highly skilled and experienced.
5. Requirement exists for strong approval and documentation control.
6. Implementation has priority over functionality, which can be added in later versions.

4.5.4 The Object-Oriented Model

- It is component-based software development model. Easy integration of existing software modules into newly developed software systems. The process begins with object-oriented analyses and object-oriented design. Then, acquire suitable components from reusable software component libraries. Otherwise, develop as needed.

- It can involve adding to repertoire of library components. Because of integrating reusable components; much lower cost than developing. It improved quality by using tested components.
- The system under development is refined and transformed through analysis, design, code and test phases-details are added in successive iterations and incremental releases of software modules are delivered.
- In an object-oriented environment, software is a collection of discrete objects that encapsulate their data as well as the functionality of model real-world events "objects" and emphasizes its cooperative philosophy by allocating tasks among the objects of the applications.
- The basic software development life cycle consists of analysis, design, implementation, testing and refinement. Its main aim is to transform users' needs into a software solution.
- The development is a process of change, refinement, transformation or addition to the existing product. The software development process can be viewed as a series of transformations, where the output of one transformation becomes input of the subsequent transformation.

4.5.5 Agile Development Model

- Agile software development is a conceptual framework for software engineering that promotes development iterations throughout the life-cycle of the project.
- Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks. Agile methods also emphasize working software as the primary measure of progress.
- Characteristics of Agile Software Development

1. Light Weighted methodology
2. Small to medium sized teams
3. Vague and/or changing requirements
4. Vague and/or changing techniques
5. Simple design
6. Minimal system into production

- Most agile development methods break product development work into small increments that minimize the amount of up-front planning and design.
- Iterations, or sprints, are short time frames that typically last from one to four weeks. Each iteration involves a cross-functional team working in all functions : planning, analysis, design, coding, unit testing, and acceptance testing.

- At the end of the iteration a working product is demonstrated to stakeholders. This minimizes overall risk and allows the product to adapt to changes quickly.
- An iteration might not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of each iteration.
- Multiple iterations might be required to release a product or new features. Working software is the primary measure of progress .

4.6 Types of Products

- Different types of software product are listed below :

4.6.1 LIfe Affecting Product

- Software product which affect directly/indirectly human life. They are considered as most critical products from user view point.
- This type of product come under the purview of regulatory and safety requirement, in addition to normal customer requirement.
- These product are classified as follows :
  1. Any product failure resulting into death of a person.
  2. Any product failure which may cause permanent disablement to a patient.
  3. Any product failure which may cause temporary disablement to a patient.
  4. Any product failure which may cause minor injury and does not result into anything.
  5. Other product which do not affect health.

4.6.2 Product Affecting Huge Amount of Money

- This type of product has direct relationship with loss of huge sum of money. These products require large testing effort and have many regulatory as well as statutory requirements. Example : eCommerce and eBusiness software.
- Some of the important quality factors of such products are security, accuracy and confidentially.
- These product also need high confidence level and huge testing will represent the critically.

4.6.3 Product based on Simulator

- Some product tested only on simulator. In this case, real life scenario is either impossible to create or may not be economically viable.

- Example : Product used in space research, aeronautic may be put in this class.
- This type of product also needs huge testing.

4.7 Schemes of Criticality Definitions

- Product are classified in different ways.

1. From user perspective

- Failure of product disrupts the entire business. There is no fallback arrangement available.
- Product failure which affects business partially. Some fallback arrangement is available.
- Product failure which does not affect business at all is one of the options.

2. Another way of defining user perspective

- Product where user environment is very complex such as space reach, may be considered as very critical.
- Product where user environment is very simple.
- Products where user environment is comparatively less complex may represents the second stage of complexity.

4.8 Software Quality Management

- Quality management involves management of all inputs and processing to the processes defined so that the output from the process is as per defined quality criteria.
- It handles three levels of problem :

1. **Correction :** It gives idea about condition where defects found in the product or service are immediately sorted and fixed. This is natural phenomenon happens with tester while testing. This is quality control approach.
2. **Corrective actions :** To find root causes of defect, analysis is required. Corrective action programs provide an organized way to problem solving. The end goal is to fix an issue so that it doesn't come back. Once the root cause is pinpointed, timely actions are taken to prevent the problem from happening again with adequate follow-up and solution verification. The need for a corrective action is typically identified through inspections, audits, process failures or employee input. However, the deal is that the problem has already occurred and the response is reactionary.
3. **Preventive actions :** Preventive actions are implemented prior to the occurrence of a non-conformance or deviance. Preventive action programs are proactive tools

tied to quality system elements that involve monitoring and assessing system and process effectiveness. Being proactive means staying on top of process inputs, outputs, and other data that signal trends and abnormalities and acting on them before they become issues.

- Organizations typically identify preventive action opportunities during management reviews or other meetings where quality or quality system data is reviewed for would-be problems.
- Sometimes, changes in contractual or regulatory requirements dictate the need for preventive actions. Once potential issues are identified, preventive actions can be determined, implemented and evaluated for effectiveness

4.9 Why Software Has Defects ?

- Software bug is a failure or flaw in a program that produces undesired or incorrect results. It's an error that prevents the application from functioning as it should.
- There are many reasons for software bugs. Most common reason is human mistakes in software design and coding.
- When the application does not meet the requirements, or it does not perform as expected by the customer, it has defects.
- When the application does not look good to the customer, or it does not 'feel' right, it has defects.
- Defects are not always coding mistakes done by the developers. Most defects are, but not all. Defects can come from any stage in the software development cycle, because every stage involves people.
- In the requirements stage, when communicating with the customer, there is plenty of room for error. The customer's needs may be misunderstood or misinterpreted, the customer may mis-communicate his or her needs, and even the requirements document writer may inadvertently cause errors.
- In the architectural design stage, an inexperienced or careless designer can always cause problems later on.
- The development stage is probably responsible for most of the defects in all applications. The testing stage is where most defects are found, but some defects can be introduced here as well.

4.10 Processes Related to Software Quality

- Software quality management is the collection of all processes that ensure that software products, services, and life cycle process implementations meet organizational software quality objectives and achieve stakeholder satisfaction

1. Vision

- Vision statement is an organization's declaration of its mid-term and long-term goals.
- Vision statements are often confused with mission statements. Some organizations provide one or the other, and some provide a single message that combines elements of both.
- Strictly speaking, both messages communicate the organization's values and purpose but a mission statement focuses on current operations and a vision statement on the future. The value of both communications lies in their ability to foster positive public relations.
- Examples of vision statements :
  1. Amazon : Our vision is to be earth's most customer-centric company; to build a place where people can come to find and discover anything they might want to buy online.
  2. Microsoft : Our vision is to provide experiences for our customers and partners, across all of their interactions with Microsoft, that they value and recognize, and enable them to realize their full potential.

2. Mission

Success of missions is essential for achieving the organization vision. Mission are expected to support each other to achieve the overall vision put by management. Mission may have different life-spans and completion dates.

1. Policy : It talks about a way of doing business as defined by senior management. This statement helps employee, suppliers and customers to understand the thinking and intent of management.
2. Objectives : It define quantitatively what is meant by a successful mission. Every mission must have minimum one objective.
3. Strategy : It defines the way of achieving a particular mission.
4. Goals : Goals define the milestones to be achieved to make the mission successful.

4.11 Quality Management System Structure

- There is no certain standard for the structure of the quality management system, it just has to be documented. The documenting can be done in any form as long as it is suitable for the organization's use.
- When the organization is starting to implement the quality management system in its operations, there should be a reason why this is done. The reason should come from top management, and it should not be "because the competitors are doing it or be-cause we want to have quality certification".
- The review of current business is done to clarify the purpose of the organization and to understand the current market position. The following items should be considered, when reviewing the business :
  1. Customer
  2. Product
  3. Stakeholders
  4. Description of the operation
  5. Strengths and weaknesses
  5. Market share
  6. Competitors and the state of competition
- Fig. 4.11.1 shows quality management system for typical organization.

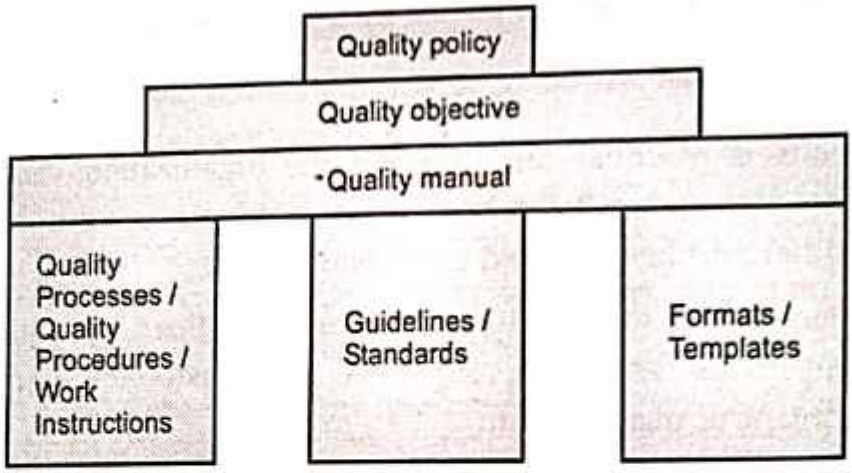

Fig. 4.11.1 Quality management system for typical organization

- Quality Policy : It is first tier system. It is basic framework on which the quality temple rests. Quality policy sets the wish, direction by the management about how activities will be conducted.
- Quality objective : It is two tier system. Objectives are the measurement established by the management to define progress and achievements in a numerical ways.

- Quality manual : It is three tier system. It is published by management of the organization.

4.12 Pillars of Quality Management System

- Quality processes, quality procedures, work instruction and methods are defined at an organization level by the functional area experts, and project and functional level by the experts in those area separately.
- Guidelines and standards : Guidelines and standards are used by organization project team for achieving quality goals for the products and services delivered to customers.
- Formats and templates : Common format and templates are used for tracking a project, function, information within organizations. It act as checklist to maintain consistency across the projects in the organizations.

4.13 Important Aspects of Quality Management

- For a TQM program to be successful, five major things must be in place within an organization. They are :

1. Continuous improvement
2. Customer satisfaction
3. Managerial involvement
4. Measurability; the ability to accurately measure and record quality and defects
5. Organizational support for total quality

- Through continuous improvement, production cycles are short and iterative; this allows for fixes and enhancements to be made almost immediately on the spot.
- Customer satisfaction is also necessary for a TQM program to work, obviously, it is not enough to simply improve processes to the point where errors are at an absolute minimum; an organization looking to refine its processes must be aware of the needs of its customers and tie those needs into the refactoring process.
- Additionally, managerial involvement and an overall support system for TQM are required for the program to be successful. It is not enough to have only the front-line workers be knowledgeable in the area of total quality management - every one within the organization must be involved in an implementation of this magnitude.
- Finally, there must also be an organizational support system in place for the program to work. That is, there must be a linkage between the TQM program and the financial, strategic, and the human resources aspects of the organization in order for an organization to truly reap the rewards of total quality management.

Aspects of Total Quality Management Implementation

TQM Phases	TQM Elements	Management Methods and Tools
Customer Focus	Customer focus	Customer complaint, Investigation, Market investigation, Customer Satisfaction survey, After sales service, Formally feedback system, Customer day
Planning Process	Leadership	Top management commitment, Policy, Corporate quality council, Division and site quality council, Cross functional quality council
	Vision and plan statement	Vision/Mission statement, Quality policy, Quality goals, Quality planning
Process Management	Supplier quality management	Supplier audit, Training, Potential supplier evaluation, Supplier certification.
	Product design	Concurrent engineering, Reliability engineering, Designing for manufacturability, Design of Experiments (DOE), Quality Function Deployment (QFD), Value Engineering, Computerized Design (CAD)
Process Improvement	Evaluation	Strategic evaluation, Business evaluation, Quality costs, Department evaluation Bench marking, Employee performance evaluation, Quality audit, Team evaluation
	Process control and improvement	PDCA Cycle, 7 QC tools, SPC, FTA, FMEA, Process capability, Equipment maintenance / improvement, Sampling inspection, Self inspection
Employee Participation	Recognition and reward	Condition improvement, Salary promotion, Bonus system, Moral award, Award ceremony
	Education and training	Quality awareness education program, Quality management method education, Training for job requirements, Individual training plan, Education promotion.
	Participation	Within functional delegated team, Cross functional delegated team, Information communication, Quality control circle, Voluntary team, Job rotation, Improving employee, Establishing, Quality culture, Suggestion activities.
	Communication	Information communication, News letter, Poster, Slogan, Quality day.

4.4 Software Quality Control

- Software is computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
- Software quality, by definition, is the degree to which software possesses a desired combination of attributes.
- Software quality is the degree to which a system, component, or process meets specified requirements.
- Software quality is the degree to which a system, component, or process meets customer or user needs or expectations. Quality means conformance to requirements
- Quality that is defined as a matter of products and services whose measurable characteristics satisfy a fixed specification - that is, conformance to an in beforehand defined specification.
- The nine causes of software errors are :
  1. Faulty requirements definition
  2. Client-developer communication failures
  3. Deliberate deviations from software requirements
  4. Logical design errors
  5. Coding errors
  6. Non-compliance with documentation and coding instructions
  7. Shortcomings of the testing process
  8. User interface and procedure errors
  9. Documentation errors
- Meeting customer needs : Quality that is identified independent of any measurable characteristics. That is, quality is defined as the products or services capability to meet customer expectations explicit or not.
- The fundamental reason for measuring software and the software process is to obtain data that helps us to better control the schedule, cost, and quality of software products. It is important to be able to consistently count and measure basic entities that are directly measurable, such as size, defects, effort and time.
- Consistent measurements provide data for doing the following :
  1. Quantitatively expressing requirements, goals and acceptance criteria.
  2. Monitoring progress and anticipating problems.
  3. Quantifying tradeoffs used in allocating resources.
  4. Predicting the software attributes for schedule, cost and quality.

- With software or anything else, assessing quality means measuring value. Something of higher quality has more value than something that is of lower quality. Yet measuring value requires answering another question : value to whom ? In thinking about software quality, it's useful to focus on three groups of people who care about its value.
  1 The software's users, who apply this software to some problem.
  2 The development team that creates the software.
  3 The sponsors of the project, who are the people paying for the software's creation.

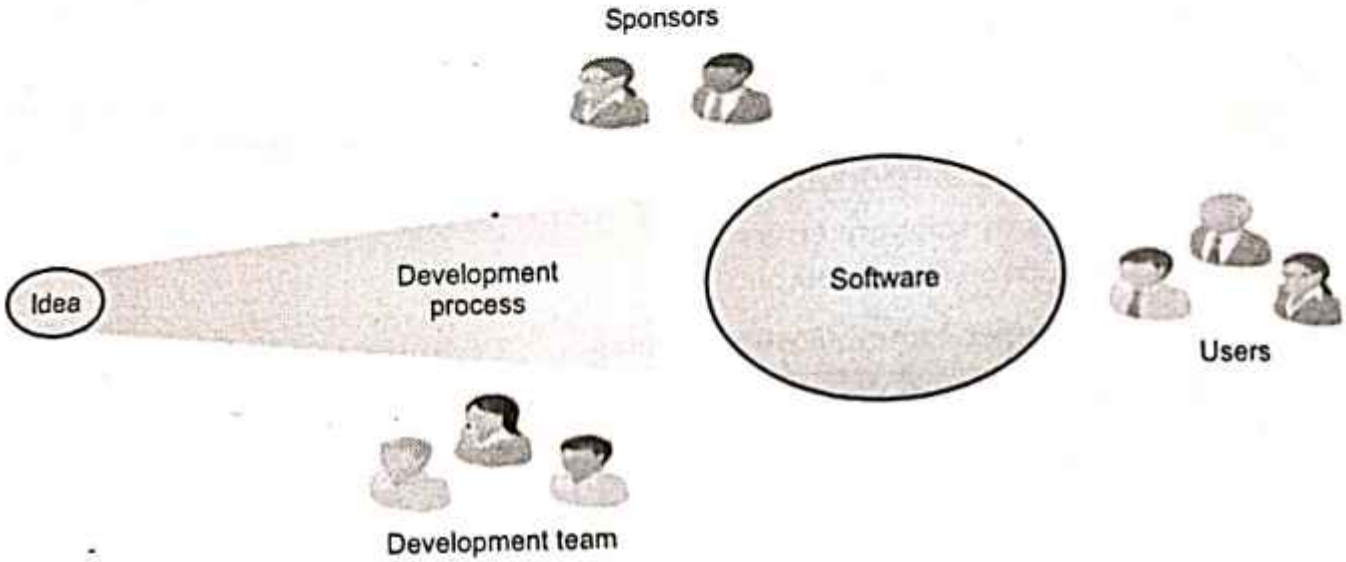

Fig. 4.14.1 Need of sofwtare quality

4.14.1 Quality Challenges

- Statements like "a system will have high performance" or "a system will be user friendly" are acceptable in the really early stages of requirements elicitation process.
- As more information becomes available, the above statements become useless as they are meaningless for the purpose of actually designing a solution. In each of the two examples above an attempt is made to describe the behavior of a system. Both statements are useless as they provide no tangible way of measuring the behavior of the system.
- The quality attributes must be described in terms of scenarios, such as "when 100 users initiate 'complete payment' transition, the payment component, under normal circumstances, will process the requests with an average latency of three seconds." This statement or scenario, allows an architect to make quantifiable arguments about a system.

- **Limited warranty on software and media :** Company warrants the disks on which the software is distributed to be free from defects in materials and workmanship and that the software will perform substantially in accordance with the documentation for a period of 90 days from your receipt of the product.
- If the product fails to comply with the warranty set forth above, company entire liability and your exclusive remedy will be replacement of the disk. Company reasonable effort to make the product meet the warranty set forth above. ...
- Software products differ from other industrial products with respect to the following characteristics :

1. **Product complexity :** Typical software product allows tens of thousands of operational options. Typical industrial products and even advanced industrial products do not reach this level of variety of options.
2. **Product visibility :** Since software products are invisible, defects in the software are not visible unless testing procedures are applied. But industrial products are visible and most defects are visible to the production team by changes in color or shape.
3. **Development and production process :** Defects in industrial products usually occur during each stage through which the product has to pass, namely development, product production planning and manufacturing. At each stage the product is examined and tested independently by a different team. In contrast, software products are examined and tested during the development stage only.

- Factors affecting defect detection in software products vs. other industrial products.

Parameters	Software products	Other industrial products
Complexity	Higher complexity It allows larger number of operational options	Lower complexity It allows thousands of operational options
Product visibility	Product is invisible Detection of defects is difficult by sight	Product is visible Detection of defects is easy by sight
Nature of development and production process	Defect detection is possible only in one phase	Defect detection is possible in all phase of development and production

4.15 Software Quality Models

- The models of software quality are representations abstract and simplified which touch or affect the software quality. There are two different types of models of software quality, that is the general models and the specific models :
  a) The general models are developed to be used with all the classes of existing software applications. So, the attributes of these models are chosen to be applicable to any software.
  b) The specific models are developed to be exclusively used with a class of software application in particular.
- The attributes of these models are thus chosen to cover all the aspects of the quality which are relevant in the considered class. These models of software quality are conceived to identify the quality requirements and the criteria of acceptance for software so they allow to estimate and to guide the progress of the software development with regard to the quality criteria.
- Finally, these models facilitate the communication of the aspects of the quality to the customers, the users and the various groups of the team of development. Then, to measure the quality, these models propose quality factors which are directly observable by the users.
- These factors will be estimated by means of several criteria which are observable that by developers and which are measured and estimated by means of one or several metrics

4.15.1 McCall's Quality Model

- Classifies all software requirement into 11 software quality factors, grouped into three categories :
  1. Product operation factors : Correctness, Reliability, Efficiency, Integrity, Usability.
  2. Product revision factors : Maintainability, Flexibility, Testability.
  3. Product transition factors : Portability, Reusability, Interoperability.

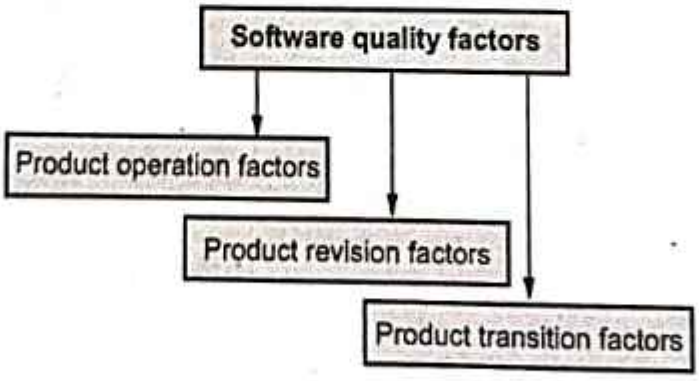

Fig. 4.15.1

Product operation factors

a. **Correctness** : Extent to which a program fulfills its specification. Correctness requirements are defined in a list of the software system's required outputs. Output specifications are usually multidimensional; some common dimensions include :

i. The output mission
ii. The required accuracy of output
iii. The completeness of the output information
iv. The up-to-dateness of the information
v. The availability of the information
vi. The standards for coding and documenting the software system.

b. **Reliability** : Ability not to fail. Determine the maximum allowed software system failures rate, and can refer to the entire system or to one or more of its separate functions.

c. **Efficiency** : Use of resources execution and storage. Efficiency requirements deal with the hardware resources needed to perform all the functions of the software system in conformance to all other requirements

d. **Integrity** : Protection of the program from unauthorized access.

e. **Usability** : Ease of use of the software.

Product revision factors

a. **Maintainability** : Effort required to locate and fix a fault in a program.

b. **Flexibility** : Ease of making changes required by changes in operating environment.

c. **Testability** : Ease of testing the program to ensure that it is error-free and meets its specification.

Product transition factors

a. **Portability** : Effort required to transfer a program from one environment to another system.

b. **Reusability** : Ease of re-using software in a different context.

c. **Interoperability** : Effort required to couple a system to another system.

Criteria for evaluation of software quality at NASA

- At NASA, the criteria for evaluation of software quality are taken from McCall's software quality factors model.

- Selection of criteria is application dependent.
- At NASA, selection of criteria is mission dependent and environment dependent.
  - Each quality factor is positively influenced by a set of quality criteria, and the same quality criterion impacts a number of quality factors. Example : Simplicity impacts reliability, usability and testability.
  - If an effort is made to improve one quality factor, another quality factor may be degraded. Portable code may be less efficient.
  - Some quality factors positively impact others. An effort to improve the correctness of a system will increase its reliability.

4.15.2 Boehm Model

- Boehm model establishes large-scale characteristics that constitute an improvement over the McCall model because it adds factors at different levels. Their quality model represents a hierarchical structure of characteristics, each of which contributes to the total quality. The high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put.
- Fig. 4.15.2 shows Boehm model. (See Fig. 4.15.2 on next page.)
- The high-level factors are :
  a) Utility indicating the easiness, reliability and efficiency of use of a software product;
  b) Maintainability that describe the facilities to modify, the testability and the aspects of understanding;
  c) Portability in the sense of being able to continue being used with a change of environment.
- The intermediate level characteristic represents Boehm's 7 quality factors that together represent the qualities expected from a software system :
  a) Portability (General utility characteristics) : Code possesses the characteristic portability to the extent that it can be operated easily and well on computer configurations other than its current one.
  b) Reliability (As-is utility characteristics) : Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily.
  c) Efficiency (As-is utility characteristics) : Code possesses the characteristic efficiency to the extent that it fulfills its purpose without waste of resources.

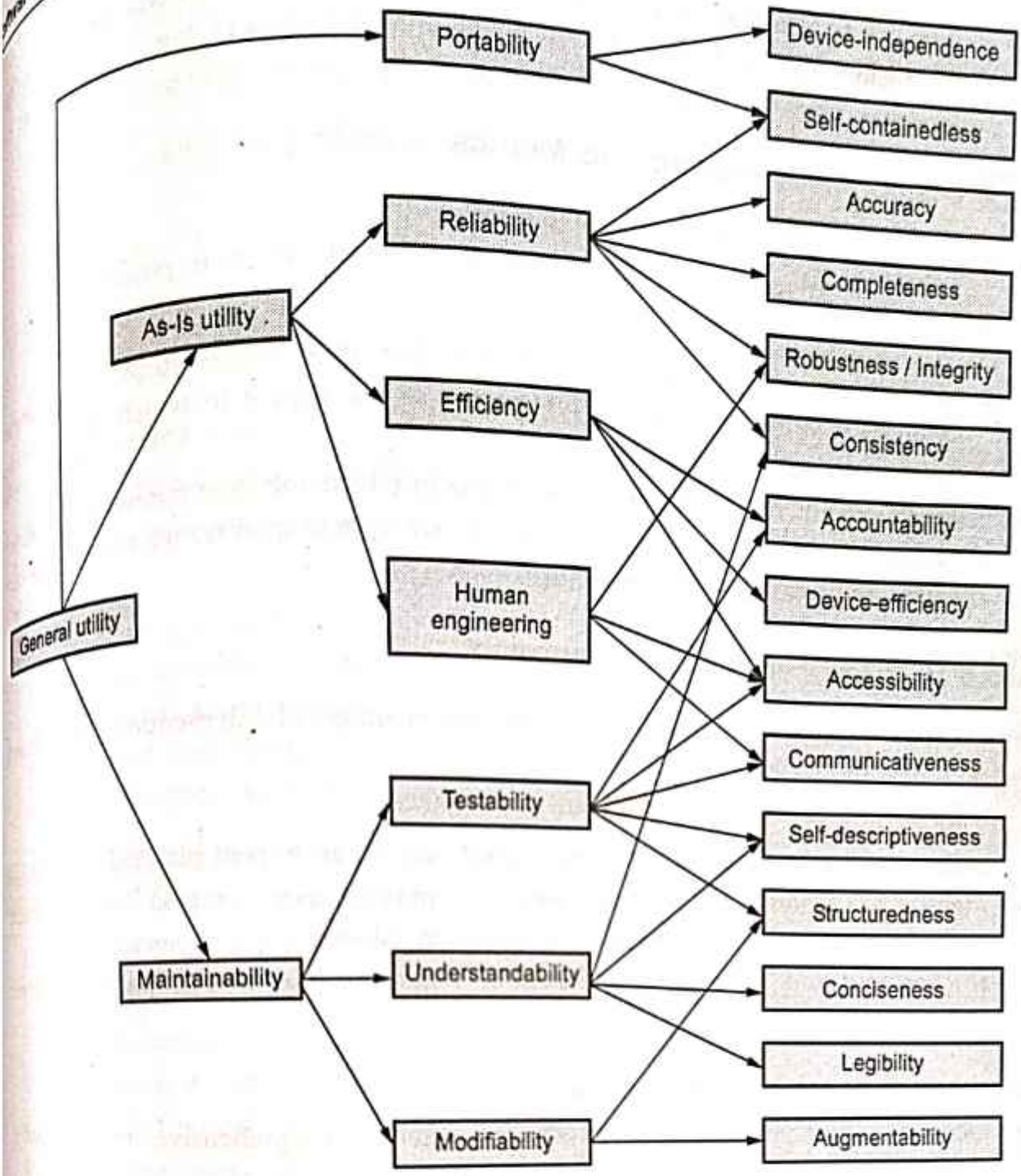

Fig. 4.15.2 Boehm model

d) Usability (As-is utility characteristics, Human Engineering) : Code possesses the characteristic usability to the extent that it is reliable, efficient and human-engineered.

e) Testability (Maintainability characteristics) : Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.

f) Understandability (Maintainability characteristics) : Code possesses the characteristic understandability to the extent that its purpose is clear to the inspector.

g) Flexibility (Maintainability characteristics, Modifiability) : Code possesses the characteristic modifiability to the extent that it facilitates the incorporation of changes, once the nature of the desired change has been determined.

4.16 Quality Measurement and Metrics

- IEEE definition of software quality metrics :
  1. A quantitative measure of the degree to which an item possesses a given quality attribute.
  2. A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.
- Software quality, by definition, is the degree to which software possesses a desired combination of attributes. Therefore, to improve system quality, we must focus our attention on software-quality attributes.

4.16.1 Objectives of Quality Measurement

- Facilitate management control, planning and managerial intervention. It is based on following parameters :
  a. Deviations of actual from planned performance.
  b. Deviations of actual timetable and budget performance from planned.
- Identify situations for development or maintenance process improvement (preventive or corrective actions). It is based on following parameters :
  a. Accumulation of metrics information regarding the performance of teams, units, etc.

Software quality metrics - requirements

- General requirements are relevant, valid, reliable, comprehensive and mutually exclusive.
- Operative requirements are as follows :
  a. Easy and simple b. Does not require independent data collection
  c. Immune to biased interventions by interested parties

4.16.2 Classification of Software Quality Metrics

- Classification by phases of software system :
  1. Process metrics : Metrics related to the software development process
  2. Maintenance metrics : Metrics related to software maintenance
  3. Product metrics : Metrics related to software artifact

- Product metrics describe the characteristics of the product such as size, complexity, design features, performance, and quality level. Process metrics can be used to improve software development and maintenance
- Classification by subjects of measurements
  1. Quality
  2. Timetable
  3. Effectiveness
  4. Productivity

Software size/volume measures

1. Thousands (Kilos) of Lines of Code (KLOC) : KLOC is a way of measuring the size of a computer program by counting the number of lines of source code a program has.
2. Number of Function Points (NFP) : Function points measure software size by quantifying the functionality provided to the user based solely on logical design and functional specifications. Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user.

Review Questions

1. *Explain how software quality metrics are categorized.*
2. *Explain why statistical analysis method is required for software quality metric results.*
3. *Discuss in detail how to manage software quality with suitable example.*
4. *Briefly discuss the different types of software quality metrics with an example.*
5. *Discuss about the difficulties involved in managing software quality.*
6. *Explain about various software metrics that plays a role in measuring the aality of software.*

4.17 Quality plan

- Project needs development plan and quality plan :
  1. Based on proposal material that have been re-examined and updated.
  2. More comprehensive than the approved proposal, especially with on the schedule, resource estimates and development risk.
  3. Include additional subjects, absent from the approved proposal.

Objective of development plan and quality plan

- Planning is meant to prepare adequate foundations for successful and timely completion of the project
- Planning, as a process, has several objectives, each is meant to prepare enough foundation for the following :-
  1. Scheduling development activities
  2. Recruiting team members and allocating development resources
  3. Resolving development risk
  4. Implementing required SQA activities
  5. Providing management with data needed for project control

Elements of the development plan

- The project development plan is prepared using proposal materials. The following elements, each is applicable to different project component.
  1. **Project products :** The development plan includes the following :
     - Design documents specifying dates of completion
     - Software products (completion date and installation site)
     - Training task (dates, participants and sites)
  2. **Project interfaces :** The development plan include the following :
     - Interfaces with existing software interface
     - Interface with other software /hardware development teams that are working in the same project
     - Interfaces with existing hardware
  3. Project methodology and development tools to be applied at each phase of the project
  4. Software development standards and procedures
  5. The mapping of the development process. It involves providing detailed descriptions of each project phases
  6. Project milestones
  7. Project staff organization
  8. Development facility and risk
  9. Various control methods
  10. Project cost estimation

Elements of the quality plan

1. List of quality goals : These refer to the quality requirements in the developed software system. Quality goals should reflect the major acceptance criteria found in the requirement's document (an RFP) correctness, reliability, robustness, maintainability.
2. Planned review activities : The planned reviews should include a listing of all reviews. For example design reviews, technical reviews, managerial reviews, code inspections etc. All reviews need to include :
   a. Scope - what does it cover
   b. Type - emphasis - managerial, technical, super detailed...
   c. Schedule - often based on previous reviews and outcomes
   d. Procedures - action lists; present and discuss; ...
   e. Who is to attend ? Collateral interest ?? *****
3. Planned software tests : Must include a complete list of planned tests
4. Planned acceptance test for externally developed software. A complete set of acceptance tests to be run for externally developed software must be provided within the quality plan. Complete set must be run for our own developed software. Especially critical for purchased software, contracted software, customer-supplied software. These tests can be run in parallel with internally-developed software tests.
5. Configuration management : The quality plan must include configuration management tools and procedures for managing the software configurations, versions, etc.

- The quality plan may be included within the development plan or as an independent document. The document, however compiled, must be reviewed and approved by the organization's standard approval process

Development and quality plan for small projects

- As this team knew, some projects just don't need a long involved project plan document; they may be "small" in that they're of short duration, simple or straightforward, well-defined, and/or to be executed by a small team.

Small project assumptions

1. A small project has 25 people or less
2. Project team generally works together on all phases of product development
3. Must trade-off limited resources
4. Testers are often the developers

5. Need independent inspection at critical phases
6. Quality engineers must have technical expertise to add value on a small project

Several advantages to planned over unplanned projects

1. A more comprehensive / thorough understanding of the task is likely (gained when developing the plan)
2. Greater responsibility for meeting obligations can be assigned, as they can be 'seen' more clearly since articulated (who does what)
3. Easier to share control of the project and identify unexpected delays (any plan better than no plan at all!)
4. Better understanding of requirements and timetable can be reached between customer and developer.

4.18 Quality Tools Including CASE Tools

- Collection of useful tools that help in every step of building a product. Organized layout that enables tools to be found quickly and used efficiently. Skilled craftsperson who understands how to use the tools effectively.
- There are lots of automated tools to assist in software engineering. Their purpose is to make the work of software development and maintenance esier and more reliable.
- Software engineers need two types of tools.

1) Analytical tools used in software development such as stepwise refinement and cost-benefit analysis theoretical tools.
2) Products that assist the teams of software engineers in developing and maintaining software usually termed CASE tools.

- CASE is Computer Aided Software Engineering. CASE tool means any tool used to automate some activity associated with software development.
- CASE tools are computerized software development tools that support the developer when performing one or more phases of the software life cycle and/or support software maintenance.
- The primary reasons for using a CASE tool are to increase productivity and to help produce better quality software at lower cost.
- Depending on the CASE tool, the following system development activities may be covered :

1. Project identification and selection
2. Project initiation and planning

3. Analysis
4. Design
5. Implementation
6. Maintenance

Objectives of CASE

1. Improve quality of developed systems.
2. Increase speed of systems development.
3. Improve testing process through automatic checking.
4. Integrate development activities (common methods).
5. Improve documentation (quality and completeness).
6. Standardise the development process.
7. Simplify program maintenance.
8. Promote reusability of modules and documentation.
9. Improve software probability across environments.

4.18.1 Components of CASE tool

- Fig. 4.18.1 shows Components of CASE tool

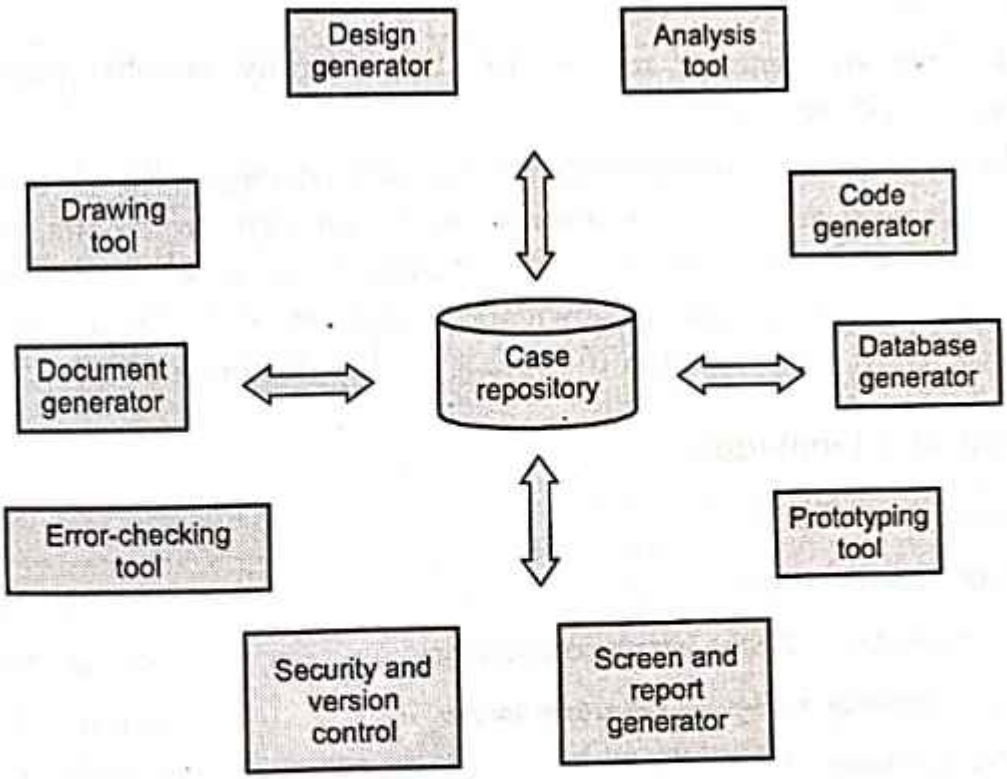

Fig. 4.18.1 Components of CASE tool

- **CASE repository :** It is central component of any CASE tool and also known as the information repository or data dictionary. It is a centralized database and allows easy sharing of information between tools and SDLC activities.
- It is used to store graphical diagrams and prototype forms and reports during analysis and design workflows. It provides wealth of information of project manager and allows control over project.
- CASE repository acts as information repository and data dictionary. In information repository, it combines information about organization's business information and application portfolio and provides automated tools to manage and control access.
- In data dictionary, it is used to manage and control access to information repository and facilities for recording, storing and processing resources. It is useful for cross-referencing.
- Diagraming tools allow you to represent a system and its components visually. It allows higher level processes to be easily decomposed and it can examine processes or data models at high or low level. It allow you to draw DFDs, ERDs, use case diagrams, case diagrams and stepwise refinement in building model.
- **Screen and report generators** used to create, modify and test prototypes of computer displays and reports and identify which data items to display or collect for each screen or report.
- **Analysis tools** are generate reports that help identify possible inconsistencies, redundancies and omissions.
- **CASE documentation generator tools** create standard reports based on contents of repository. It need textful descriptions of needs, solutions, trade-offs, diagrams of data and processes, prototype forms and reports, program specifications and user documentation. High-quality documentation leads to 80 % reduction in system maintenance effort in comparison to average quality documentation.

4.18.2 Benefit and Limitation

Benefits of using CASE tools

1. Automation of tedious tasks.
2. Syntax/completeness checks ensure consistency.
3. Amending/updating achieved far more easily.
4. More rapid development.
5. May guide the use of a methodology.
6. Cheater communication with users.
7. Continuity of development with changing staff.

8. Standardised quality assurance/test procedures.
9. Improvements in system quality.
10. Reduction in defects increases morale.

Limitations of CASE

- Limited flexibility in documentation,
- Development approach limited to fit capabilities of the CASE tool.
- Training and experience required.
- Incomplete coverage of syntax/consistency checks.

4.19 Quality Control and Reliability of Quality Process

- Reliability is the ability of an item to perform a required function under stated conditions for a stated period of time.
- Quality Control (QC) of manufacturing processes obviously makes an essential contribution to the reliability of a product. QC can be considered as an integral part of an overall reliability programme. Reliability is generally concerned with failures during the life of a product.
- Quality control is a deliberate and planned activity in order to determine the quality of a product with a view to accepting it as such. If it does not satisfy these requirements, then appropriate remedial measures are taken to correct the process or activity.

4.19.1 Quality Management System Models

- Software quality management is a management process that aims to develop and manage the quality of software in such a way so as to best ensure that the product meets the quality standards expected by the customer while also meeting any necessary regulatory and developer requirements, if any.
- Software quality managers require software to be tested before it is released to the market and they do this using a cyclical process-based quality assessment in order to reveal and fix bugs before release. Their job is not only to ensure their software is in good shape for the consumer but also to encourage a culture of quality throughout the enterprise.
- Quality management system helps coordinate and direct an organization's activities to meet customer and regulatory requirements and improve its effectiveness and efficiency on a continuous basis.

- Benefits are as follows :
  a) Defining, improving and controlling processes
  b) Reducing waste
  c) Preventing mistakes
  d) Lowering costs
  e) Facilitating and identifying training opportunities
  f) Engaging staff
  g) Setting organization-wide direction
  h) Communicating a readiness to produce consistent results

4.19.2 Complexity Metrics and Customer Satisfaction

- IEEE definition of software quality metrics :
  1. A quantitative measure of the degree to which an item processes a given quality attribute.
  2. A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software processes a given quality attribute.
- Software quality, by definition, is the degree to which software processes a desired combination of attributes. Therefore, to improve system quality, we must focus our attention on software-quality attributes.

What are Metrics ?

- Software process and project metrics are quantitative measures. They are a management tool.
- They offer insight into the effectiveness of the software process and the projects that are conducted using the process as a framework.
- Basic quality and productivity data are collected. These data are analyzed, compared against past averages and assessed.
- Classification by phases of software system :
  1. Process metrics : Metrics related to the software development process
  2. Maintenance metrics : Metrics related to software maintenance
  3. Product metrics : Metrics related to software artifact
- Product metrics describe the characteristics of the product such as size, complexity, design features, performance and quality level. Process metrics can be used to improve software development and maintenance.

- Classification by subjects of measurements
  1. Quality
  2. Timetable
  3. Effectiveness
  4. Productivity

Customer satisfaction :

- **Customer satisfaction is measured by customer survey data. Following scale is used for that purpose.**
  a) Very satisfied b) Satisfied
  c) Neutral d) Dissatisfied e) Very dissatisfied
- These scales are changed according to organizations. Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis.

4.20 International Quality Standards : ISO

- A quality management system defines and establishes an organization's quality policy, objectives and procedures. Quality management system standards do not provide specific quality indicators or describe ways of achieving them, since these will be different in every situation. Instead, they provide generic frameworks and general principles that can be applied to any organization, of any size, in any industry.
- Quality management principles are a set of fundamental beliefs, norms, rules and values that are accepted as true and can be used as a basis for quality management.
- High levels of quality are essential to achieve company business objectives. Quality does not only relate solely to the end products and services a company provides but also relates to the way the company employees do their job and the work processes they follow to produce products or services.
- The work processes should be as efficient as possible and continually improving. Company employees constitute the most important resource for improving quality. Each employee in all organizational units is responsible for ensuring that their work processes are efficient and continually improving.
- Certification standards and assessment standards both are different in content. Quality management should be separate from project management to ensure independence.

- The benefits of using standards :

1. The ability to apply methodologies and procedures of the highest professional level.
2. Better mutual understanding and coordination among development teams but especially between development and maintenance teams.
3. Greater cooperation between the software developer and external participants in the project.
4. Better understanding and cooperation between suppliers and customers, based on the adoption of standards as part of the contract.

Scope of quality management standards - certification standards

1. Enable a software development organization to demonstrate consistent ability to assure acceptable quality of its software products or maintenance services. Certification is granted by an external body.
2. Serve as an agreed-upon basis for customer and supplier evaluation of the supplier's quality management system. It accomplished by performance of a quality audit by the customer.
3. Support the organization's efforts to improve its quality management system through compliance with the standard's requirements.

4.20.1 ISO 9001 and ISO 9000-3

ISO 9000

- ISO 9000 is an international set of standards for quality management. It is applicable to a range of organizations from manufacturing to service industries.
- ISO 9000 has a broad scope : Hardware, software, processed materials and services. The standard outlines the basic elements of a good quality management system. These elements are good business practice.
- ISO 9001 is applicable to organizations which design, develop and maintain products. ISO 9001 is a generic model of the quality process must be instantiated for each organization.
- ISO 9000-1 provides guidelines for selection and use of the ISO 9000 standards.
- The ISO 9004 series provide guidance for quality management, i.e. for the design, implementation and improvement of a quality system. A quality system is a set of 'organizational structures, procedures, processes and resources needed to implement quality management.
- ISO 9001, 9002 and 9003 are models for external quality assurance. They specify a set of requirements.

- ISO 9000 and ISO 9004 are guidelines for quality management and are not Mandatory for certification.
- ISO 9001, ISO 9002 and ISO 9003 are quality system standards.
- ISO 9001 is the broadest standard and provides a model for design, development, production, installation and servicing.
- ISO 9002 is limited to production, installation and servicing.
- ISO 9003 is further limited to inspection and testing.
- A company should first use ISO 9000 to design and to implement a quality system. Once the quality has been installed, the company may use the quality assurance models of ISO 9001, ISO 9002 or ISO 9003 to demonstrate the adequacy of the quality system.
- ISO 9002 is used for production and installation. It is the standard that governs the manufacture of a product. It is designed to ensure conformance to production and installation methods.
- ISO 9003 is the standard directed at the final test and inspection of products. The standard pre-assumes an extensive quality control function and specifies what is needed for conformance to requirements.
- ISO 9004 is for internal use only and lists the components that compose quality systems.

Why ISO 9000 certification ?

1. Better organizational definition
2. Greater quality awareness
3. Better documentation of processes
4. Increased control of operations
5. Ongoing analysis of and solution to problems
6. Positive cultural change
7. Improved customer satisfaction and increased market opportunities.

ISO 9000 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- External body may certify that an organisation's quality manual conforms to (ISO 9000) standards.
- Customers are, increasingly, demanding that suppliers are ISO 9000 certified.

4.20.2 ISO 9000-3 Quality Management System Principles

1. Customer focus
2. Leadership
3. Involvement of people
4. Process approach
5. System approach to management
6. Continual improvement
7. Factual approach to decision making
8. Mutually supportive supplier relationships

Fig. 4.20.1 The eight quality management principles

- The eight principles of quality management can be used by senior management as a framework to guide their organizations towards improved performance.

Principle	Functions	Benefit
Customer focus	• Organizations depend on their customers and therefore should understand current and future customer needs. • Meet customer requirements and strive to exceed customer expectations.	1. Increased revenue and market share. 2. Increased effectiveness in the use of the organization's resources. 3. Improved customer loyalty leading to repeat business.

Leadership	• Leaders establish unity of purpose and direction of the organization. • They should create and maintain the internal environment in which people can become fully involved in achieving the organization's objectives.	1. People will understand and be motivated towards the organization's goals and objectives. 2. Mis-communication will be minimized.
Involvement of people	• People at all levels are the essence of an organization and their full involvement enables their abilities to be used for the organization's benefit.	1. Motivated, committed and involved people within the organization.
Process approach	• A desired result is achieved more efficiently when activities and related resources are managed as a process.	1. Lower costs and shorter cycle times through effective use of resources. 2. Improved, consistent and predictable results
System approach to management	• Identifying, understanding and managing interrelated processes as a system contributes to the organization's effectiveness and efficiency in achieving its objectives.	1. Ability to focus effort on the key processes. 2. Providing confidence to interested parties as to the consistency, effectiveness and efficiency of the organization.
Continual improvement	• Continual improvement of the organization's overall performance should be a permanent objective of the organization.	1. Flexibility to react quickly to opportunities. 2. Performance advantage through improved organizational capabilities.
Factual approach to decision making	• Effective decisions are based on the analysis of data and information.	Increased ability to review, challenge and change opinions and decisions.
Mutually supportive supplier relationships	• An organization and its suppliers are interdependent and a mutually beneficial relationship enhances the ability of both to create value.	1. Increased ability to create value for both parties. 2. Optimization of costs and resources.

4.20.3 ISO 9000-3 Requirements

- The ISO 9000-3 includes about 20 requirements that relate to various aspects of software quality management classified into the following five groups :
  1. Quality management system
  2. Management responsibilities

3. Resource management
4. Product realization
5. Measurement, analysis and improvement

Requirements	Functions
Quality management system requirements	1. Establish a quality management system 2. Document the quality management system
Management responsibilities	1. Management commitment 2. Customer focus 3. Quality policy 4. Planning 5. Administration 6. Management review
Resource management	1. Provision of resources 2. Human resources 3. Facilities 4. Work environment
Product realization	1. Planning of realization processes 2. Customer-related processes 3. Design and/or development 4. Purchasing 5. Production and service operations 6. Control of measuring and monitoring devices
Measurement, analysis and improvement	1. Planning 2. Measurement and monitoring 3. Control of nonconformity 4. Analysis of data 5. Improvement

4.20.4 Certification According to ISO 9000-3

- The ISO 9000-3 certification process verifies that an organization's software development and maintenance processes fully comply with the standard's requirements.
- The objective of the International Standards Organization 9000 series of standards is to certify that an organization has quality manufacturing processes. Thus, if a supplier wanted to demonstrate competence of products, it would make application to demonstrate that it meets the ISO 9000 standards.

- **Documentation requirements :** The ISO 9000 audit is heavily focused on evaluation of documentation. Four tiers of quality system documentation are required :

1. First tier-the quality manual.
2. Second tier-quality management procedures (i.e., core procedures).
3. Third tier-area work instructions (i.e., standard operating procedures, test methods, calibration methods).
4. Fourth tier-forms, records, books and files.

Certification process :

1. The company first implements the control and documentation procedures outlined in the series.
2. It then involves a thorough audit by an independent certification organization that is licensed to register quality systems by an accreditation body.
3. Upon compliance, it receives a registration certificate and its name is included in a published directory of registered suppliers.
4. The systems will be continually verified by the registrar in periodic surveillance and full audits are conducted every few years.

Development of the organization's SQA system :

- Organization first development of a quality model and SQA procedures. Second steps is to development of other SQA infrastructure like staff training, preventive and corrective actions procedures, configuration management services and documentation and quality record control.
- Lastly the development of a project progress control system.

Implementation of the organization's SQA system :

- Setting up a staff instruction program
- Leaders and managers are expected to follow up and support the implementation efforts made by their units.
- Internal quality audits are carried out to verify the success in implementation.
- The findings will determine of whether the organization has reached a satisfactory level of implementation.

Problems with certification

1. Costs - application and maintenance
2. Time - application and maintenance

3. Level of internal expertise
4. Executive commitment
5. Selection of registration.

4.21 Capability Maturity Models (CMM)

- Capability maturity model is developed by the software engineering institute of the carnegie mello university. It is a framework that describes the key elements of an effective software process.
- The application of process management and quality improvement concepts to software development and maintenance. A guide for evolving toward a culture of engineering excellence. A model for organizational improvement.
- In 1991, SEI released the capability maturity model for software version 1.0.
- In 2001, SEI released the capability maturity model - integrated, superseding the CMM for software.
- It presents a minimum set of recommended practices that have been shown to enhance a software development and maintenance capability. It defines the expectation (the "what"). Without overly constraining the implementation (the "how").
- The key to CMM model is it is designed to provide good engineering and organizational mnanagement practices "for any project in any environment".

4.21.1 Principles of CMM

- Quantitative management methods increase the organization's capability to control the quality and improve the productivity.
- Application of the five-level capability maturity model that enables to evaluate the achievements and determine the efforts needed to reach the next capability.
- Generic process areas that define the "what" - not "how" enables the model's applicability to a wide range of implementation organizations :
  a. It allows use of any life cycle model.
  b. It allows use of any design methodology, development tool and programming language.
  c. It does not specify any particular documentation standard.
- The CMM is called a framework or model rather than a standard. The CMM includes the following components

  a. Maturity levels b. Process capabilityc

c. Key process areas
d. Goals
e. Common features
f. Key practices

4.21.2 CMM Levels

- The CMM is organized into five levels of organizational maturity, with each level representing a higher evolutionary stage of process capability and a progressively greater likelihood of producing quality software.
- Maturity level indicates level of process capability :
  a. Initial
  b. Repeatable
  c. Defined
  d. Managed
  e. Optimizing

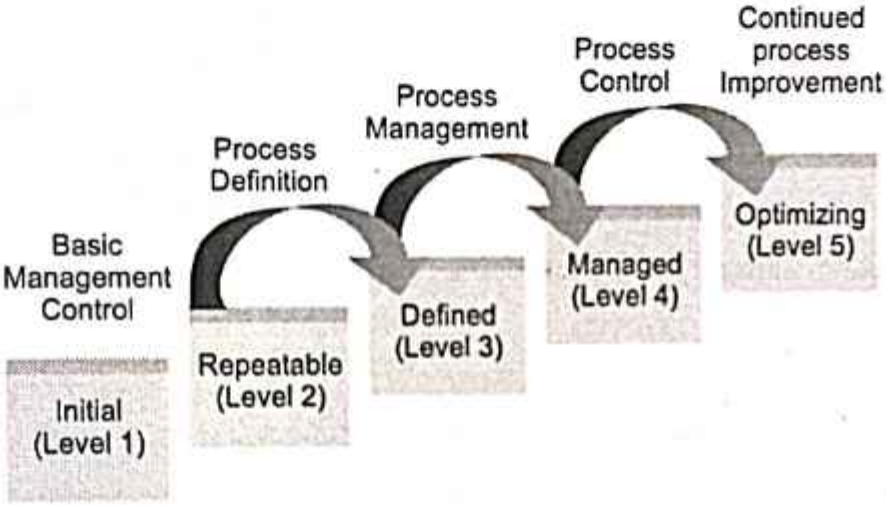

Fig. 4.21.1 CMM level

- These levels are described in terms of key process areas. A key process area is a group of related activities considered important for an organization functioning at the appropriate process maturity level.

Level	Focus	Description
Level 5 - optimizing	Continuous process improvement	• The key process area for level 5 is to maintain continuous improvement and optimize existing processes. • The performing organization at this level will be equipped to proactively address the strengths and weaknesses of the business processes and software engineering practices.

Level 4 - managed	Product and process quality	• The key process area to level 4 focuses on the process controls in place to measure quality. • Detailed measures of the management and software processes are collected and used to identify and correct issues with process performance. • A managed process for these continuous improvements helps to establish and maintain a high performing organization.
Level 3 - defined	Engineering process	• The software process for both management and engineering activities is documented, standardized and integrated into a standard software process for the organization. • All projects use an approved, tailored version of the organizations standard software process for developing and maintaining software. • The key process area to level 3 is defined process management
Level 2 - repeatable	Project management	• Basic project management processes are established to track cost, schedule and functionality.
Level 1 - initial	No focus	• This level has no key process areas. • Project success primary depends on individuals and their heroics :

- The CMM is not prescriptive; it does not tell an organization how to improve. The CMM describes an organization at each maturity level without prescribing the specific means for getting there.

Overall benefits of CMM :

a. Defect rates have dropped

b. Defect detection occurs earlier

c. User requirements are documented, controlled and managed

d. Estimating improves and becomes more precise

e. Risk management is a practice

f. Development processes remain agile!

4.21.3 CMMI

- The Capability Maturity Model Integration (CMMI) project was initiated in 1997 by the DoD and National Defense Industrial Association (NDIA) to : Establish a framework to integrate current and future models and build an initial set of integrated models.
- CMMI was developed independently of the CMM in the mid-1990s. It is fully dependent on the original CMM in form and structure.
- CMMI provides guidance for improving the development, acquisition and maintenance of software products and services.
- CMMI maturity levels are as follows :
  a. Initial
  b. Managed (in the CMM, this level is known as repeatable)
  c. Defined
  d. Quantitatively managed (in the CMM, this level is known as managed)
  e. Optimizing
- CMMI has the following four common features :
  1. Commitment to perform
  2. Ability to perform
  3. Directing implementation
  4. Verifying implementation
- CMMI will helps in following area :
  1. Improve delivery of promised performance, cost, and schedule.
  2. Collaborate with external stakeholders and manage their expectations.
  3. Provide competitive world-class products and services.
  4. Implement an integrated, enterprise business and engineering perspective
  5. Master system-of-systems evolutionary development complexity.
  6. Use common, integrated and improving processes for systems and software.
  7. Implement proactive program management techniques.
  8. Develop project leaders who look ahead and not over their shoulder.
  9. Develop a staff that uses best practices to cope with changing development, technology and customer environments.
  10. Enable staff members to move between projects and still use the same processes.
  11. Create and improve processes that adapt to a changing business environment.

CMMI maturity levels :

Maturity level	Process characteristics	Behaviors
5. Optimizing	Focus is on continuous quantitative improvement.	Focus on "fire prevention", improvement anticipated and desired and impacts assessed.
4. Quantitatively managed	Process is measured and controlled.	Greater sense of teamwork and inter-dependencies.
3. Defined	Process is characterized for the organization and is proactive.	Reliance on defined process. People understand, support and follow the process.
2. Managed	Process is characterized for projects and is often reactive.	Over reliance on experience of good people - when they go, the process goes.
1. Initial	Process is unpredictable, poorly controlled and reactive.	The process performance may not be stable and may not meet specific objectives such as quality, cost and schedule, but useful work can be done.

CMMI components :

- Within each of the 5 maturity levels, there are basic functions that need to be performed - these are called Process Areas (PAs).
- For maturity level 2 there are 7 process areas that must be completely satisfied.
- For maturity level 3 there are 11 process areas that must be completely satisfied.
- Given the interactions and overlap, it becomes more efficient to work the maturity level 2 and 3 issues concurrently.
- Within each PA there are goals to be achieved and within each goal there are practices, work products, etc. to be followed that will support each of the goals.

4.21.4 Comparison between ISO 9000 and CMM

ISO 9000	CMM
ISO 9000 series of standards, developed by the International Standards Organization (ISO), share a common concern with quality and process management.	CMM developed by the software engineering Institute of carnegie mellon university
Minimum requirements with implied continuous improvements.	Explicit continuous quality improvement.

Not specific to any one industry or service	Software focus.
Continual audits required	No follow up audits
Registration document	No documentation
With ISO 9001, once you are certified, your challenge is only to maintain certification.	The challenge here is to maintain and continuously improve.
ISO 9001 certification requires auditors, which places emphasis on opinions of outsiders whose capabilities may be unknown or marginal.	CMM can be used as a self-assessment tool.

❑❑❑

UNIT V

5 Automation Testing Tools / Performance Testing Tools

Syllabus

Automation Testing : What is automation testing, Automated Testing Process, Automation Frameworks, Benefits of automation testing, how to choose automation testing tools. Selenium Automation Tools : Selenium's Tool Suite- Selenium IDE, Selenium RC, Selenium Web driver, Selenium Grid. Automation Tools : SoapUI, Robotic Process Automation (RPA), Tosca, Appium.

Contents

5.1 Automation Testing

- Automation testing is a software testing technique that uses specialized automated testing software tools to execute a test case suite. In contrast, manual testing is performed by a human sitting in front of a computer and carefully executing the test steps.
- Automation testing software can feed test data to test-under-test, compare expected and actual results and generate detailed test reports. Software test automation demands a significant investment of money and resources.
- The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate **Manual Testing** altogether.

5.1.1 What Tests should be Automated ?

Before creating a test automation strategy, let's see which tests are most practical for automation :

A] Regression testing : Regression suites are growing all the time and the same variables need to be filled in multiple times to ensure that new features don't disrupt old functions. This can be easily automated.

B] Testing of complex functionalities : Automate all tests that require complex calculations and are prone to human error.

C] Smoke testing : Run automated suites to check the quality of key functions. This saves time by providing a quick analysis of whether a build needs more in-depth testing.

D] Data-driven testing : Automate tests to validate operations that need to be tested repeatedly with multiple data sets.

E] Performance testing : Automate tests that monitor software performance under different conditions. Doing this manually is very laborious and time consuming.

F] Functional testing : Each time a developer submits a PR, functional testing should be performed quickly and feedback provided immediately. This is impossible to achieve without automation, especially as organizations grow.

5.2 Automated Testing Process

1] Test automation feasibility analysis - The first step is to check whether the application can be automated or not. Not all applications can be automated due to limitations.

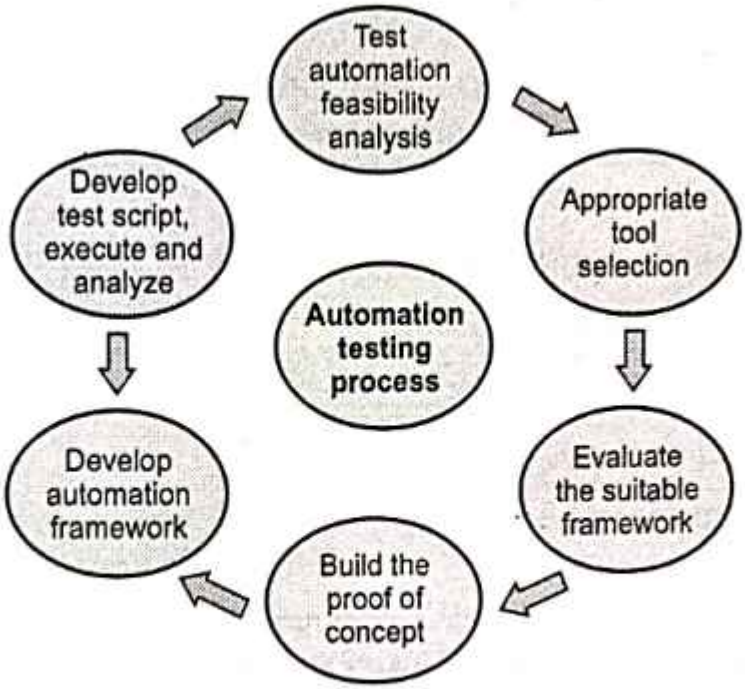

Fig. 5.2.1 Automated Testing Process

2] **Appropriate tool selection** - The next important step is the selection of equipment. It depends on the technology on which the application is built, its features and usage.

3] **Evaluate the suitable framework** - After choosing the tool, the next step is to choose a suitable framework. There are different types of frameworks and each framework has its own importance.

4] **Build proof of concept** - A Proof Of Concept (POC) is developed in an end-to-end scenario to evaluate whether the device can support automation of the application. It is implemented in an end-to-end scenario, which ensures that key functions can be automated.

5] **Develop automation framework** - After the POC is built, framework development is done, which is a critical step for the success of any test automation project. The framework should be built after carefully analyzing the technology used by the application and its key features.

6] **Develop test script, execute and analyze** - Once the script development is complete, the scripts are executed, the results are analyzed and any errors are logged. Test scripts are usually version controlled.

5.3 Automation Framework

In order to facilitate automated testing of any application, the automation framework is a collection of tools and procedures rather than a single one. It integrates various functions like libraries, test data and various reusable modules.

5.3.1 Test Automation Framework Types

There are six common types of test automation frameworks, each with their own architecture and differing benefits and disadvantages.

1. Linear automation framework
2. Modular based testing framework
3. Library architecture testing framework
4. Data-driven framework
5. Keyword-driven framework
6. Hybrid testing framework

1) Linear automation framework

- The linear automation framework is commonly used in the testing of small applications. This framework is also called as a Record and playback framework.
- **Pros :** There is no need to write custom code, so expertise in test automation is not necessary.
- **Cons :** Since the data is context sensitive in the test script, it is impossible to run the test cases again with other sets of data. If the data has changed, you must make certain adjustments.

2) Modular driven framework

- In this structure, the tester may write test scripts independently by segmenting the entire programme into smaller parts in accordance with customer requirements.
- **Pros :** A framework that is powered by modules guarantees that scripts are divided, which makes maintenance and scalability easier. Independent test scripts can be written.
- **Cons :** It takes more time to analyze the test cases and find reusable flows using the modular driven structure.

3) Behavior driven development framework

- The goal of the behavior driven development platform is to build a platform that encourages active participation from all users, including developers, testers, business analysts, etc. Additionally, it improves cooperation on your project between the developers and testers.
- **Advantages :** When using behavior-driven testing, test requirements may be written in plain, non-technical language.
- **Cons :** Working with this framework requires both significant technical know-how and prior expertise with test-driven development.

4) Data-driven testing framework

- The test script often reads test data from external files such Excel files, text files, CSV files, ODBC sources and DAO objects and loads that data into variables. We may write test automation scripts using the data-driven architecture by passing various test data sets.
- **Benefits** : It requires less scripts overall. This allows for the testing of many situations with minimal code.
- **Cons** : To fully leverage the architecture of this framework, you will require a skilled and experienced tester who is knowledgeable in a variety of programming languages.

5) The keyword-driven testing framework

- Table-driven testing is another name for the keyword-driven testing framework. This framework is only appropriate for little applications or projects. The keywords listed in the project's excel sheet serve as the basis for the test automation scripts that are executed.
- **Advantages** : Because a single term may be used in several test scripts, code reuse is possible.
- **Cons** : Setting up the structure requires a significant initial investment and is difficult and time-consuming.

6) The hybrid test automation framework

- To combine the advantages of keyword-driven and data-driven frameworks, we use hybrid frameworks.
- Advantages of various related frameworks are utilised by this type.
- **Cons** : A hybrid testing framework requires more automation work because tests are completely written.

5.4 Benefits of Automation Testing

1. Enhanced results

- Automation testing saves a lot of time even when considering complex and large systems. This allows for repeatable testing, providing better and faster results with significantly less effort and less time

2. Swifter feedback system

- Automation testing is very critical in the validation phase of any software project. This significantly increases communication between developers, designers and product marketers, and provides room for immediate correction of possible defects, thereby increasing the efficiency of the development team.

3. Brand enhancement

- The effectiveness of testing always depends on the quality of the test data used. Because generating relevant and quality test data takes a lot of time, testing is often performed on replicas of live databases.
- Automation solutions allow you to reuse your data over and over again. This saves a lot of cost from project management and project maintenance perspective.
- The best aspect of automated testing is that it adds value to all stakeholders attached to it. Automated testing systems not only enhance system capability but also pave the way for digital innovation and revolution.
- This not only improves the brand name but also increases the brand recall value, thus ensuring greater customer retention. Because of automation testing, intractable problems have permanent solutions.

4. Cost-effective

- Although the initial investment required for automation testing is high, it saves the company a lot of money in the long run. This is mainly due to the reduced time required to conduct the tests.
- It also contributes to high quality of work as there is no chance of negligence or human error. This reduces the need to fix defects in the post-release phase, thereby saving a large amount of project cost.

5. Efficiency testing

- Testing is one of the most important parts of the entire application development cycle. The most attractive part of automation testing is that it can be left virtually unattended. This leaves plenty of room for monitoring the results at the end of the process. This allows increasing the overall efficiency of the application.

6. Increase in coverage area

- Through the use of automation testing, more tests can be allocated to any application. This leads to higher testing coverage and reduced software defects. It also allows room for testing more features and complex applications. However, a manual testing scenario requires a large team with heavy time constraints for the same task.

7. Detailed testing

- All testers have different testing approaches with different focus areas depending on their exposure and expertise. With the help of automation, all areas of inspection are equally focused, thus ensuring the best quality of the final product with more emphasis on each aspect of the product.

- Automation testing is known for its atom - level approach and is therefore considered error - free.

8. Reusability

- Test automation is iterative in nature due to the nature of its test automation cases. In addition to easy setup configuration, it gives software developers an opportunity to evaluate the program's response. Automated test cases are fully reusable and can therefore be used to test any aspect of the code depending on importance and with different approaches.

9. Earlier detection of defects

- Automation testing software records defects, so it makes testing teams much easier. This makes it relatively easy for the development and support team to consider defects together and provide quick output.
- Accelerates the overall development speed of the project by ensuring proper functioning across relevant sectors. The earlier any fault is identified, the better and less expensive it is to fix and deploy

10. Time to market

- Test automation significantly helps in reducing the time-to-market launch of an application. Automation testing allows consistent and consistent execution of test cases. Test library execution runs much faster and longer after automation.

5.5 How to Choose Automation Testing Tools

- Success in any test automation depends on identifying the right tool for automation. Choosing the "right" testing tool for your project is the best way to achieve the project target.

5.5.1 Tool Selection Process

- To select the most suitable testing tool for the project, the Test Manager should follow the below tools selection process

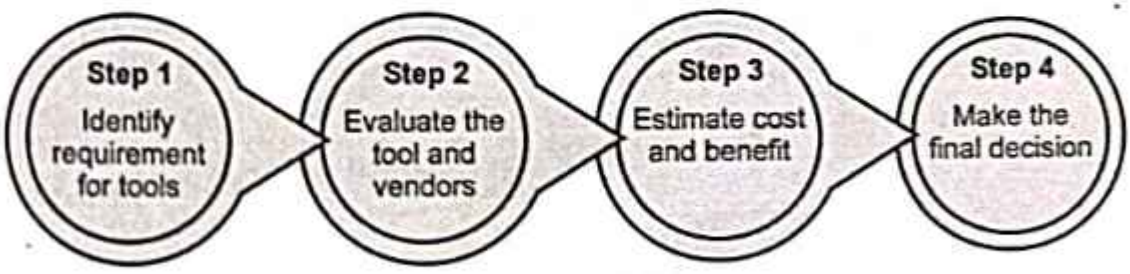

Fig. 5.5.1 Tool selection process

Step 1) Identify the requirement for tools

- How can you choose a testing tool if you don't know what you're looking for?
- To precisely identify your test tool requirements. All requirements should be documented and reviewed by the project teams and management board.

Step 2) Evaluate the tools and vendors

- After baselining the requirement of the tool, the Test Manager should
  - Analyze commercial and open source tools available in the market based on project requirement.
  - Create a tool shortlist that best meets your criteria.
  - One factor you should consider is the vendors. You should consider the vendor's reputation, after-sales support, tool update frequency, etc. while making your decision.
  - Evaluate the quality of the tool by taking trial usage and launching a pilot. Many vendors make trial versions of their software available for download

Step 3) Estimate cost and benefit

- To ensure that the test tool is beneficial to the business, the test manager needs to balance the following factors :
- A **cost-benefit analysis** should be performed before acquiring or building a tool.

However, after discussing with the software vendor, you found that the cost of this tool is too high compare to the **value** and **benefit** that it can bring to the teamwork. In such a case, the **balance** between cost and benefit of the tool may affect the final decision.

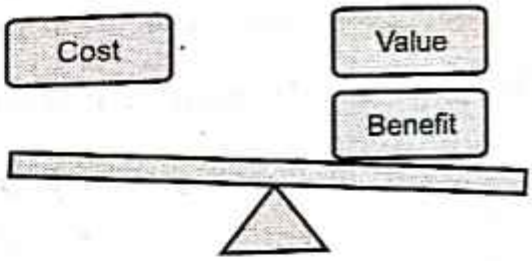

Fig. 5.5.2 A cost-benefit analysis

Step 4) Make the final decision

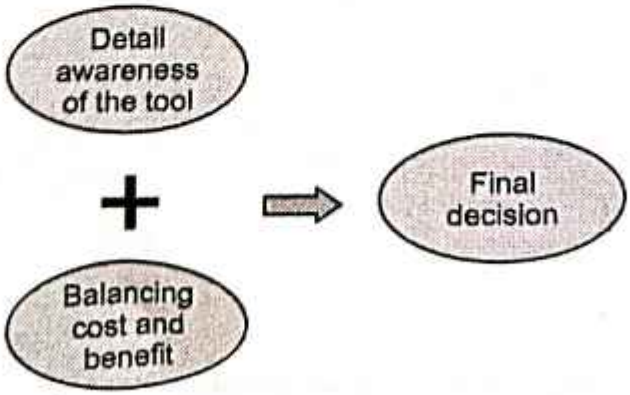

Fig. 5.5.3 Final Decision

- To make the final decision, the test manager must have :
  - A strong awareness of the device. This means that you should understand what the strong points and weak points of the device.
  - **Balance** cost and benefit.
  - Even if you spend hours reading the software manual and vendor information, you need to test the device in your actual operating environment before purchasing a license.
  - You should meet with the project team and consultants to gain in-depth knowledge of the tool.
  - Your decision may adversely affect the project, testing process and business objectives; You should spend a good amount of time thinking about it.

5.5.2 Type of Test Tools

There're many types of test tool, which test manager can consider when selecting the test tools.

1. Open-source tools

- Open source tools are the program wherein the source code is **openly** published for use and/or modification from its original design, **free of charge.**
- Open-source tools are available for almost any phase of the testing process, from test case management to defect tracking. Compared to commercial tools, open source tools may have fewer features.

2. Commercial tools

- Commercial tools are the software which are produced for sale or to serve commercial purposes.
- Commercial tools have more support and more features from a vendor than open-source tools.

3. Custom tools

- In some testing project, the testing environment and the testing process has special characteristics. No open-source or commercial tool can meet the requirement. Therefore, the test manager has to consider the development of the custom tool.

5.6 Selenium

- Selenium is a free (open source) automated testing framework used to validate web applications across different browsers and platforms. You can use multiple programming languages like Java, C# and Python to create selenium test scripts. Testing performed using Selenium Testing Tool is commonly called Selenium

Testing. Selenium software is not just a tool, but a suite of software, each piece catering to the different Selenium QA testing needs of an organization. Here is the equipment list

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver
- Selenium Grid

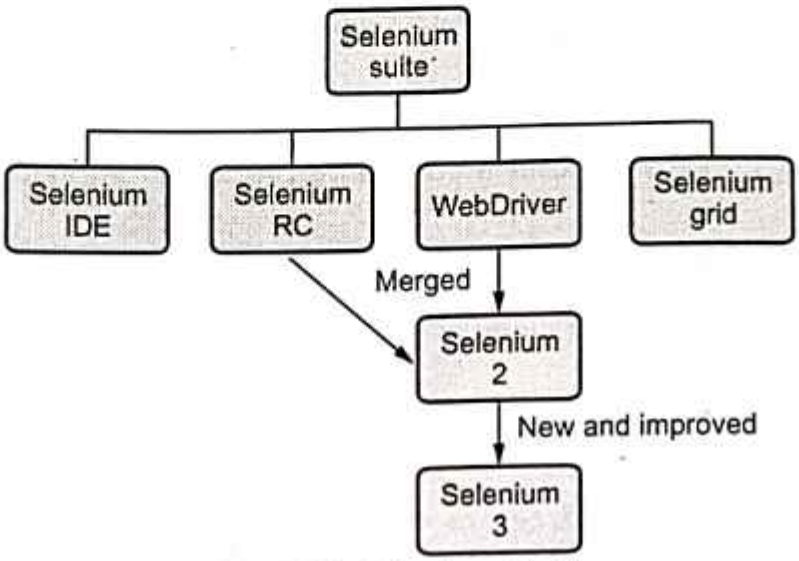

Fig. 5.6.1 Selenium Suite

5.6.1 Introduction to Selenium

- At the moment, Selenium RC and WebDriver are merged into a single framework to form **Selenium 2.** Selenium 3, by the way, refers to Selenium RC.

5.7 Selenium Tool Suits

1. Selenium IDE

- The Selenium Integrated Development Environment (IDE) is the simplest framework in the Selenium suite and one of the easiest to learn. This is a Firefox plugin that you can install as easily as you can with other plugins.
- However, due to its simplicity, Selenium IDE should only be used as a prototyping tool. If you want to create more advanced test cases, you need to use Selenium RC or WebDriver.

2. Selenium Remote Control (Selenium RC)

- Selenium RC was the **flagship testing framework** of the whole Selenium project for a long time. This is the first automated web testing tool that **allowed users to use a programming language they prefer.** As of version 2.25.0, RC can support the following programming languages :

- Java
- C#
- PHP
- Python
- Perl
- Ruby

3. Selenium WebDriver

- WebDriver proves itself better than Selenium IDE and Selenium RC. It implements a more modern and sustainable approach to automating browser operations. WebDriver, unlike Selenium RC, does not rely on JavaScript for Selenium automation testing.
- It controls the browser by directly communicating with it.
- The supported languages are the same as those in Selenium RC.
  - Java
  - C#
  - PHP
  - Python
  - Perl
  - Ruby

4. Selenium Grid

- Selenium Grid is a tool used in conjunction with Selenium RC to run parallel tests on different machines and different browsers at the same time. Parallel execution means running multiple tests simultaneously.
- Features :
  - Enables **simultaneous running of tests in multiple browsers and environments.**
  - **Saves time** enormously.
  - Utilizes the **hub-and-nodes** concept. The hub acts as a central source of Selenium commands to each node connected to it.

5.8 Automation Testing Tool

Automation testing tools are applications designed to verify function and/or non-functional requirements via automated test scripts.

5.8.1 SOAP UI

- SOAP UI is the leading open source cross - platform API testing tool
- SOAPUI allows testers to execute automated functional, regression, compliance and load tests on different Web API.
- SOAPUI supports all the standard protocols and technologies to test all kinds of API's.
- SOAPUI interface is simple that enables both technical and non-technical users to use seamlessly.

5.8.1.1 Why use SOAPUI ?

SOAPUI is not just a functional **Api Testing** tool but also lets us perform non-functional testing such as performance and security test.

Let us discuss the 5 important features of SOAPUI

1) Functional testing

- A powerful tool allows testers to write Functional API Tests in SoapUI
- Supports Drag - Drop feature which accelerates the script development
- Supports debugging of tests and allows testers to develop data driven tests.
- Supports Multiple Environments - Easy to switch between QA, Dev and Prod Environments
- Allows advanced scripting (tester can develop their custom code depending on the Scenario)

2) Security testing

- Has the capability to perform a complete set of vulnerability scan.
- Prevents **SQL** Injection to secure the databases
- Scans for Stack overflows that are caused by documents huge in size
- Scans for Cross Site Scripting, which usually occurs when service parameters are exposed in messages.
- Performs fuzzing scan and boundary scan to avoid erratic behavior of the services.

3) Load testing

- Distribute the load tests across any number of loadUI Agents.
- Simulate high volume and real - world load testing with ease.
- Allows advanced custom reporting to capture performance parameters.
- Allows End-to-End system performance monitoring

4) Supported Protocols / Technologies

- SoapUI has the most comprehensive Protocol Support

5) SOAP - INTEGRATION with other automation tools

- SoapUI integrated very well with popular tools
- **Maven**
  Apache Maven is a software project management tool that can manage a project's production, reporting and documentation from a central repository. Maven can also run SOAPUI tests within a Maven build using simple commands.
- **HUDSON**
  Hudson, a Java-based continuous integration tool, integrates with tools like CVS, Subversion, Git, Perforce, ClearCase, and RTC. SOAPUI integrates with Hudson, which helps us quickly find bugs for every developer commit.
- **JUnit**
  JUnit is a unit testing framework built in Java that can control the flow of tests from SOAPUI.
- **Apache - Ant**
  Apache Ant is a Java library, a command - line tool that helps build software. Using the SOAP UI's command line, we can run tests within an ANT automated build.

5.8.2 Robotic Process Automation

- RPA stands for Robotic Process Automation. It is the technology used for software tools that automate manual, rule-based, or repetitive human tasks. Typically, it's like a bot that does that kind of work at a much higher rate than a human.
- These RPA software bots never sleep or make mistakes and can interact with in-house applications, websites, user portals, etc. They can log into applications, enter data, open emails and attachments, calculate and complete tasks and then log out.
- The term robotic process automation conjures up images of physical robots performing human physical tasks such as loading or unloading heavy items from a vehicle or cleaning the house.
- However, in reality, the picture is quite different. The word 'robot' in 'RPA' is not a physical robot but a virtual system that helps automate repetitive manual computing or business process tasks.

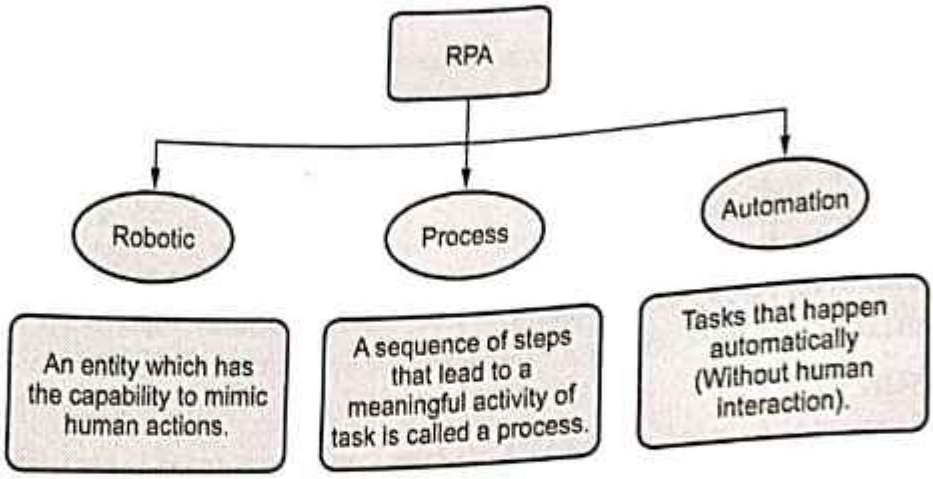

Fig. 5.8.1 Robotic process automation

5.8.2.1 RPA Technologies

It can be divided into three categories :

1. **Probots**

   These are the bots that follow simple, repeatable rules to process data.

2. **Knowbots**

   These are the bots that search user - specified information from the internet and respond to the user.

3. **Chatbots**

   These are the bots that act and respond as virtual agents. They reply to customer queries in real - time.

5.8.2.2 Benefits of RPA

Robotic Process Automation technology provides the following benefits :

1. Cost savings

- RPA helps organizations save a lot of money as it is cheaper than hiring an employee to do the same set of tasks.

2. Less error

- RPA works on standard logic and is not boring, distracting or exhausting. Hence, the chance of making errors is somewhat reduced, i.e., less rework and a better reputation for efficiency.

3. Faster processing

- RPA works faster than human employees as computer software does not require breaks, meals, rest etc. and can perform repetitive tasks without rest. With RPA, processing times become predictable and consistent, ensuring high - quality customer service across operations.

4. Better regulatory compliance

- RPA software works on logic and data and does only what is required according to the given instructions. Hence, there are minimum chances of non - compliance with standard regulations.

5. Better customer service

- When RPA is implemented in a business, it frees up many employees who can spend time working on customer - related services. This is very beneficial for businesses that receive a lot of customer inquiries. It also results in increased productivity of workers.

6. Auditable and secure

- RPA bots only access data for which they are authorized and create a detailed audit trail of all activity.

5.8.3 Tosca

- TOSCA stands for Topology and Orchestration Specification for Cloud Applications. Tricentis Tosca is a software testing tool used to automate end-to-end testing for software applications. It was developed by Trecentist. It is a software tool for automated execution of functional and regression software testing.
- Besides testing automation functions, TOSCA includes
  - Integrated Test Management
  - The Graphical User Interface (GUI)
  - Command Line Interface (CLI)
  - Application Programming Interface (API)
- The test suite supports the entire lifecycle of the test project. It starts with transferring and synchronizing specifications from the requirement management system.
- TOSCA supports its users in creating efficient test cases on a methodologically sound basis, serves as an executive assistant and summarizes the test results in various reports.
- Tosca are related to **Model-based testing** and **Risk-based testing.**

Model-based testing

- Instead of using scripting for test automation, Tricentis Tosca applies a model - based testing approach and creates a model of the application under test. Technical details about the application under test, test case logic and test data are saved separately and merged together during test execution.

- When a component in the application under test changes, the technical details are updated once in the central model. Because the test cases inherit from this model, the various test cases that test the modified component do not need to be manually modified to reflect the change

Risk - based testing

- Based on the risk assessment of the application under test requirements, Tricentis Tosca uses risk - based test design to propose the most effective test cases and identify the risk contribution of each test case.
- It uses various methods (such as equivalence partitioning, boundary testing and combinatorial methods such as linear expansion) to try to reduce the number of test cases while increasing risk coverage. After executing the tests, the tool aggregates risk coverage from business, technical and compliance perspectives.

5.8.4 Appium

- Appium is an open source automation mobile testing tool which is used to test the application. It is developed and supported by Sauce Labs for automating native and hybrid mobile apps. It is a cross - platform mobile automation tool, which means it allows the same test to be run on multiple platforms. Appium can easily test multiple devices in parallel.
- Mobile applications are in high demand in today's development sector. Nowadays, people are converting their websites into mobile apps. Hence, it is very important to know about mobile software automation testing technology and keep in touch with new technology.
- Appium is a mobile application testing tool that is currently trending in Mobile Automation Testing Technology.
- Appium is used for automated testing of native, hybrid and web applications. It supports automation test on simulators (iOS), emulators (Android) and physical devices (Android and iOS).
- Previously, this tool was mainly focused on iOS and Android applications which were limited to mobile application testing. A few updates ago, Appium announced that it would support desktop application testing for Windows.
- Appium is very similar to the Selenium web driver testing tool. So, Appium is very easy to learn if you already know Selenium WebDriver. Appium doesn't depend on the mobile device OS because it has a framework that converts Selenium WebDriver commands to UI Automator and UI Automation commands for Android and iOS, respectively, depending on the device type rather than the OS type.

- It supports a number of languages with Selenium client libraries, including Java, PHP, Objective C, C#, Python, JavaScript with node.js, Ruby and many more. Selenium is Appium's backend that provides control over Selenium's functionality for testing purposes.

5.8.4.1 Features of Appium

- Appium does not require application source code or library.
- Appium provides a strong and active community.
- Appium has multi - platform support i.e., it can run the same test cases on multiple platforms.
- Appium allows the parallel execution of test scripts.
- In Appium, a small change does not require re - installation of the application.

5.8.4.2 Advantages of Appium

- Appium is a free utility since it was created using open - source software. Installing it is simple.
- The testing of hybrids, native, and web apps may be done automatically.
- Unlike other vulnerability scanning, you can use Appium with automation without adding any more agents to your app. The same software that will be uploaded to the App Store is tested.
- An enhancement to Appium's feature set. Now, in addition to supporting testing of mobile applications, it would also offer testing of Windows desktop applications.
- Cross-platform mobile testing is made possible by the free, open-source programme known as Appium. This implies that you can test across several platforms.

5.8.4.3 Disadvantages of Appium

Along having various benefits and features, Appium also has several downsides, including the following :

- A lack of thorough reporting.
- The tests are a little sluggish since they rely on the remote web driver.
- The fact that Appium uses UI Automator for Android, which only supports Android SDK, API 16, or higher, is not a restriction but rather an overhead. But Appium does not natively support earlier APIs. To support earlier APIs, it makes use of Selendroid, another open-source library.

- In iOS, only one instance (iOS Script) can run on one Mac OS device, which means one test can be executed at a time per Mac. If you want to run your tests on multiple iOS devices at the same time, you need to arrange the same number of Mac machines. But it would be expensive to arrange various Mac machines.

5.9 Performance Testing

- Performance testing is a software testing process used to test a software application's speed, response time, stability, reliability, scalability and resource utilization under specific workloads.
- The main purpose of performance testing is to identify and eliminate performance bottlenecks in the software application. It is a subset of performance engineering and is also known as "perf testing".
- The focus of Performance Testing is checking a software program's
  - Speed - Determines whether the application responds quickly
  - Scalability - Determines maximum user load the software application can handle.
  - Stability - Determines if the application is stable under varying loads

5.9.1 Types of Performance Testing

1. **Load testing** - Checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.
2. **Stress testing** - Involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.
3. **Endurance testing** - Is done to make sure the software can handle the expected load over a long period of time.
4. **Spike testing** - Tests the software's reaction to sudden large spikes in the load generated by users.
5. **Volume testing** - Under volume testing large no. of data is populated in a database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.
6. **Scalability testing** - The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity addition to your software system.

5.10 Performance Testing Process

- The methodology adopted for performance testing may vary widely, but the objective of performance tests is the same. This will help demonstrate that your software system meets certain pre-defined performance criteria. It helps to compare the performance of two software systems. It can also help identify parts of your software system that degrade performance.
- Below is a generic process on how to perform performance testing

. Identify your testing environment -

- Know your physical test environment, production environment and what testing tools are available. Before you begin the testing process, understand the details of the hardware, software and network configurations that will be used during testing.
- This will help testers create more efficient tests. It will also help identify challenges that testers may face during the performance testing process

. Identify the performance acceptance criteria -

- This includes goals and constraints for throughput, response time and resource allocation. It is also necessary to identify project success criteria outside of these objectives and constraints.
- Testers should be empowered to set performance criteria and objectives, because often project specifications do not include a sufficient variety of performance criteria. Sometimes there is nothing. When possible, finding a similar application to compare to is a good way to set performance goals.

1. **Plan and design performance tests** - Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
2. **Configuring the test environment** - Prepare the testing environment before execution. Also, arrange tools and other resources.
3. **Implement test design** - Create the performance tests according to your test design.
4. **Run the tests** - Execute and monitor the tests.
5. **Analyze, tune and retest** - Consolidate, analyze and share test results. Then fine tune and test again to see if there is an improvement or decrease in performance. Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU. Then you may have the consider option of increasing CPU power.

5.11 Performance Test Tools

There are a variety of performance testing tools available in the market. The tool you choose for testing will depend on several factors, such as supported protocol types, license cost, hardware requirements and platform support. Below is a list of popularly used testing tools.

1. **LoadNinja** - We are revolutionizing the way we load test. This cloud-based load testing tool enables teams to record and instantly playback comprehensive load tests without complex dynamic correlation. Teams can increase test coverage and reduce load testing time by 60 %.
2. **HeadSpin** - It offers its users the industry's best performance testing capabilities. Users can optimize their digital experiences with the Headspin platform's performance testing capabilities by identifying and resolving performance issues across applications, devices and networks.

   HeadSpin provides actual, real-world data removing ambiguity from thousands of devices, networks and locations. Users can leverage advanced AI capabilities to automatically identify performance issues in testing before they impact users.
3. **BlazeMeter** - Designed and built by engineers passionate about open source. BlazeMeter provides massive load and performance testing directly from your IDE. Also, see what your user sees on load with integrated UX and load testing. And the best part ? It's all there : performance, functional, scriptless, API testing and monitoring, test data and mock services.
4. **HP LoadRunner** - It is the most popular performance testing tool on the market today. This tool can simulate hundreds of thousands of users and put applications under real-life loads to determine their behavior under expected loads. LoadRunner features a virtual user generator that simulates the actions of real-time human users.
5. **Jmeter** - One of the leading tools used for load testing of web and application servers.

5.11.1 Tools Used For Performance Testing

5.11.1.1 Apache Jmeter

- It is used to test the performance of static and dynamic resources and dynamic web applications. This tool is completely designed in JAVA application to load functional test behavior and measure application performance.
- It is an open source tool that facilitates users and developers to use the source code for the development of other applications.

- It can be used to test the strength of a large load on a server, object or network and group of servers or to explore the full performance under multiple load types. Previously it was used to test web application but now it has been extended to other test functions as well.

Features of JMeter

Below are some essential elements of JMeter :

- This tool supports a user-friendly GUI that is interactive and simple.
- JMeter supports multiple testing approaches such as functional, distributed and load testing.
- Load performance test on multiple types of servers such as Database Server : LDAP, JMS, JDBC, Web Server: SOAP, HTTPS, HTTP, Mail Server : POP3 is incredibly scalable.
- It is platform-independent because it is designed with the help of Java, so it can run on any platform that accepts the JVM, such as Window, Mac, and Linux.

5.11.1.2 LoadRunner

- It is one of the most powerful tools for performance testing and is used to support performance testing for a wide range of protocols, number of technologies and application environments.
- It quickly identifies the most common causes of performance problems. and accurately predict application scalability and capacity.

I. Feature of LoadRunner

- It will support XML; that's why we can easily view and handle the XML data within the test scripts.
- It supports a large range of applications, which will reduce the time to understand and explain the reports.
- With the help of this tool, we can get detailed performance test reports.
- It will reduce the cost of distributed load testing.
- It will provide the operational tool for deployment tracking.
- This tool is used to reduce the cost of software and hardware.

5.11.1.3 LoadNinja

- LoadNinja is powered by SmartBear. With the help of this tool, product teams and test engineer will build the application with more concentration instead of writing load testing scripts.

- We can keep track of user interactions and directly detect and debug performance issues in real-time. It will replace load emulators with real browsers.

5.11.1.4 WebLOAD

- Webload testing tool is used to test the test application with the help of load testing, performance testing and stress testing. The WebLOAD tool combines performance, scalability and integrity into a single process for authenticating web and mobile applications.It will support the multi-protocols such as HTTPS, XML, HTTP and so on, which helps us to control the load of the large number of users.

1. Features of WebLOAD

Following are the most commonly used features of WebLOAD:

- It will provide a flexible test scenario creation.
- This tool detects the bottleneck automatically.
- Customer support can be easily approachable.
- It can evaluate the performance test results from any browser or mobile device.
- It will generate the load from the cloud.

5.11.1.5 LoadComplete

- This is another performance (load) testing tool. It is used to create and run automated tests for web services and web servers. It supports all types of browsers and web services. It will test the performance of our web server when it encounters a heavy load.
- With the help of this tool, we can observe multiple server metrics such as CPU usage, throughout the test runs.

1. Features of LoadComplete

- It will provide load modeling for performance testing, which means that it allows us to generate a massive load for stress testing.
- With the help of this, we can record and playback our actions in the web browser.
- It supports various platforms like Windows, UNIX.
- During the load testing, it will verify the server message body by taking the help of template-based rules, which make sure that the correct functioning of the server.
- It can test various types of applications like Flash, Flex, Silverlight, and Ajax.
- It will generate the load test reports, which include the customization of the user interface.

5.11.1.6 NeoLoad

- Neotys develops a testing tool called NeoLoad. Neoload is used to test performance test scenarios. With the help of NeoLoad, we can find bottleneck areas in web and mobile app development process.
- Neoload testing tool is faster than traditional tools. It will support a complete range of web, mobile and packaged applications such as SAP, Oracle and Salesforce covering all our testing needs. and share and manage test resources.

1. Features of NeoLoad

Following are some essential features of NeoLoad :

- It will support various frameworks and protocols such as HTTP/2, HTML5, API, AngularJS, Web Socket, SOAP, etc.
- It has a robust code-less design.
- It will change the functional test script into the performance test scripts.
- It will automatically update the test scripts.
- It will generate real-time test results.

5.11.1.7 LoadView

- It is powered by dotcom-monitor. With the help of this tool, we can display the actual performance of the application. It is used to perform load testing on real browsers that return correct data. It is a cloud-based tool that can be deployed in less time.

1. Features of LoadView

- It is used to find the bottlenecks and ensure the scalability of the applications.
- It will perform cloud-based load testing in real browsers.
- With the help of this tool, we can easily build our test scripts.
- It will support various Rich internet applications like Java, PHP, Ruby, HTML5, Flash, Silverlight, and so on.
- It includes global cloud-based testing, point and click scripting.
- It provides the dedicated Static IPs, which can be configured and allows us to execute our tests for the target resource behind a firewall.

❑❑❑

UNIT VI

6 Testing Framework

Syllabus

Testing Framework : *Software Quality, Software Quality Dilemma, Achieving Software Quality, Software Quality Assurance Elements of SQA, SQA Tasks, Goals and Metrics, Formal Approaches to SQA, Statistical Software Quality Assurance, Six Sigma for Software Engineering, ISO 9000 Quality Standards, SQA Plan, Total Quality Management, Product Quality Metrics, In process Quality Metrics, Software maintenance, Ishikawa's 7 basic tools, Flow Chart, Checklists, Pareto diagrams, Histogram, Run Charts, Scatter diagrams, Control chart, Cause Effect diagram. Defect Removal Effectiveness and Process.*

Contents

6.1 Testing Framework : Software Quality

- Software quality can be defined as : *An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.*
- An effective software process establishes the infrastructure that supports any effort at building a high quality software product.
- A useful product delivers the content, functions, and features that the end-user desires, but as important, it delivers these assets in a reliable, error free way.
- By adding value for both the producer and user of a software product, high quality software provides benefits for the software organization and the end-user community.
- An effective software process establishes the infrastructure that supports any effort at building a high quality software product.
- The management aspects of process create the checks and balances that help avoid project chaos, a key contributor to poor quality.
- Software engineering practices allow the developer to analyze the problem and design a solid solution, both critical to building high quality software.
- Finally, umbrella activities such as change management and technical reviews have as much to do with quality as any other part of software engineering practice.

Quality as a Pragmatic View

1. The transcendental view argues that quality is something that you immediately recognize, but cannot explicitly define.
2. The user view sees quality in terms of an end-user's specific goals. If a product meets those goals, it exhibits quality.
3. The manufacturer's view defines quality in terms of the original specification of the product. If the product conforms to the spec, it exhibits quality.
4. The product view suggests that quality can be tied to inherent characteristics (e.g., functions and features) of a product.
5. Finally, the value-based view measures quality based on how much a customer is willing to pay for a product. In reality, quality encompasses all of these views and more.

Quality Dimensions are as follows :

1. **Performance Quality :** Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end-user ?

2. **Feature quality** : Does the software provide features that surprise and delight first-time end-users ?
3. **Reliability** : Does the software deliver all features and capability without failure ? Is it available when it is needed ? Does it deliver functionality that is error free ?
4. **Conformance** : Does the software conform to local and external software standards that are relevant to the application ? Does it conform to de facto design and coding conventions ? For example, does the user interface conform to accepted design rules for menu selection or data input ?

6.1.1 Need of Software Quality

- Software is computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
- Software quality, by definition, is the degree to which software possesses a desired combination of attributes.
- Software quality is the degree to which a system, component, or process meets specified requirements.
- Software quality is the degree to which a system, component, or process meets customer or user needs or expectations. Quality means conformance to requirements
- Quality that is defined as a matter of products and services whose measurable characteristics satisfy a fixed specification - that is, conformance to an in beforehand defined specification.
- The nine causes of software errors are :
  1 Faulty requirements definition
  2 Client-developer communication failures
  3 Deliberate deviations from software requirements
  4 Logical design errors
  5 Coding errors
  6 Non-compliance with documentation and coding instructions
  7 Shortcomings of the testing process
  8 User interface and procedure errors
  9 Documentation errors.
- Meeting customer needs : Quality that is identified independent of any measurable characteristics. That is, quality is defined as the products or services capability to meet customer expectations explicit or not.

- The fundamental reason for measuring software and the software process is to obtain data that helps us to better control the schedule, cost, and quality of software products. It is important to be able to consistently count and measure basic entities that are directly measurable, such as size, defects, effort and time.
- Consistent measurements provide data for doing the following :
  1 Quantitatively expressing requirements, goals and acceptance criteria.
  2 Monitoring progress and anticipating problems.
  3 Quantifying tradeoffs used in allocating resources.
  4 Predicting the software attributes for schedule, cost and quality.
- With software or anything else, assessing quality means measuring value. Something of higher quality has more value than something that is of lower quality. Yet measuring value requires answering another question : value to whom ? In thinking about software quality, it's useful to focus on three groups of people who care about its value.

1 The software's users, who apply this software to some problem.
2 The development team that creates the software.
3 The sponsors of the project, who are the people paying for the software's creation.

6.1.2 Quality Challenges

- Statements like "a system will have high performance" or "a system will be user friendly" are acceptable in the really early stages of requirements elicitation process.
- As more information becomes available, the above statements become useless as they are meaningless for the purpose of actually designing a solution. In each of the two examples above an attempt is made to describe the behavior of a system. Both statements are useless as they provide no tangible way of measuring the behavior of the system.
- The quality attributes must be described in terms of scenarios, such as "when 100 users initiate 'complete payment' transition, the payment component, under normal circumstances, will process the requests with an average latency of three seconds." This statement or scenario, allows an architect to make quantifiable arguments about a system.
- **Limited warranty on software and media :** Company warrants the disks on which the software is distributed to be free from defects in materials and workmanship and that the software will perform substantially in accordance with the documentation for a period of 90 days from your receipt of the product.

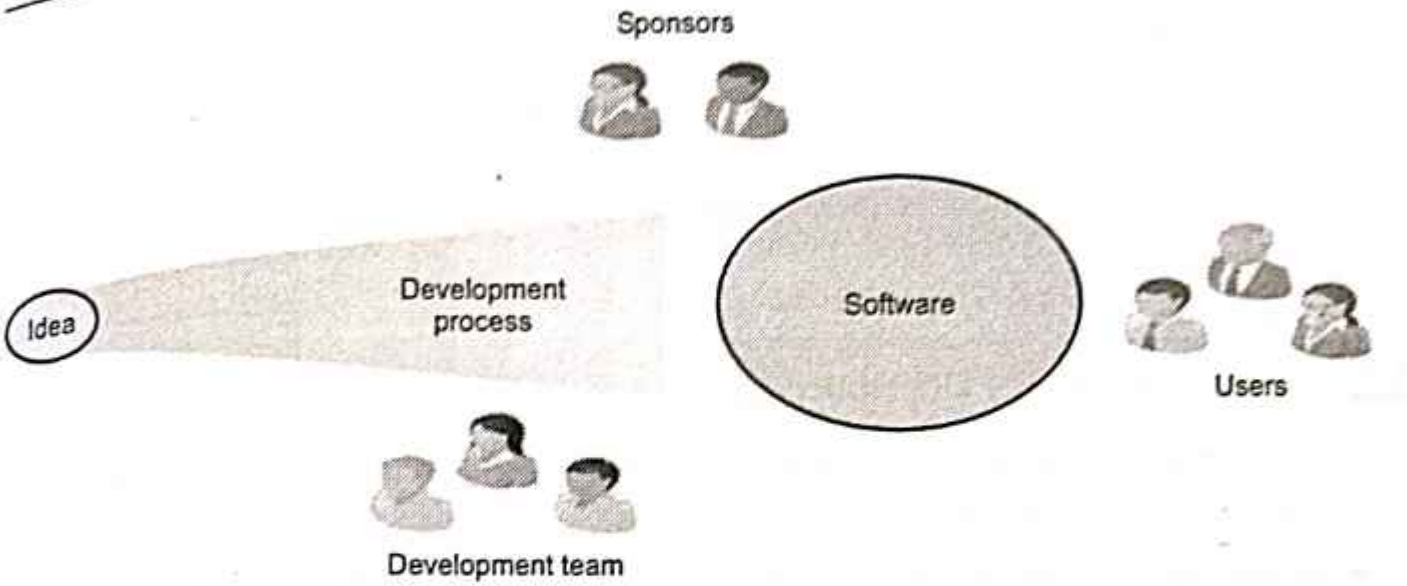

Fig. 6.1.1 Need of software quality

- If the product fails to comply with the warranty set forth above, company entire liability and your exclusive remedy will be replacement of the disk. Company reasonable effort to make the product meet the warranty set forth above.
- Software products differ from other industrial products with respect to the following characteristics :

1. **Product complexity :** Typical software product allows tens of thousands of operational options. Typical industrial products and even advanced industrial products do not reach this level of variety of options.
2. **Product visibility :** Since software products are invisible, defects in the software are not visible unless testing procedures are applied. But industrial products are visible and most defects are visible to the production team by changes in color or shape.
3. **Development and production process :** Defects in industrial products usually occur during each stage through which the product has to pass, namely development, product production planning and manufacturing. At each stage the product is examined and tested independently by a different team. In contrast, software products are examined and tested during the development stage only.

- Factors affecting defect detection in software products vs. other industrial products.

Parameters	Software products	Other industrial products
Complexity	Higher complexity It allows larger number of operational options	Lower complexity It allows thousands of operational options

Product visibility	Product is invisible Detection of defects is difficult by sight	Product is visible Detection of defects is easy by sight
Nature of development and production process	Defect detection is possible only in one phase	Defect detection is possible in all phase of development and production

6.1.3 Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive that you'll be out of business.
- Either you missed the market window, or you simply exhausted all your resources.
- So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete.
- Fig. 6.1.2 shows cost to find and repair an error.

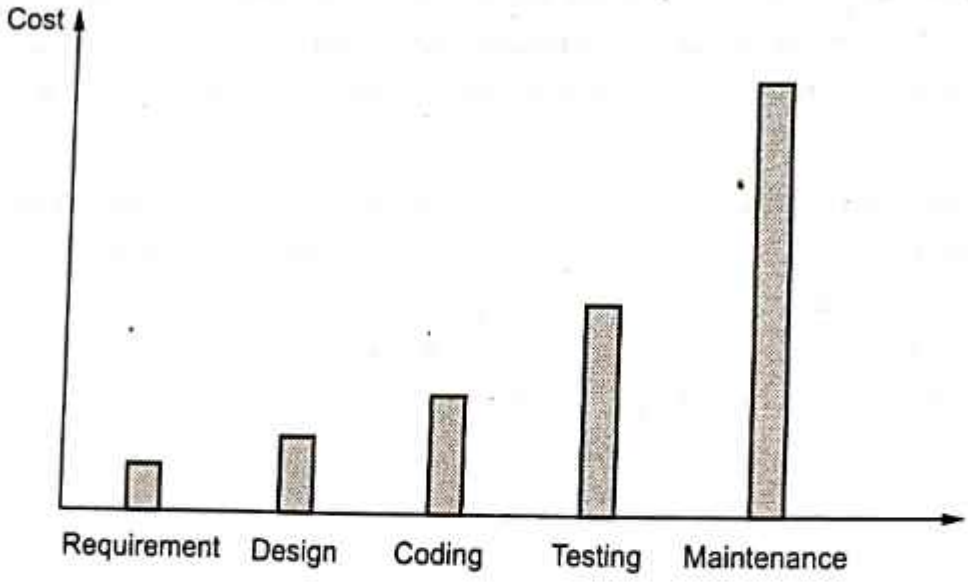

Fig. 6.1.2 Cost to find and repair an error

- The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.

Cost of Quality

Types of cost	Parameters
Prevention costs	• Quality planning • Formal technical reviews • Test equipment • Training
Appraisal costs	• Technique review • Data collection • Testing and debugging
Failure costs	• Internal failure cost (prior to shipment) • External failure cost (after shipment)

6.1.4 Achieving Software Quality

- Software quality is the result of good project management and solid engineering practice. To build high quality software you must understand the problem to be solved and be capable of creating a quality design the conforms to the problem requirements
- Eliminating architectural flaws during design can improve quality.
- Project management - Project plan includes explicit techniques for quality and change management
- Quality control - Series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications
- Quality assurance - Consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions.

Impact of Management Actions

1. Estimation decisions : Irrational delivery date estimates cause teams to take short-cuts that can lead to reduced product quality.
2. Scheduling decisions : Failing to pay attention to task dependencies when creating the project schedule may force the project team to test modules without their subcomponents and quality may suffer.
3. Risk-oriented decisions : Reacting to each crisis as it arises rather than building in mechanisms to monitor risks and having established contingency plans may result in products having reduced quality.

6.2 Software Quality Assurance

SPPU : Dec-18, 19, May-19

- Software Quality Assurance (SQA) is a process that ensures that developed software meets and complies with defined or standardized quality specifications. SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets desired quality measures.
- **IEEE definition of "Software Quality Assurance" :** A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. A set of activities designed to evaluate the process by which the products are developed or manufactured
- **The various objectives of SQA are as follows :**
  1. Quality management approach.
  2. Measurement and reporting mechanisms.
  3. Effective software-engineering technology.
  4. A procedure to assure compliance with software-development standards where applicable.
  5. A multi-testing strategy is drawn.
  6. Formal technical reviews that are applied throughout the software process.
- **The major goals of SQA are as follows :**
  1. SQA activities are planned.
  2. Non-compliance issues that cannot be resolved within the software project are addressed by senior management.
  3. Adherence of software products and activities to the applicable standards, procedures and requirements is verified objectively.
  4. Affected groups and individuals are informed of SQA activities and results.

6.2.1 Software Quality Assurance vs. Software Quality Control

Software quality assurance	Software quality control
SQA is pro-active means it identifies weaknesses in the processes.	SQC is reactive means it identifies the defects and also corrects the defects or bugs also.
A set of activities designed to evaluate the process by which the products are developed or manufactured.	Quality control is defined as "a set of activities designed to evaluate the quality of a developed or manufactured product".

Quality assurance is the process of managing for quality.	Quality control is used to verify the quality of the output.
It does not involve executing the program or code.	It always involves executing the program or code.
Verification is an example of quality assurance.	Validation/Software testing is an example of quality control.
Quality assurance is process oriented.	Quality control is product oriented.
In order to meet the customer requirements QA defines standards and methodologies.	QC confirms that the standards are followed while working on the product.
It requires involvement of the whole team.	It requires involvement of testing team.

6.2.2 Objectives of SQA Activities

- The objectives of SQA activities are functional, managerial and economic aspects of process and product oriented. SQA activities must also include scheduling and budgeting.
- **The objectives of SQA activities in software development (process-oriented) :**
  a. Assuring an acceptable level of confidence that the software will conform to functional technical requirements.
  b. Assuring an acceptable level of confidence that the software will conform to managerial scheduling and budgetary requirements.
  c. Initiation and management of activities for the improvement and greater efficiency of software development and SQA activities.
- **The objectives of SQA activities in software maintenance (product-oriented) :**
  a. Assuring an acceptable level of confidence that the software maintenance activities will conform to the functional technical requirements.
  b. Assuring an acceptable level of confidence that the software maintenance activities will conform to managerial scheduling and budgetary requirements.
  c. Initiate and manage activities to improve and increase the efficiency of software maintenance and SQA activities.

6.2.3 Elements of SQA

- Software quality assurance elements are as follows :
  a. Standards
  b. Reviews and Audits
  c. Testing
  d. Error/defect collection and analysis
  e. Change management
  f. Education

g. Vendor management h. Security management

i. Safety j. Risk management

a. Standards : It ensures that standards are adopted and followed.

b. Reviews and audits : Audits are reviews performed by SQA personnel to ensure that quality guidelines are followed for all software engineering work.

c. Testing ensure that testing id properly planned and conducted.

d. Error/defect collection and analysis: it collects and analyses error and defect data to better understand how errors are introduced and can be eliminated.

e. Changes management ensures that adequate change management practices have been instituted.

f. Education takes lead in software process improvement and educational program.

g. Vendor management suggests specific quality practices vendor should follow and incorporates quality mandates in vendor contracts

h. Security management ensures use of appropriate process and technology to achieve desired security level.

i. Safety is responsible for assessing impact of software failure and initiating steps to reduce risk.

j. Risk management ensures risk management activities are properly conducted and that contingency plans have been established.

6.2.4 SQA Tasks

- SQA Tasks are as follows :
  1. Prepare SQA plan for the project. The plan identifies an evaluations to be performed, audits and reviews to be performed, standards that are applicable to the project, procedures for error reporting and tracking, documents to be produced by the SQA group and amount of feedback provided to the software project team
  2. Participate in the development of the project's software process description. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards and other parts of the software project plan.
  3. Review software engineering activities to verify compliance with the defined software process.
  4. Audit designated software work products to verify compliance with those defined as part of the software process.

5. Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
6. Record any evidence of noncompliance and reports them to management.

6.2.5 SQA Goals

- Requirements quality : The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow.
- Design quality : Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements.
- Code quality : Source code and related work products must conform to local coding standards and exhibit characteristics that will facilitate maintainability.
- Quality control effectiveness : A software team should apply limited resources in a way that has the highest likelihood of achieving a high quality result.

6.2.6 Formal Approaches to SQA

- SQA is defined as a planned and systematic approach to the evaluate of the quality and act to software product standard, software process, and software procedure. It is a systematic approach is used to enhance the quality of software
- Assumes that a rigorous syntax and semantics can be defined for every programming language. It allows the use of a rigorous approach to the specification of software requirements
- Applies mathematical proof of correctness techniques to demonstrate that a program conforms to its specification.
- How Formal approaches to SQA works ? :
  1. Involve 3 to 5 people for reviewers.
  2. Preparation required more than 2 hours.
  3. Meeting review should be less than 2 hours.
  4. Focus on discrete work.
  5. Review organize way.
  6. Questioning enable the producer ti to discover the error.
  7. Record and write down the issues.
  8. Review decided to accept or reject the work on the review of product or not.

Formal Technical Reviews - 2

1. Reviewers ask questions that enable the producer to discover his or her own error (the product is under review not the producer)
2. Producer of the work product walks the reviewers through the product
3. Recorder writes down any significant issues raised during the review
4. Reviewers decide to accept or reject the work product and whether to require additional reviews of product or not.

Review Questions

1. *What does SQA ensure ? What are the goals of SQA ?*
   SPPU : Dec.-18,19, End Sem, Marks 6
2. *Define software quality and software quality assurance. List various objective of SQA.*
   SPPU : May-19, Dec-18, 19, End Sem, Marks 5
3. *Explain software quality assurance and elements of SQA.*
   SPPU : May-19, End Sem, Marks 6

6.3 Statistical Software Quality Assurance

- Information about software errors and defects is collected and categorized. An attempt is made to trace each error and defect to its underlying cause, for example : Non-conformance to specifications, design error, violation of standards, poor communication with the customer.
- Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the vital few).
- Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects.

Software Reliability

- It is defined as the probability of failure free operation of a computer program in a specified environment for a specified time period
- It can be measured directly and estimated using historical and developmental data. Software reliability problems can usually be traced back to errors in design or implementation.
- Reliability metrics are units of measure for system reliability.
- System reliability is measured by counting the number of operational failures and relating these to demands made on the system at the time of failure.

- A long-term measurement program is required to assess the reliability of critical systems.

6.4 Six Sigma for Software Engineering

SPPU : Dec-18, 19, May-19

- According to David Card, Six sigma is "A generic quantitative approach to improvement that applies to any process."
- Six Sigma is a disciplined, data-driven approach and methodology for eliminating defects in any process from manufacturing to transactional and from product to service.
- In essence, six sigma is an approach to finding the cause of business problems and solving them making an impact on the bottom line of a corporation and decreasing variation for products.
- Since the early 1990, Six Sigma has been extensively used for achieving total customer satisfaction with innovative products at competitive price. Its objectives are to deliver products when promised, without delivered defects, early life failures, or failures during use.
- Key Definitions :
  1. **Critical to quality :** Internal critical quality parameters that relate to the wants and needs of the customer.
  2. **Critical to customer :** The input to the Quality Function Deployment activity.
  3. **Defect :** Any type of undesired result.
  4. **Opportunity :** area within a product, process, service, or other system where a defect could be produced or where you fail to achieve the ideal product in the eyes of the customer

Features of Six Sigma :

1. Six Sigma's aim is to eliminate waste and inefficiency, thereby increasing customer satisfaction by delivering what the customer is expecting.
2. Six Sigma follows a structured methodology, and has defined roles for the participants.
3. Six Sigma is a data driven methodology, and requires accurate data collection for the processes being analyzed.
4. Six Sigma is about putting results on Financial Statements.
5. Six Sigma is a business-driven, multi-dimensional structured approach

- Six Sigma is not team driven like TQM. It is a management approach. Defect metrics are the main tool for leaders to impact cost and margin.

- Applying Six Sigma to software development makes product development and other projects transparent to both management and customers. However, transparency requires an important cultural change.
- We know from experience that bad communication is a major reason why projects fail, and software projects in particular. We expect from better transparency that meeting both deadlines and customer requirements becomes easier.
- The major problem with early Six Sigma attempts was that there was no connection of software metrics to economic success.
- Counting mistakes and defects is not a clear indication if the software project is going to be successful. Other metrics like time-to-market and user friendliness are much more important in many application areas. Sometimes, reliability is of essence, but not always

Why do we need Six Sigma for Software ?

- Software today is responsible for most of the added value in products, and must be blamed for many of its failures. Even if the iron hook breaks, it may be the software embedded in the measurement instrument to blame for not having detected it in time.
- When in Germany the high speed intercity express train crashed into an overpass, it was software that didn't detect the broken wheel ring well before the accident.
- Mobile networks are suffering from not being able to provide interconnection to the Internet and interoperability between their own services. It is the software that fails.
- In e-Commerce and for making Web Services to work, security, reliability and fault tolerance are of essence. Software and business processes are not cooperating, as they should to make it profitable.
- Software is so ubiquitous that we must solve the software development problem to address a lot of other problems the society has.
- The Six Sigma approach is :
  1. Set the goal - Define
  2. Define the metrics - Measure
  3. Measure where you go - Analyse
  4. Improve your processes while you go - Improve
  5. Act immediately if going the wrong path - Control.
- The Six Sigma methodology defines five core steps :
  1. Define customer requirements and deliverables and project goals via well-defined methods of customer communication

2. Measure the existing process and its output to determine current quality performance (collect defect metrics)
3. Analyze defect metrics and determine the vital few causes.
4. Improve the process by eliminating the root causes of defects.
5. Control the process to ensure that future work does not reintroduce the causes of defects.

- Benefits of Six Sigma

Six Sigma offers six major benefits that attract companies -

1. Generates sustained success
2. Sets a performance goal for everyone
3. Enhances value to customers
4. Accelerates the rate of improvement
5. Promotes learning and cross-pollination
6. Executes strategic change.

Review Question

1. *What is six sigma ? Explain the terms DMAIC and DMADV.*

SPPU : May-19, Dec-18, 19, End Sem, Marks 6

6.5 ISO 9000 Quality Standards

SPPU : Dec-18, 19

- A quality management system defines and establishes an organization's quality policy, objectives and procedures. Quality management system standards do not provide specific quality indicators or describe ways of achieving them, since these will be different in every situation. Instead, they provide generic frameworks and general principles that can be applied to any organization, of any size, in any industry.
- Quality management principles are a set of fundamental beliefs, norms, rules and values that are accepted as true and can be used as a basis for quality management.
- High levels of quality are essential to achieve company business objectives. Quality does not only relate solely to the end products and services a company provides but also relates to the way the company employees do their job and the work processes they follow to produce products or services.
- The work processes should be as efficient as possible and continually improving. Company employees constitute the most important resource for improving quality.

Each employee in all organizational units is responsible for ensuring that their work processes are efficient and continually improving.

- Certification standards and assessment standards both are different in content. Quality management should be separate from project management to ensure independence.
- The benefits of using standards :
  1. The ability to apply methodologies and procedures of the highest professional level.
  2. Better mutual understanding and coordination among development teams but especially between development and maintenance teams.
  3. Greater cooperation between the software developer and external participants in the project.
  4. Better understanding and cooperation between suppliers and customers, based on the adoption of standards as part of the contract.

Scope of quality management standards - certification standards

1. Enable a software development organization to demonstrate consistent ability to assure acceptable quality of its software products or maintenance services. Certification is granted by an external body.
2. Serve as an agreed-upon basis for customer and supplier evaluation of the supplier's quality management system. It accomplished by performance of a quality audit by the customer.
3. Support the organization's efforts to improve its quality management system through compliance with the standard's requirements.

6.5.1 ISO 9001 and ISO 9000-3

ISO 9000

- ISO 9000 is an international set of standards for quality management. It is applicable to a range of organizations from manufacturing to service industries.
- ISO 9000 has a broad scope : Hardware, software, processed materials and services. The standard outlines the basic elements of a good quality management system. These elements are good business practice.
- ISO 9001 is applicable to organizations which design, develop and maintain products. ISO 9001 is a generic model of the quality process must be instantiated for each organization.
- ISO 9000-1 provides guidelines for selection and use of the ISO 9000 standards.

- The ISO 9004 series provide guidance for quality management, i.e. for the design, implementation and improvement of a quality system. A quality system is a set of 'organizational structures, procedures, processes and resources needed to implement quality management.
- ISO 9001, 9002 and 9003 are models for external quality assurance. They specify a set of requirements.
- ISO 9000 and ISO 9004 are guidelines for quality management and are not Mandatory for certification.
- ISO 9001, ISO 9002 and ISO 9003 are quality system standards.
- ISO 9001 is the broadest standard and provides a model for design, development, production, installation and servicing.
- ISO 9002 is limited to production, installation and servicing.
- ISO 9003 is further limited to inspection and testing.
- A company should first use ISO 9000 to design and to implement a quality system. Once the quality has been installed, the company may use the quality assurance models of ISO 9001, ISO 9002 or ISO 9003 to demonstrate the adequacy of the quality system.
- ISO 9002 is used for production and installation. It is the standard that governs the manufacture of a product. It is designed to ensure conformance to production and installation methods.
- ISO 9003 is the standard directed at the final test and inspection of products. The standard pre-assumes an extensive quality control function and specifies what is needed for conformance to requirements.
- ISO 9004 is for internal use only and lists the components that compose quality systems.

Why ISO 9000 certification ?

1. Better organizational definition
2. Greater quality awareness
3. Better documentation of processes
4. Increased control of operations
5. Ongoing analysis of and solution to problems
6. Positive cultural change
7. Improved customer satisfaction and increased market opportunities.

ISO 9000 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- External body may certify that an organisation's quality manual conforms to (ISO 9000) standards.
- Customers are, increasingly, demanding that suppliers are ISO 9000 certified.

6.5.2 ISO 9000-3 Quality Management System Principles

1. Customer focus
2. Leadership
3. Involvement of people
4. Process approach
5. System approach to management
6. Continual improvement
7. Factual approach to decision making
8. Mutually supportive supplier relationships.

Fig. 6.5.1 The eight quality management principles

- The eight principles of quality management can be used by senior management as a framework to guide their organizations towards improved performance.

Principle	Functions	Benefit
Customer focus	• Organizations depend on their customers and therefore should understand current and future customer needs. • Meet customer requirements and strive to exceed customer expectations.	1. Increased revenue and market share. 2. Increased effectiveness in the use of the organization's resources. 3. Improved customer loyalty leading to repeat business.

Leadership	• Leaders establish unity of purpose and direction of the organization. • They should create and maintain the internal environment in which people can become fully involved in achieving the organization's objectives.	1. People will understand and be motivated towards the organization's goals and objectives. 2. Mis-communication will be minimized.
Involvement of people	• People at all levels are the essence of an organization and their full involvement enables their abilities to be used for the organization's benefit.	1. Motivated, committed and involved people within the organization.
Process approach	• A desired result is achieved more efficiently when activities and related resources are managed as a process.	1. Lower costs and shorter cycle times through effective use of resources. 2. Improved, consistent and predictable results
System approach to management	• Identifying, understanding and managing interrelated processes as a system contributes to the organization's effectiveness and efficiency in achieving its objectives.	1. Ability to focus effort on the key processes. 2. Providing confidence to interested parties as to the consistency, effectiveness and efficiency of the organization.
Continual improvement	• Continual improvement of the organization's overall performance should be a permanent objective of the organization.	1. Flexibility to react quickly to opportunities. 2. Performance advantage through improved organizational capabilities.
Factual approach to decision making	• Effective decisions are based on the analysis of data and information.	Increased ability to review, challenge and change opinions and decisions.
Mutually supportive supplier relationships	• An organization and its suppliers are interdependent and a mutually beneficial relationship enhances the ability of both to create value.	1. Increased ability to create value for both parties. 2. Optimization of costs and resources.

6.5.3 ISO 9000-3 Requirements

- The ISO 9000-3 includes about 20 requirements that relate to various aspects of software quality management classified into the following five groups :
  1. Quality management system
  2. Management responsibilities
  3. Resource management
  4. Product realization
  5. Measurement, analysis and improvement.

Requirements	Functions
Quality management system requirements	1. Establish a quality management system 2. Document the quality management system
Management responsibilities	1. Management commitment 2. Customer focus 3. Quality policy 4. Planning 5. Administration 6. Management review
Resource management	1. Provision of resources 2. Human resources 3. Facilities 4. Work environment
Product realization	1. Planning of realization processes 2. Customer-related processes 3. Design and/or development 4. Purchasing 5. Production and service operations 6. Control of measuring and monitoring devices
Measurement, analysis and improvement	1. Planning 2. Measurement and monitoring 3. Control of nonconformity 4. Analysis of data 5. Improvement

6.5.4 Certification According to ISO 9000-3

- The ISO 9000-3 certification process verifies that an organization's software development and maintenance processes fully comply with the standard's requirements.
- The objective of the International Standards Organization 9000 series of standards is to certify that an organization has quality manufacturing processes. Thus, if a supplier wanted to demonstrate competence of products, it would make application to demonstrate that it meets the ISO 9000 standards.
- **Documentation requirements :** The ISO 9000 audit is heavily focused on evaluation of documentation. Four tiers of quality system documentation are required :

1. First tier-the quality manual.

2. Second tier-quality management procedures (i.e., core procedures).
3. Third tier-area work instructions (i.e., standard operating procedures, test methods, calibration methods).
4. Fourth tier-forms, records, books and files.

Certification process :

1. The company first implements the control and documentation procedures outlined in the series.
2. It then involves a thorough audit by an independent certification organization that is licensed to register quality systems by an accreditation body.
3. Upon compliance, it receives a registration certificate and its name is included in a published directory of registered suppliers.
4. The systems will be continually verified by the registrar in periodic surveillance and full audits are conducted every few years.

Development of the organization's SQA system :

- Organization first development of a quality model and SQA procedures. Second steps is to development of other SQA infrastructure like staff training, preventive and corrective actions procedures, configuration management services and documentation and quality record control.
- Lastly the development of a project progress control system.

Implementation of the organization's SQA system :

- Setting up a staff instruction program
- Leaders and managers are expected to follow up and support the implementation efforts made by their units.
- Internal quality audits are carried out to verify the success in implementation.
- The findings will determine of whether the organization has reached a satisfactory level of implementation.

Problems with certification

1. Costs - application and maintenance
2. Time - application and maintenance
3. Level of internal expertise
4. Executive commitment
5. Selection of registration.

Review Question

1. *What is ISO standard ? What are its advantages ?*

SPPU : Dec-18, 19, End Sem, Marks 5

6.6 SQA Plan

- Management section - describes the place of SQA in the structure of the organization.
- Documentation section - describes each work product produced as part of the software process.
- Standards, practices, and conventions section - lists all applicable standards/practices applied during the software process and any metrics to be collected as part of the software engineering work.
- Reviews and audits section - provides an overview of the approach used in the reviews and audits to be conducted during the project.
- Test section - references the test plan and procedure document and defines test record keeping requirements.
- Problem reporting and corrective action section - defines procedures for reporting, tracking and resolving errors or defects, identifies organizational responsibilities for these activities.
- Other - tools, SQA methods, change control, record keeping, training, and risk management.

6.7 Total Quality Management

SPPU : Dec-18, 19, May-19

- Quality is the fullfillment of individual needs of the stakeholders, users and customers of a product, service or result of a project.
- So testing does not seem enough to get a high quality product. In particular, pure curative action like testing is not sufficient to get a product of high quality. To aim for higher targets, the concept of Total Quality Management (TQM) has been introduced to the manufacturing industry and is widely accepted nowadays.
- TQM describes that a good quality of product is determined by a good quality of the processes which lead to the product. This is related to all processes which are conducted by an organization to deliver a product.
- It is not only about the developing team or department but also about the procurement, controlling, portfolio management departments. In general, the overall organizational environment determines the process of task fullfilling.

- A management philosophy embracing all activities through which the needs and expectations of the CUSTOMER and COMMUNITY and the objectives of the organization are satisfied in the most efficient and cost effective manner by maximising the potential of ALL employees in a continuing drive for improvement."
- TQM involves following basic concept :

1. Leadership
2. Customer satisfaction
3. Employee involvement
4. Continuous process improvement
5. Supplier partnership
6. Performance measures.

- **Leadership** : Top management must realize importance of quality. Quality is responsibility of everybody, but ultimate responsibility is CEO. Quality excellence must become part of business strategy. Lead in the implementation process.
- **Customer satisfaction** : Customer is always right, in Japan customer is "King". Customer expectations constantly changing. Satisfaction is a function of total experience with organization. It must give customers a quality product or service, reasonable price, on-time delivery and outstanding service. It need to continually examine the quality systems and practices to be responsive to ever changing needs, requirements and expectations.
- **Employee involvement** : People are the most important resource/asset. Quality comes from people.
- **Continuous process improvement** : View all work as process, production and business. Process means purchasing, design, invoicing.
- **Supplier partnership** : The 40 % product cost comes from purchased materials; therefore supplier quality management is important. The substantial portion of quality problems comes from suppliers and need partnership to achieve quality improvement, i.e. long-term purchase contract. It is supplier management activities.
- **Performance measures** : It is managing by fact rather than gut feelings. The effective management requires measuring performance. It uses a baseline, to identify potential projects, to asses results from improvement

Review Questions

1. *Write short note on total quality management.*

SPPU : May-19, Dec-18, 19, End Sem, Marks 6

2. *Describe key elements of total quality management.*

SPPU : May-19, Dec-18, 19, End Sem, Marks 6

6.8 Product Quality Metrics

SPPU : Dec.-18, 19, May-19

- Product metrics refer to the software system's operational phase. In most of the project, software developer is required to provide customer service during the software's operational phase.
- Customer services are divided into two category :
  1. Help Desk services (HD)
  2. Corrective maintenance services.
- **HD metrics** are based on all customer calls while **corrective maintenance metrics** are based on failure reports.
- **HD quality metrics** are as follows :
  1. HD calls density metrics - Measured by the number of calls.
  2. HD calls severity metrics - The severity of the HD issues raised.
  3. HD success metrics - The level of success in responding to HD calls

HD calls density metrics :

$$\text{HD calls Density (HDD)} = \frac{\text{NHYC}}{\text{KLMC}}$$

$$\text{WHDD Weighted HD calls Density WHDD} = \frac{\text{WHYC}}{\text{KLMC}}$$

$$\text{WHDF Weighted HD calls per Function point WHDF} = \frac{\text{WHYC}}{\text{NMFP}}$$

- NHYC = Number of HD calls during a year of service.
- KLMC = Thousands of lines of maintained software code.
- WHYC = Weighted HD calls received during one year of service.
- NMFP = Number of function points to be maintained.

Severity of HD Calls Metrics and HD Success Metrics

Average severity of HD calls (ASHC)

$$= \frac{\text{Weighted HD calls received during one year of service (WHYC)}}{\text{Number of HD calls during a year of service (NHYC)}}$$

HD service success (HDS)

$$= \frac{\text{Number of yearly HD calls completed on time during one year of service (NHYNOT)}}{\text{Number of HD calls during a year of service (NHYC)}}$$

HD productivity and effectiveness metrics

- Productivity metrics relate to the total of resources invested during a specified period. It uses the easy-to-apply KLMC measure of maintained software system's size.
- Effectiveness metrics relate to the resources invested in responding to a HD customer call.

$$\text{HD Productivity (HDP)} = \frac{\text{Total yearly working hours investedin HD servicing of the software system (HDYH)}}{\text{Thousands of lines of maintained software code(KLMC)}}$$

Failures of maintenance services metrics

- A customer call related to a software failure problem that was supposed to be solved after a previous call is commonly treated as a *maintenance service failure*.

Maintenance repeated repair failure (MRepF)

$$= \frac{\text{Number of repeated software failure calls (RepYF)}}{\text{Number of software failures detected during a year of maintance service(NYF)}}$$

Benefits of product metrics

1. Assist in the evaluation of the analysis and evaluation model
2. Provide indication of procedural design complexity and source code complexity
3. Facilitate design of more effective testing.

Review Question

1. *Explain with example product quality metric.*

SPPU : May-19, Dec.-18, 19, End Sem, Marks 5

6.9 Inprocess Quality Metrics

- Software development process metrics can fall into one of the following classes :
  1. Software process quality metrics
  2. Software process timetable metrics
  3. Error removal effectiveness metrics
  4. Software process productivity metrics.

What are Metrics?

- Software process and project metrics are quantitative measures. They are a management tool.

- They offer insight into the effectiveness of the software process and the projects that are conducted using the process as a framework
- Basic quality and productivity data are collected. These data are analyzed, compared against past averages, and assessed.

6.9.1 Software Process Quality Metrics

- Process metrics are collected across all projects and over long periods of time. They are used for making strategic decisions.
- The intent is to provide a set of process indicators that lead to long-term software process improvement. The only way to know how/where to improve any process is to measure specific attributes of the process or develop a set of meaningful metrics based on these attributes.
- Use the metrics to provide indicators that will lead to a strategy for improvement.
- We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as,
  1. Errors uncovered before release of the software
  2. Defects delivered to and reported by the end users
  3. Work products delivered
  4. Human effort expended
  5. Calendar time expended
  6. Conformance to the schedule
  7. Time and effort to complete each generic activity.
- Software process quality metrics is divided into two types :
  a. Error density metrics b. Error severity metrics.
- Weighted measures that determine the severity of the errors are considered to provide more accurate evaluation of the error situation.
- This measure is classification of the detected errors into severity classes, followed by weighting each class.
- The weighted error measure is computed by summing up multiples of the number of errors found in each severity class by the adequate relative severity weight.

Error density metrics

- Calculation of error density metrics involves two measures : **Software volume and errors counted.**
- Software volume measures. Some density metrics use the number of lines of code while others apply function points.

- Errors counted measures. Some relate to the number of errors and others to the weighted number of errors.

Name	Formula
Code Error Density	CED = NCE / KLOC
Development Error Density	DED = NDE /KLOC
Weighted Code Error Density	WCDE = WCE / KLOC
Weighted Development Error Density	WDED = WDE / KLOC
Weighted Code Errors per Function Point	WCEF = WCE / NFP
Weighted Development Errors per Function Point	WDEF = WDE / NFP
NCE = The number of code errors detected by code inspections and testing.	
NCE = The number of code errors detected by code inspections and testing. NDE = Total number of development (design and code) errors) detected in the development process. WCE = Weighted total code errors detected by code inspections and testing. WDE = Total weighted development(design and code) errors detected in development process.	

- A software development department may apply two alternative metrics for calculation of code error density : CED and WCED.
- While the CED does not indicate quality below the acceptable level, the WCED metric does indicate quality below the acceptable level.

Error severity metrics

- The metrics belonging to this group are used to detect adverse situations of increasing numbers of severe errors in situations where errors and weighted errors, as measured by error density metrics, are generally decreasing

Code	Formula
Average Severity of Code Errors	ASCE = WCE / NCE
Average Severity of Development Errors	ASDE = WDE / NDE

6.9.2 Software Process Timetable Metrics

- Software process timetable metrics may be based on accounts of success (completion of milestones per schedule) in addition to failure events (non-completion per schedule).

- It is an alternative approach calculates the average delay in completion of milestones.

Code	Formula
Time Table Observance	TTO = MSOT / MS
Average Delay of Milestone Completion	ADMC = TCDAM / MS
MSOT = Milestones completed on time. MS = Total number of milestones. TCDAM = Total Completion Delays (days, weeks, etc.) for all milestones.	

- The TTO and ADMC metrics are based on data for all relevant mile-stones scheduled in the project plan.

6.9.3 Error Removal Effectiveness Metrics

- Software developers can measure the effectiveness of error removal by the software quality assurance system after a period of regular operation of the system.

  Development Errors Removal Effectiveness (DERE) = $\frac{NDE}{NDE+NYF}$

  Development Weighted Errors Removal Effectiveness (DWERE) = $\frac{WDE}{WDE+WYF}$

- NDE = Total number of development (design and code) errors detected in the development process.
- WCE = Weighted total code errors detected by code inspections and testing.
- WDE = Total weighted development (design and code) errors detected in development process.
- NYF = Number software failures detected during a year of maintenance service.
- WYF = Weighted number of software failures detected during a year of maintenance service.

6.10 Software Maintenance

- **Definition of maintenance :** It is the set of activities, both technical and managerial, that ensures that software continues to meet organizational and business objectives in a cost effective way.

Types of maintenance

1. **Corrective :** Taking existing code and correcting a fault that causes the code to behave in some way that deviates from its documented requirements.

2. **Adaptive :** Taking existing code and adapting it to provide new features and functionality. These are typically part of a new release of the code and part of a larger development effort.
3. **Perfective :** These are typically made to improve the maintainability of the code such as restructuring it to make it more easily understood or to remove ambiguities.
4. **Inspection :** These are usually made as a result of code inspections and focus more of adhering to coding standards or to reduce the likelihood of a failure.

6.10.1 Pre-maintenance Software Quality Components

- These software quality components are important activities to be carried out prior to initiating the required maintenance services. These entail the maintenance contract review and the maintenance plan construction.

Maintenance contract review :

- When considering the maintenance contract, many issues need to be addressed. These issues and decisions can vary from the type of customer or software the contract review is for.
- These are just two reasons that an adequate maintenance contract needs to be drawn up before agreeing to supply software maintenance services to any type of customer. The main objectives of software maintenance contract reviews are :
  1. Customer requirements classification : Issues such as; hours of service, size of user population, location of users and the types of applications to be used, all need to be considered.
  2. Review of alternative approaches to maintenance provision : such as subcontracting types of services.
  3. Review of estimates of required maintenance resources : Firstly these estimates should be examined with regards to advice from the proposal team about the required maintenance services. After this, the estimates should be reviewed with consideration to the company's ability to meet commitments, with respect to the professional competencies as well as the maintenance teams' abilities.
  4. Review of maintenance services to be provided by subcontractors and/or the customer : This review allows each party involved, to assess the services provided by one-another. Issues such as; quality assurance, payment to subcontractors and follow-up procedures are reviewed.
  5. Review of maintenance cost estimates : "These estimates should be reviewed on the basis of required resources."

Maintenance plan

- Maintenance plan is prepared for all projects and customers. The plan provide the framework within which maintenance provision is organized. The plan contains the following :
  1. A list of the contracted maintenance services : It includes name of the internal and external customers, the number of users, characteristics of corrective maintenance services (remote and on site) etc.
  2. A description of the maintenance team's organization.
  3. A list of maintenance facilities : This is the infrastructure, which includes the maintenance support centre and a documentation centre that makes it possible to provide a good service.
  4. A list of identified maintenance service risks : The list of identified maintenance service risks refers to situations whereby the inability to provide sufficient maintenance is anticipated.
  5. A list of required software maintenance procedures and controls.
  6. The software maintenance budget.

Quality metrics for software maintenance

- Software maintenance quality metrics are used mainly to identify trends in maintenance efficiency, effectiveness and customer satisfaction.

Costs of software maintenance quality

- The quality costs of corrective maintenance are divided into six types.
  1. Costs of prevention : It is costs of instruction and training of maintenance team, costs of preventative and corrective actions.
  2. Costs of appraisal : It is costs of error detection.
  3. Costs of managerial preparation and control : Costs of managerial activities carried out to prevent errors.
  4. Costs of internal failure : Costs of software failure corrections initiated by the maintenance team.
  5. Costs of external failure : Costs of software failure corrections initiated by customer complaints.
  6. Costs of managerial failure : Costs of software failures caused by managerial actions or inaction, i.e. costs of damages resulting from shortage of maintenance staff and/or inadequate maintenance task organization.

6.11 Ishikawa's 7 Basic Tools

SPPU : Dec-18, 19, May-19

- Most of QA knows about the Ishikawa's Seven Basic Tools of quality. These tools are basically used to analyze the reports and finding the root cause of the issues in the system. These seven tools are different ways to analyse the system.
- Ishikawa's Seven Basic Tools of quality are as follows :

1. Check Sheet (Checklist)
2. Pareto Diagram
3. Histogram
4. Scatter Diagram
5. Run Chart
6. Cause Effect Diagram
7. Control Chart.

6.11.1 Checklist

- A check sheet can be introduced as the most basic tool for quality. It is basically used for gathering and organizing data. Fig. 6.11.1 shows check sheet.

Telephone interruptions						
Reason	Day					
	Mon	Tues	Wed	Thus	Fri	Total
Wrong number	𝍸	||	|	𝍸	𝍸||	20
Info request	||	||	||	||	||	10
Boss	𝍸	||	𝍸||	|	||||	19
Total	12	6	10	8	13	49

Fig. 6.11.1 Checklist/checksheet

- Data are "collected and tabulated" on the check sheet to record the frequency of specific events during a data collection period. They prepare a "consistent, effective and economical approach" that can be applied in the auditing of quality assurance for re-viewing and to follow the steps in a particular process
- When this is done with the help of software packages such as Microsoft Excel, you can derive further analysis graphs and automate through macros available.
- The check sheets are in several, three major types are such as Defect-location check sheets; tally check sheets, and; defect-cause check sheets.

Advantages

- The main advantages of check sheets are to be very easily to apply and understand.

- It can make a clear picture of the situation and condition of the organization.
- They are efficient and powerful tools to identify frequently problems.

Disadvantages

- They do not have effective ability to analyze the quality problem into the workplace.

6.11.2 Pareto Diagram

- Pareto diagram is a frequency chart of bars in descending order. This means the categories represented by the tall bars on the left are relatively more significant then those on the right. This bar chart is used to separate the "vital few" from the "trivial many".
- Fig. 6.11.2 shows pareto diagram.

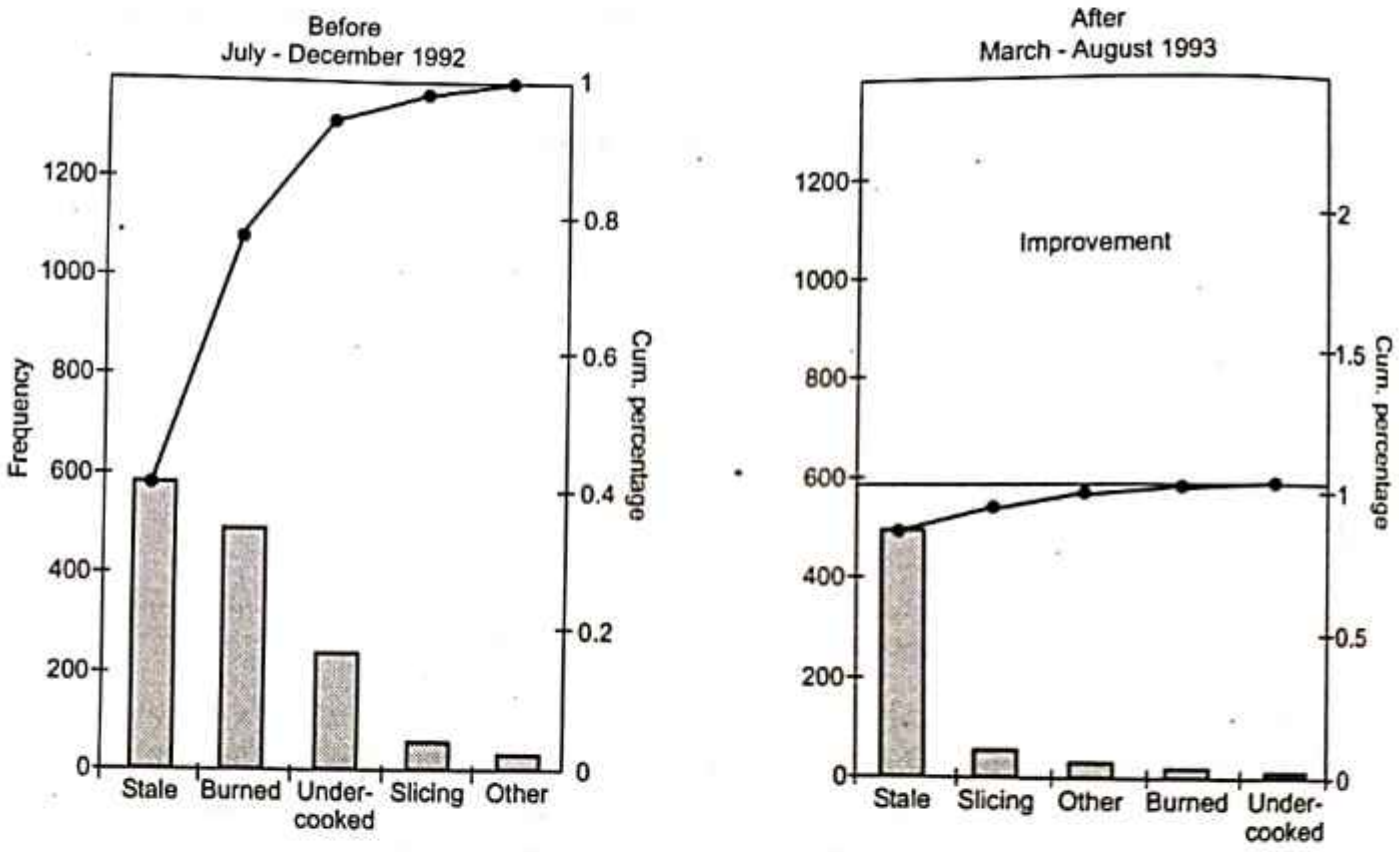

Fig. 6.11.2 Pareto diagram

- The aim of Pareto chart is to figure out the different kind of "nonconformity" from data figures, maintenance data, repair data, parts scrap rates, or other sources.
- Pareto chart can generate a mean for investigating concerning quality improvement and improving efficiency, "material waste, energy conservation, safety issues, cost reductions", etc

- In software development, the X -axis for a Pareto diagram is usually the defect cause and the Y -axis the defect count. By arranging the causes based on defect frequency, a Pareto diagram can identify the few causes that account for the majority of defects. It indicates which problems should be solved first in eliminating defects and improving the operation.
- Pareto analysis is commonly referred to as the 80 "20 principle (20 % of the causes account for 80 % of the defects), although the cause-defect relationship is not always in an 80 "20 distribution.

6.11.3 Histogram

- The histogram is a graphic representation of frequency counts of a sample or a population. The X -axis lists the unit intervals of a parameter ranked in ascending order from left to right and the Y -axis contains the frequency counts.
- It is a type of bar chart that visualizes both attribute and variable data of a product or process, also assists users to show the distribution of data and the amount of variation within a process.
- This grouping allows you see how frequently data in each class occur in the data set. Higher bars represent more data values in a class. Lower bars represent fewer data values in a class.
- Fig. 6.11.3 shows histogram.

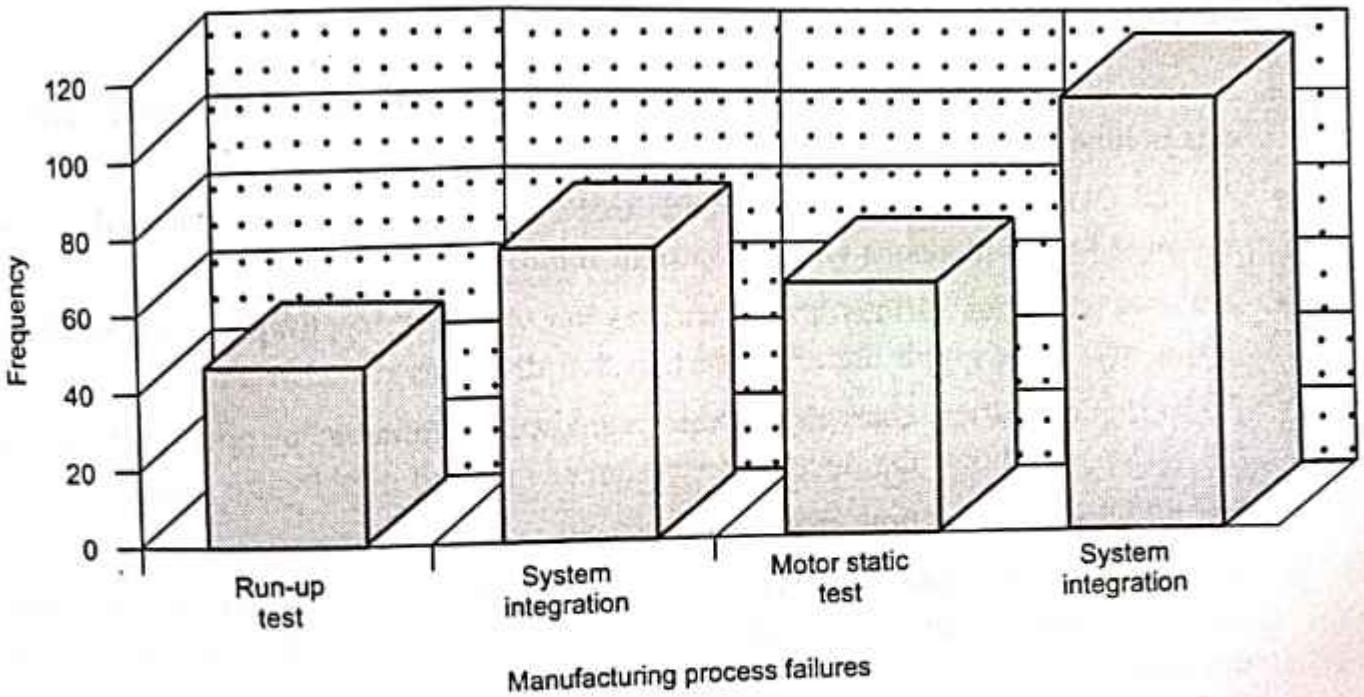

Fig. 6.11.3 Histogram

6.11.4 Scatter Diagram

- Scatter diagram is a powerful tool to draw the distribution of information in two dimensions. It helps to detect and analyze pattern relationships between two quality and compliance variables. It understanding if there is a relationship between them, so what kind of the relationship is (Weak or strong and positive or negative).
- Fig. 6.11.4 shows scatter diagram.
- The shape of the scatter diagram often shows the degree and direction of relationship between two variables, and the correlation may reveale the causes of a problem.
- In these diagrams, one variable denotes one axis and another variable denotes the other axis.

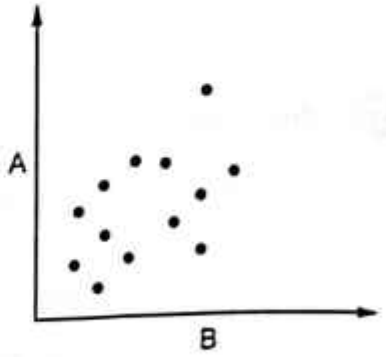

Fig. 6.11.4 Scatter diagram

- Correlation refers to the measure of the relationship between two sets of numbers or variables.
- However, correlation does not necessarily mean a direct cause and effect relationship. If it appears that values for one of the variables can be predicted based on the value of the other variable, then there is correlation.

6.11.5 Run Chart

- A run chart tracks the performance of the parameter of interest over time. The X-axis is time and the Y -axis is the value of the parameter.
- A run chart is best used for trend analysis, especially if historical data are available for comparisons with the current trend.
- Ishikawa includes various graphs such as the pie chart, bar graph, compound bar graph, and circle graph under the section that discusses run charts .
- An example of a run chart in software is the weekly number of open problems in the backlog; it shows the development team's workload of software fixes.
- Fig. 6.11.5 shows run time chart.
- An organization's desire is to have their product arrive to their customers on time, but they have noticed that it doesn't take the same amount of time each day of the week.
- They decided to monitor the amount of time it takes to deliver their product over the next few weeks.

Run chart/Time plots

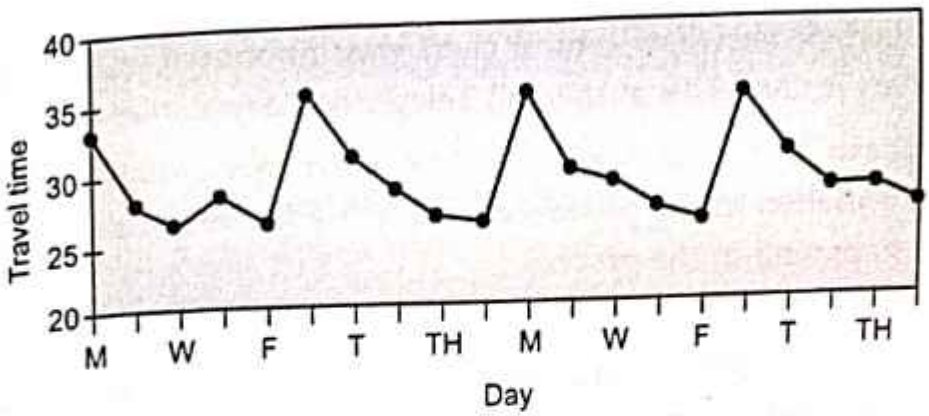

Fig. 6.11.5 Run chart

6.11.6 Cause Effect Diagram

- The cause-and-effect diagram, also known as the fishbone diagram, was developed by Ishikawa and associates in the early 1950s in Japan.
- It has also two other names that are Ishikawa diagram and fishbone because the shape of the diagram looks like the skeleton of a fish to identify quality problems based on their degree of importance.
- Fig. 6.11.6 shows cause effect diagram.

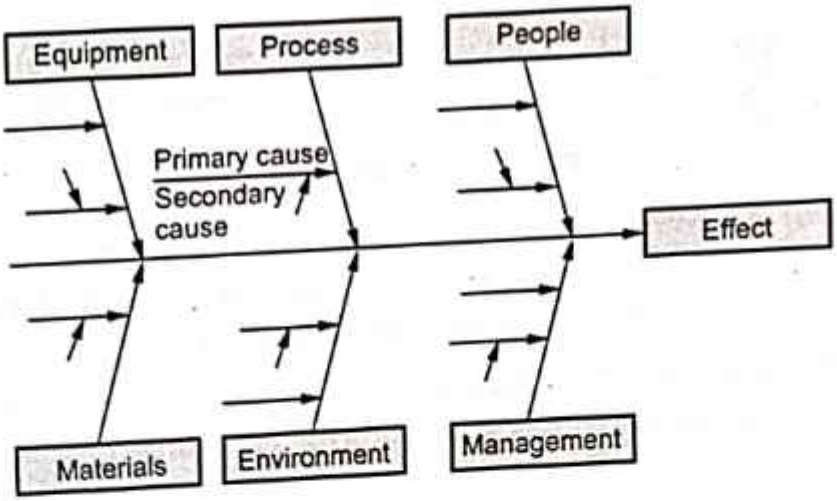

Fig. 6.11.6 Cause effect diagram

- The cause and effect diagram is used to explore all the potential or real causes (or inputs) that result in a single effect (or output).
- Causes are arranged according to their level of importance or detail, resulting in a depiction of relationships and hierarchy of events. This can help you search for root causes, identify areas where there may be problems and compare the relative importance of different causes.

6.11.7 Control Chart

- Control chart or Shewhart control chart was introduced and developed by Walter A. Shewhart in the 1920s at the Bell Telephone Laboratories.
- Control charts is a special form of "run chart that it illustrates the amount and nature of variation in the process over time". Also, it can draw and describe what has been happning in the process.
- The main aim of control chart is to prevent the defects in process. It is very essentiality for different businesses and industries, the reason is that unsatisfactory products or services are more costed than spending expenses of prevention by some tools like control charts.
- Fig. 6.11.7 shows control chart.

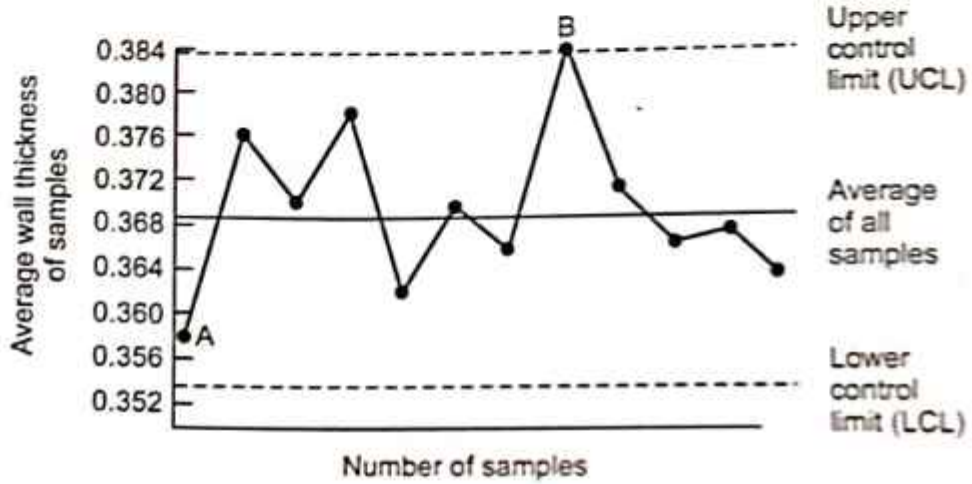

Fig. 6.11.7 Control chart

- It consists of a central line, a pair of control limits (and sometimes a pair of warning limits within the control limits), and values of the parameter of interest plotted on the chart, which represent the state of a process.
- The X -axis is real time. If all values of the parameter are within the control limits and show no particular tendency, the process is regarded as being in a controlled state.
- If they fall outside the control limits or indicate a trend, the process is considered out of control. Such cases call for causal analysis and corrective actions are to be taken.
- Control charts are used to routinely monitor quality. Depending on the number of process characteristics to be monitored, there are two basic types of control charts.
- The first, referred to as a uni-variate control chart, is a graphical display (chart) of one quality characteristic.

- The second, referred to as a multivariate control chart, is a graphical display of a statistic that summarizes or represents more than one quality characteristic.

Review Questions

1. *Explain following terms (any Two)*
   i) Control chart ii) Scatter diagrams iii) Checklists.
   SPPU : Dec-19, End Sem, Marks 6
2. *Enumerate Ishikawa's seven basic quality tools. Explain any two in detail.*
   SPPU : May-19, Dec-18, 19, End Sem, Marks 6
3. *Explain following terms (any Two)*
   i) Checklists ii) Histogram iii) Run Chart. **SPPU : May-19, End Sem, Marks 6**
4. *Explain following terms (any Two)*
   Pareto chart, Scatter diagrams, cause and effect diagrams.
   SPPU : Dec-18, End Sem, Marks 6

6.12 Defect Removal Effectiveness

SPPU : Dec-18, 19, May-19

6.12.1 Defect

- A major test objective is to identify defects. Once identified, defects need to be recorded, monitored, reported and corrected. The primary goal is to prevent defects.
- The defect management process like the entire software development process, should be risk driven, i.e., strategies, priorities and resources should be based on an assessment of the risk.
- Defect measurement should be integrated into the development process and be used by the project team to improve the development process.
- Defect information should be used to improve the process. Imperfect or flawed processes cause most defects.
- Keeping track of all the defects that have been discovered. Keeping track of all the steps required to validate, correct, and take preventative action for a defect.
- A "defect" is something that is caused by a coder making a correctable mistake.
- One of main purposes is to provide defect visibility to enable management to ensure defects are appropriately prioritized. Management must :
  1. Overview all active defect records
  2. Ensure priorities are good

3. If Languishing too long in a given state, act
4. Ensure coders are working on defects of appropriate priority at any given time.

- A defect can be defined in one of two ways
- From the producer's viewpoint : A defect is a deviation from specifications, whether missing, wrong, or extra.
- From the customer's viewpoint : A defect is anything that causes customer dissatisfaction, whether in the requirements or not; this is know as "fit for use."
- The steps below describe a simple defect tracking process :
  1. Execute the test and compare the actual results to the documented expected results.
  2. If a discrepancy exists, log the discrepancy with a status of "open". Supplementary documentation, such as screen prints or program traces, should be attached if available.
  3. The test manager or tester should review the problem log with the appropriate member of the development team to determine if the discrepancy is truly a defect.
  4. Assign the defect to a developer for correction.
  5. Once the defect is corrected, the developer will usually enter a description of the fix applied and update the defect status to "Fixed" or "Retest".
  6. The defect is routed back to the test team for retesting.
  7. Additional regression testing is performed as needed based on the severity and impact of the fix applied.
  8. If the retest result match the expected result, the defect status is updated to "closed". If the test results indicate that the defect is still not fixed, the status is changed to "open" and sent back to the developer.

Classification of Defects

- Defects are of different types. Some defects can lead to method failure. Some defects can be mild. Thus, the defects must be classified according to its impact on the functionality of the application. Defects can be classified as following are of different types of defects :

1. **Critical :** The defects termed as 'critical', needs immediate attention and treatment. A critical defect directly affects the critical and essential functionalities, which may affect a software product or its functionality on a large scale, such as failure of a feature/functionality or the whole system, system crash-down, etc.
2. **Major :** Defects, which are responsible for affecting the core and major functionalities of a software product. Although, these defects does not results into

complete failure of a system, but may bring several major functions of software, to rest.

3. **Minor :** These defects produce minor impact, and does not have any significant influence on a software product. The results of these defects may be seen in the product's working, however, it does not stops users to execute it task, which may be carried out, using some other alternative.
4. **Trivial :** These types of defects, have no impact on the working of a product, and sometimes, it is ignored and skipped, such as spelling or grammatical mistake.

6.12.2 Defect Removal Efficiency

- Defect removal effectiveness is a very important aspect of product quality. A good defect removal process promotes the release of products with lower latent defects, generating high customer confidence.
- Defect Removal Efficiency (DRE) is a powerful metric used to measure test effectiveness. From this metric we come to know how many bugs we found from the set of bugs which we could have found.
- The following is the formula for calculating DRE.

$$\text{Defect removal efficiency} = \frac{\text{Total defect detected during inspection and testing}}{\text{Total defects detected}} \times 100$$

- Using the above defect removal efficiency metric, reviewers/testers might uncover defects during software development that are not critical but still achieve higher defect removal efficiency.
- In spite of high defect removal efficiency the customer might be unhappy, due to the presence of critical defects in the software.
- A defect found early, or before a customer finds it, has value. To the developer, the value lies in a higher quality product that will have a better reputation, or, when the defect is found early the cost to fix it is substantially less than if found later in the project.
- To the customer, the value of a defect found by the developer is that the customer will not encounter the problems caused by the defect.
- If processes, people, and tools were perfect, perhaps we could achieve "zero defects." However, defects are a fact of life with the resources we work with at present.
- Another given is that defects can cause serious problems in a project if they are not managed properly. Defect correction consumes resources and can even cause new defects.

6.12.3 Defect Injection and Defect Removal Activity

- Defects are injected into the product or intermediate deliverables of the product (e.g., design document) at various phases. It is wrong to assume that all defects of software are injected at the beginning of development.
- For the development phases before testing, the development activities themselves are subject to defect injection and the reviews or inspections at end-of-phase activities are the key vehicles for defect removal.
- For the testing phases, the testing itself is for defect removal. When the problems found by testing are fixed incorrectly, there is another chance to inject defects.
- In fact, even for the inspection steps, there are chances for bad fixes.
- Fig. 6.12.1 shows the detailed mechanics of defect injection and removal at each step of the development process.

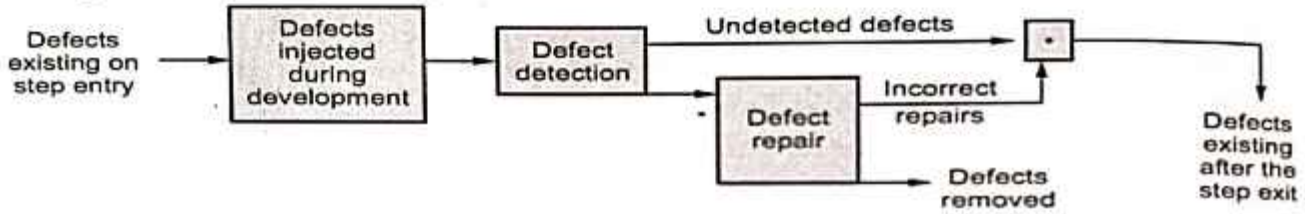

Fig. 6.12.1

- Activities associated with defect injection and removal :

Development Phase	Defect Injection	Defect Removal
Requirements	Requirements-gathering process and the development of programming functional specifications	Requirement analysis and review
High-level design	Design work	High-level design inspections
Low-level design	Design work	Low-level design inspections
Code implementation	Coding	Code inspections
Integration/build	Integration and build process	Build verification testing
Unit test	Bad fixes	Testing itself
Component test	Bad fixes	Testing itself
System test	Bad fixes	Testing itself

Review Questions

1. *Describe in detail defect removal effectiveness.* **SPPU : Dec-18, 19, End Sem, Marks 5**
2. *Describe in detail defect injection and defect removal activities for a development process.* **SPPU : May-19, End Sem, Marks 5**

6.13 Process

- When project size and complexity is increases, automated software tools are required for controlling progress report.
- Project management is a challenging task with many complex responsibilities. Fortunately, there are many tools available to assist with accomplishing the tasks and executing the responsibilities. Some require a computer with supporting software, while others can be used manually. Project managers should choose a project management tool that best suits their management style.
- No one tool addresses all project management needs. Program Evaluation Review Technique (PERT) and Gantt Charts are two of the most commonly used project management tools and are described below. Both of these project management tools can be produced manually or with commercially available project management software.
- PERT is a planning and control tool used for defining and controlling the tasks necessary to complete a project. PERT charts and Critical Path Method (CPM) charts are often used interchangeably; the only difference is how task times are computed. Both charts display the total project with all scheduled tasks shown in sequence.
- The displayed tasks show which ones are in parallel, those tasks that can be performed at the same time. A graphic representation called a "Project Network" or "CPM Diagram" is used to portray graphically the interrelationships of the elements of a project and to show the order in which the activities must be performed.
- **Examples of services that computerized tools**

1. Computerized control of risk management activities are as follows :

a. Category-wise list of software risk and their planned solution dates.

b. Lists of exceptions of software risk items.

2. Project schedule control

a. Delayed activities list

b. Lists of delayed milestones

c. Lists of delays of critical activities

d. Updated activity schedules generated according to progress reports and correction measures applied.

3. Computerized project resource control

a. Project resources allocation plan
   - For activities and software modules
   - For teams and development units
   - For designated time periods, etc.

b. Project resources utilization

c. Project resources utilization exceptions

d. Updated resource allocation plans generated according to progress reports and reaction measures applied.

4. Computerized project budget control

a. Project budget plans
   - For activity and software module
   - For teams and development units
   - For designated time periods, etc.

b. Project budget utilization reports

c. Project budget utilization deviations

d. Updated budget plans generated according to progress reports ports and correction measures

❑❑❑

SOLVED MODEL QUESTION PAPER (In Sem)

(As Per 2019 Pattern)

Software Testing and Quality Assurance

B.E. (Computer) Semester - VII (Elective IV)

Time : 1 Hour] [Maximum Marks : 30

N.B. :

1) *Attempt Q.1 or Q.2, Q.3 or Q.4.*
2) *Neat diagrams must be drawn wherever necessary.*
3) *Figures to the right side indicate full marks.*
4) *Assume suitable data if necessary.*

Q.1 a) *Define softare quality. List and explain core components of quality.* **[Refer section 1.1]** [5]

b) *What are the reasons for software failures ?* **[Refer section 1.5.1]** [4]

c) *Explain software development life cycle model.* **[Refer section 1.17.1]** [6]

OR

Q.2 a) *What are the pillars of quality management system ?* **[Refer section 1.22]** [8]

b) *What is software testing ? Explain objective of testing.* **[Refer section 1.3]** [3]

c) *Explain following terms : i) Benchmarking ii) Matrics.* **[Refer section 1.11]** [4]

Q.3 a) *Discuss entry and exit criteria.* **[Refer section 2.8].** [5]

b) *How to write a test plan ?* **[Refer section 2.1.1]** [4]

c) *What is roles of tester ? Explain purpose of test plan.* **[Refer sections 2.5 and 2.6]** [6]

OR

Q.4 a) *Explain metrics importance in software testing.* **[Refer section 2.14]** [7]

b) *What is use case testing ? Explain characteristics of use case testing.* **[Refer section 2.10]** [8]

SOLVED MODEL QUESTION PAPER (End Sem)

(As Per 2019 Pattern)

Software Testing and Quality Assurance

B.E. (Computer) Semester - VII (Elective IV)

Time : $2\frac{1}{2}$ Hours] [Maximum Marks : 70

N.B. :

1) *Attempt Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.*
2) *Neat diagrams must be drawn wherever necessary.*
3) *Figures to the right side indicate full marks.*
4) *Assume suitable data, if necessary.*

Q.1 a) *What is white box testing and black box testing ? Explain difference between white box testing and black box testing.* **[Refer section 3.1]** [8]

b) *Discuss error guessing of experienced based techniques.* **[Refer section 3.6.1]** [5]

c) *Explain the following :*
Informal reviews, walkthroughs. **[Refer sections 3.2.2 and 3.2.3]** [5]

OR

Q.2 a) *What is non-functional testing ? Explain any one non-functional testing.* **[Refer section 3.8]** [8]

b) *Explain difference between functional and non-functional testing.* **[Refer section 3.8.17]** [5]

c) *Write short note on functional testing and unit testing.* **[Refer sections 3.7.1 and 3.7.2]** [5]

Q.3 a) *What is software quality control? Explain in brief. Discuss quality challenges.* **[Refer section 4.14]** [10]

b) *What is CMM ? Explain CMM levels.* **[Refer section 4.21]** [7]

OR

Q.4 a) *What is software quality model ? Explain Boehm model.* **[Refer sections 4.15 and 4.15.2]** [10]

b) *Explain relationship between quality and productivity. What is difference between invention and innovation ?* **[Refer section 4.2]** [7]

Q.5 a) *What is automation testing ? What tests should be automated ? What are benefits of automation testing ?* **[Refer sections 5.1 and 5.4]** [10]

b) *What is Selenium? Explain various Selenium tool suits.* **[Refer sections 5.6 and 5.7]** [8]

OR

Q.6 a) *Explain following performance testing tools : Apache Jmeter, LoadRunner, WebLOAD.* **[Refer section 5.11.1]** [10]

b) *How to choose automation testing tools ? Explain tool selection process.* **[Refer section 5.5]** [8]

Q.7 a) *What does SQA ensure ? What are the goals of SQA ?* **[Refer section 6.2]** [5]

b) *What is ISO standard ? What are its advantages ?* **[Refer section 6.5]** [4]

c) *Enumerate Ishikawa's seven basic quality tools. Explain any two in detail.* **[Refer section 6.11]** [8]

OR

Q.8 a) *Explain with example product quality metric.* **[Refer section 6.8]** [5]

b) *What is six sigma ? Explain the terms DMAIC and DMADV.* **[Refer section 6.4]** [5]

c) *Describe in detail defect removal effectiveness.* **[Refer section 6.12]** [7]

❑❑❑