UNIT I

INTRODUCTION

Testing as an Engineering Activity – Testing as a Process –Testing Maturity Model- Testing axioms – Basic definitions – Software Testing Principles – The Tester's Role in a Software Development Organization – Origins of Defects – Cost of defects – Defect Classes – The Defect Repository and Test Design – Defect Examples – Developer/Tester Support of Developing a Defect Repository – Defect Prevention strategies.

1.1 TESTING AS AN ENGINEERING ACTIVITY

Testing as an Engineering Activity

- Software systems are becoming more challenging to build. They are playing an increasingly important role in the society.
- People with software development skills are in demand. There is pressure for software development professional to focus on quality issues.
- Poor quality software that can cause loss of life or property is no longer acceptable to society. Failure can result in catastrophic losses.
- Conditions demand software development staff with interest and training in the areas of software product and process quality.
- Highly qualified staff makes sure that software products are built on time, within budget, and are of the highest quality.
- Quality is determined by attributes such as reliability, correctness, usability and the ability to meet all user requirements.
- The education and training of engineers in each engineering discipline is based on the teaching of related scientific principles as shown in Figure.
- A joint task force has been formed to define a body of knowledge that covers the software engineering discipline, to discuss the nature of education for this new profession and to define a code of ethics for the software engineering discipline.
- The members of the joint task force are IEEE Computer Society and the Association of Computing Machinery (ACM).
- Using an engineering approach to software development means the following
  - ✓ The development of the process is well understood.
  - ✓ Projects are planned.
  - ✓ Life cycle models are defined and adhered to.
  - ✓ Standards are in place for product and process
  - ✓ Measurements are employed to evaluate product and process quality.
  - ✓ Components are reused.

- Validation and Verification process play a key role in quality determination. Engineers should have proper education, training and certification.

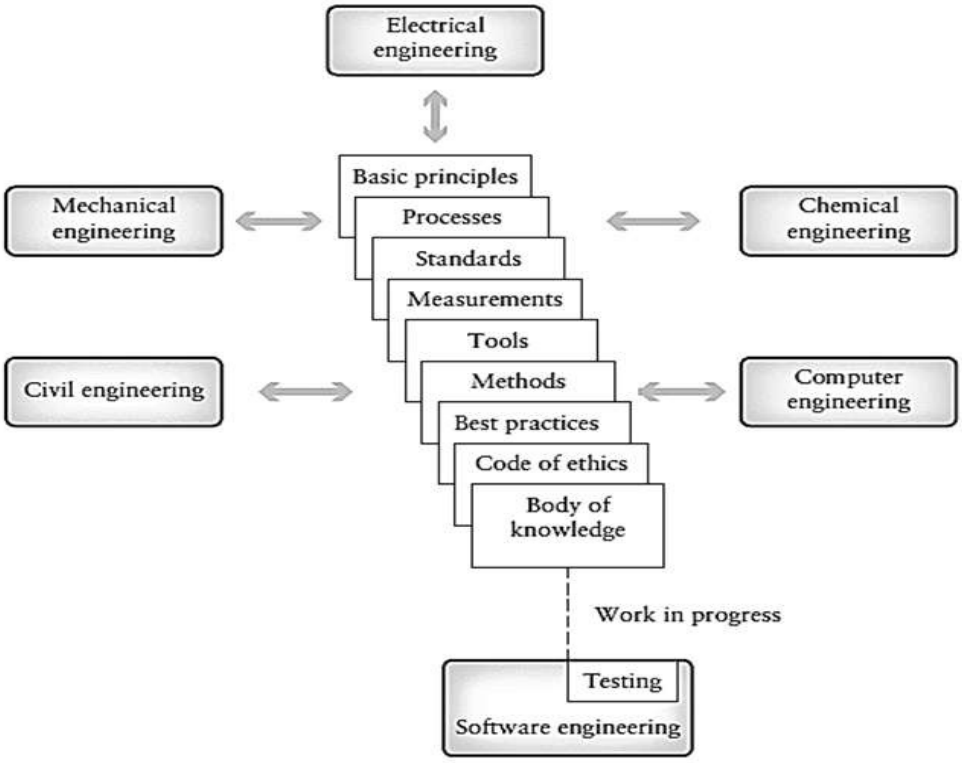


Figure 1.1: Elements of Engineering Disciplines

- A test specialist is one whose education is based on the principles, practices and processes that constitute the software engineering discipline and whose specific focus is on one area of that discipline, software testing.
- A test specialist who is trained as an engineer should have knowledge on the following
  - ✓ Test related principles,
  - ✓ Processes,
  - ✓ Measurements,
  - ✓ Standards,
  - ✓ Plans,
  - ✓ Tools and methods
  - ✓ How to apply them to the testing tasks.

- Testing concepts, are not an isolated collection of technical and managerial activities, it should be integrated within the context of a quality testing process.

1.1.1 The Role of Process in Software Quality

- The need for software products of high quality has pressured those in the software profession to identify and quantify quality factors such as usability, testability, maintainability and reliability and to identify engineering practices that support the production of quality products having these favourable attributes.

- Among the practices identified that contribute to the development of high-quality software are:
  - ✓ Project Planning
  - ✓ Requirements Management
  - ✓ Development of formal specification
- Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artifacts, such as project and test plans, design documents, code, and manuals.
- Adding individual practices to an existing software development practices in an adhoc way is not satisfactory.
- The software development process is similar to any other engineering activity, it must be engineered. It must be
  - ✓ Designed
  - ✓ Implemented
  - ✓ Evaluated
  - ✓ Maintained

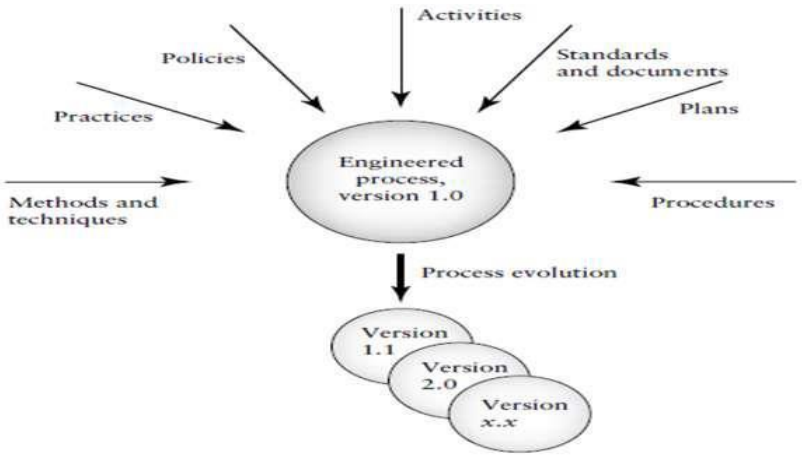


Figure 1.2: Components of an Engineering Process

- Similar to other engineering process, a software development process must evolve in a consistent and predictable manner.
- Both best technical and managerial practices must be integrated in a systematic manner.
- Most of the software process improvement modes accepted by the industries are high level modes.
- They focus on the software as a whole and do not support specific development of any sub process such as design and testing.

1.2 TESTING AS A PROCESS

The software development process is described as a series of phases, procedures and steps that result in the production of software products, embedded within the software development process are several other processes including testing.

- Testing is described as a group of procedures carried out to evaluate some aspect of a piece of software.
- Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.
- Testing covers both validation and verification activities. Testing includes the following,
  - ✓ Technical reviews
  - ✓ Test planning
  - ✓ Test tracking
  - ✓ Test case design
  - ✓ Unit test
  - ✓ Integration test
  - ✓ System test
  - ✓ Acceptance test
  - ✓ Usability test
- Testing can also be described as a dual-purpose process.
- It reveals defects and evaluates quality attributes of the software such as:
  - ✓ Reliability
  - ✓ Security
  - ✓ Usability
  - ✓ Correctness
- The debugging process begins after testing has been carried out and the tester has noted that the software is not behaving as specified.
- Debugging is the process of:
  - ✓ Locating the fault or defect
  - ✓ Repairing the code
  - ✓ Retesting the code
- Testing has economic, technical and managerial aspects.
- Testing must be managed. Organizational policy for testing must be defined and documented.
- Testing is related to two processes
  1. Verification
  2. Validation

1.2.1 Validation

Validation is the process of evaluating a software system or components during or at the end of the development cycle in order to determine whether it satisfies specified requirements.

- Validation is usually associated with traditionally execution-based testing, that is, exercising the code with test cases.

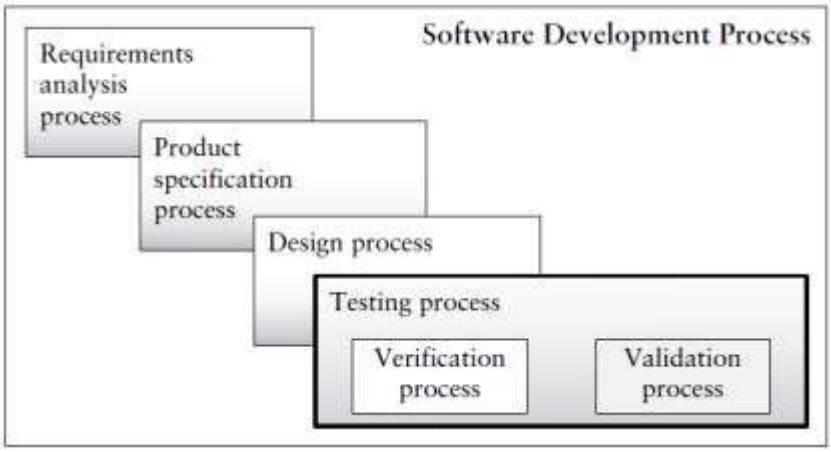


Figure1.3: Process Embedded in the Software Development Processes

1.2.2 Verification

Verification is the process of evaluating a software system or component to determine whether the products of a given development phase satisfies the conditions imposed at the start of that phase.

- Verification is usually associated with inspections and reviews of software deliverables.

1.3 TESTING MATURITY MODEL

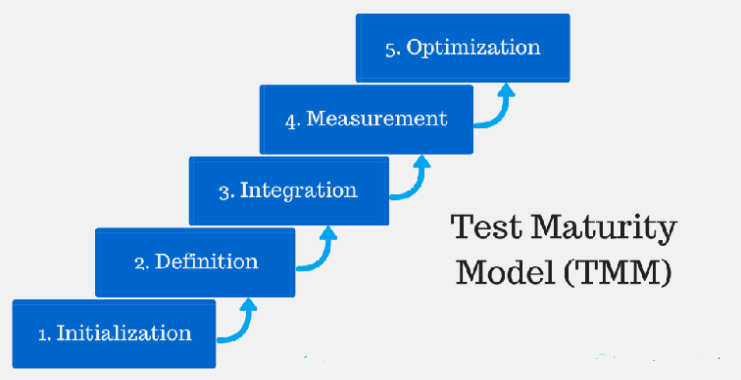
Test maturity model is based on capability maturity model specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

1. Initialization: this is the first level of TMM. In this level, there is no defined testing process. There will be exploratory or ad-hoc testing carried out on the software. The main aim of this level is to make sure that the software is running fine and there are no roadblocks. There are no quality checks before delivering the product.
2. Definition: as name suggests, this level is all about defining the requirements. In this second level of TMM, test strategies, test plans and test cases are developed according to the requirements given by the client to build a software. The main aim of this level is to make sure that the product runs according to the requirements and to achieve that test cases and test plan documents are created and followed religiously.
3. Integration: this is the third level of TMM. As name suggests, in this level testing is integrated with the

4. software life cycle and becomes a part of it. For example, V model has both development and testing phases. Testing comes after the development is over and the software under test is handed over to the professional testing team. Testing is carried out independently. The whole testing objectives are based on the risk management.

5. Management and Measurement: managing and measuring the requirements are taken care in this level of TMM. This is the fourth level where testing becomes the part of all the activities in software life cycle. From reviewing the gathered requirements and design of the software to deciding the quality criteria, is included. This builds a clearer picture for the organization which in turn helps them to achieve the required quality.
6. Optimization: this is the last level of TMM. The fifth level is responsible for optimizing the test process itself. In other words, testing processes are tested and measures are taken to improve them iteration by iteration.This testing is mainly carried out by the help of different tools. Also, in this level defects are prevented by improving the processes throughout the software life cycle so main focus is defect prevention rather than defect detection in each phase.

Each level has its own role and importance. The main goal of each level is well defined and has its own structure. The main idea of TMM was taken from Capability Maturity Model (CMM) which is basically a structured tool applied to software development and also used as a model to aid different business processes. Here "Maturity" is measured by the degree of optimized processes. It starts from ad hoc practices then defining of formal steps and documentation, managing results and reviewing documents to optimizing process and results upto the expectations.



Benefits of TMM:

1. Organized: we have discussed all the 5 levels of TMM. Each level is well defined and has a particular aim to achieve. This makes TMM a well-organized model with clear goals.
2. Assurance of quality: when we integrate testing with all the phases of software life cycle, higher quality is achievable. Testing of test processes would optimize the results which in turn gives assurance of good quality product.
3. Defect prevention: as I mentioned earlier that TMM focuses on defect prevention rather than defect detection by making testing process a part of all phases of software life cycle. This would ensure that maximum defects are prevented and final product is mostly defect free.
4. Clear requirements: when requirements and designs are reviewed and test plans and test cases are tested against requirements, the main test objectives are clearer and hence, testing is more accurate.

1.4 TESTING AXIOMS

The test axioms are a context neutral set of rules for testing that identify the critical thinking process and motivations for all test approaches. The axioms represent mechanism designed to meet the objective of the testing in scope. A mechanism may be well defined, documented process, an informal or even ad-hoc activity - but that mechanism must be understood and used by participants in the test.

- **It's Impossible to Test a Program Completely:** As a new tester, you might believe that you can approach a piece of software, fully test it, find all the bugs, and assure that the software is perfect. Unfortunately, this isn't possible, even with the simplest programs, due to four key reasons:

  - ✓ The number of possible inputs is very large.
  - ✓ The number of possible outputs is very large.
  - ✓ The number of paths through the software is very large.
  - ✓ The software specification is subjective.
  - ✓ You might say that a bug is in the eye of the beholder.

- **Software Testing Is a Risk-Based Exercise:** If you decide not to test every possible test scenario, you've chosen to take on risk. This may all sound pretty scary. You can't test everything, and if you don't, you will likely miss bugs. The product has to be released, so you will need to stop testing, but if you stop too soon, there will still be areas untested. What do you do?

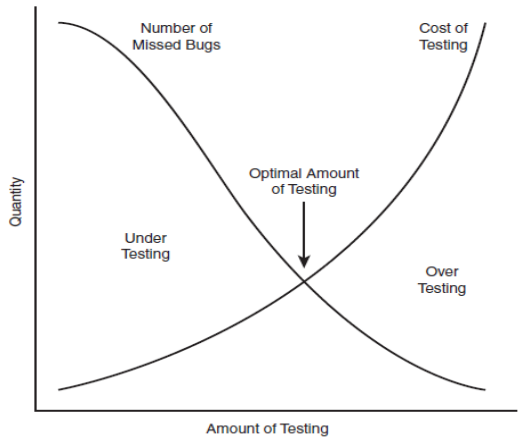


Figure 1.4: Testing Axioms

- One key concept that software testers need to learn is how to reduce the huge domain of possible tests into a manageable set, and how to make wise risk-based decisions on what's important to test and what's not.
- Figure shows the relationship between the amount of testing performed and the number of bugs found. If you attempt to test everything, the costs go up dramatically and the number of missed bugs declines to the point that it's no longer cost effective to continue.
- If you cut the testing short or make poor decisions of what to test, the costs are low but you'll miss a lot of bugs. The goal is to hit that optimal amount of testing so that you don't test too much or too little.

- **Testing Can't Show That Bugs Don't Exist**: You can perform your tests, find and report bugs, but at no point can you guarantee that there are no longer any bugs to find. You can only continue your testing and possibly find more.
- **The More Bugs You Find, the More Bugs There Are:** There are even more similarities between real bugs and software bugs. Both types tend to come in groups. If you see one, odds are there will be more nearby.
- Frequently, a tester will go for long spells without finding a bug. He'll then find one bug, then quickly another and another.
- There are several reasons for this:

  - ✓ **Programmers have bad days.** Like all of us, programmers can have off days. Code written one day may be perfect; code written another may be sloppy. One bug can be a tell-tale sign that there are more nearby.
  - ✓ **Programmers often make the same mistake.** Everyone has habits. A programmer who is prone to a certain error will often repeat it.
  - ✓ **Some bugs are really just the tip of the iceberg.** Very often the software's design or architecture has a fundamental problem. A tester will find several bugs that at first may seem unrelated but eventually are discovered to have one primary serious cause.
- It's important to note that the inverse of this ‒bugs follow bugs‖ idea is true, as well. If you fail to find bugs no matter how hard you try, it may very well be that the software was cleanly written and that there are indeed few if any bugs to be found.
- **Not All the Bugs You Find Will Be Fixed:** One of the sad realities of software testing is that even after all your hard work, not every bug you find will be fixed. Now, don't be disappointed—this doesn't mean that you've failed in achieving your goal as a software tester, nor does it mean that you or your team will release a poor quality product.
- There are several reasons why you might choose not to fix a bug:
  - ✓ There's not enough time
  - ✓ It's really not a bug - ‒It's not a bug, it's a feature!‖
  - ✓ It's too risky to fix
  - ✓ It's just not worth it.
- **When a Bug's a Bug Is Difficult to Say:** If there's a problem in the software but no one ever discovers it—not programmers, not testers, and not even a single customer—is it a bug?
  - ✓ The software doesn't do something that the product specification says it should do.
  - ✓ The software does something that the product specification says it shouldn't do.
  - ✓ The software does something that the product specification doesn't mention.
  - ✓ The software doesn't do something that the product specification doesn't mention.
  - ✓ The software is difficult to understand, hard to use, slow, or—in the software tester's eyes—will be viewed by the end user as just plain not right.
- Following these rules helps clarify the dilemma by making a bug a bug only if it's observed.
- To claim that the software does or doesn't do ‒something‖ implies that the software was run and that ‒something‖ or the lack of ‒something‖ was witnessed.
- Since you can't report on what you didn't see, you can't claim that a bug exists if you didn't see it.
- Here's another way to think of it. It's not uncommon for two people to have completely different opinions on the quality of a software product.

❖ One may say that the program is incredibly buggy and the other may say that it's perfect. How can both be right? The answer is that one has used the product in a way that reveals lots of bugs. The other hasn't.

❖ **Product Specifications Are Never Final**: As a software tester, you must assume that the spec will change. Features will be added that you didn't plan to test. Features will be changed or even deleted that you had already tested and reported bugs on.

❖ **Software Testers Aren't the Most Popular Members of a Project Team**: The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

❖ Here are a couple of tips to keep the peace with your fellow teammates:

- ✓ Find bugs early.
- ✓ Don't always report bad news
- ✓ Temper your enthusiasm

1.5 BASIC DEFINITIONS

Many of the definitions generally used in testing are based on the terms described in the IEEE Standards Collection for Software Engineering. The standards collection includes the IEEE Standard Glossary of Software Engineering Terminology, which is a dictionary of software engineering vocabulary.

❖ **Errors**: An error is a mistake, misconception, or misunderstanding on the part of a software developer.

- ✓ An error is a mistake, misconception, or misunderstanding on the part of a software developer.
- ✓ In the category of developer we include software engineers, programmers, analysts, and testers. For example, a developer may misunderstand a design notation, or a programmer might type a variable name incorrectly.

Example:

- Memory bit got stuck but CPU does not access this data
- Software ―bug in a subroutine is not ―visible while the subroutine is not called.

❖ **Faults (Defects):** A fault (defect) is introduced into the software as the result of an error. It is an irregularity in the software that may cause it to behave incorrectly, and not according to its specification.

- ✓ A fault (defect) is introduced into the software as the result of an error.
- ✓ It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification.
- ✓ Faults or defects are sometimes called ―bugs.‖ Use of the latter term trivializes the impact faults have on software quality. Use of the term ―defect is also associated with software artifacts such as requirements and design documents. Defects occurring in these artifacts are also caused by errors and are usually detected in the review process.

Examples:

- Software bug
- Random hardware fault
- Memory bit ―stuck
- Omission or commission fault in data transfer etc.

❖ **Failures:** A failure is the inability of a software system or component to perform its required functions within specified performance requirements.

- ✓ Presence of an error might cause a whole system to deviate from its required operation
- ✓ One of the goals of safety-critical systems is that error should not result in system failure
- ✓ During execution of a software component or system, a tester, developer, or user observes that it does not produce the expected results.
- ✓ In some cases a particular type of misbehaviour indicates a certain type of fault is present. We can say that the type of misbehaviour is a symptom of the fault.
- ✓ An experienced developer/tester will have a knowledge base of fault/symptoms/ failure cases stored in memory.
- ✓ Incorrect behaviour can include producing incorrect values for output variables, an incorrect response on the part of a device, or an incorrect image on a screen.
- ✓ During development failures are usually observed by testers, and faults are located and repaired by developers.
- ✓ When the software is in operation, users may observe failures which are reported back to the development organization so repairs can be made.
- ✓ A fault in the code does not always produce a failure. In fact, faulty software may operate over a long period of time without exhibiting any incorrect behaviour.
- ✓ However when the proper conditions occur the fault will manifest itself as a failure.
- ✓ Voas is among the researchers who discuss these conditions, which are as follows:
  - The input to the software must cause the faulty statement to be executed.
  - The faulty statement must produce a different result than the correct statement.
  - This event produces an incorrect internal state for the software.
  - The incorrect internal state must propagate to the output, so that the result of the fault is observable.
- ✓ Software that easily reveals its' faults as failures is said to be more testable.
- ✓ From the testers point-of-view this is a desirable software attribute. Testers need to work with designers to insure that software is testable.

❖ **Test Cases**: To detect defects in a piece of software the tester selects a set of input data and then executes the software with the input data under a particular set of conditions. A test case is a test-related item which contains the following information:

- ✓ **A set of test inputs**. These are data items received from an external source by the code under

test. The external source can be hardware, software, or human.

- ✓ **Execution conditions.** These are conditions required for running the test, for example, a certain state of a database, or a configuration of a hardware device.
- ✓ **Expected outputs**. These are the specified results to be produced by the code under test.

❖ **Test**: A test is a group of related test cases, or a group of related test cases and test procedures.

❖ **Test Oracle**: A test oracle is a document, or piece of software that allows testers to determine whether a test has been passed or failed.

❖ **Test Bed**: A test bed is an environment that contains all the hardware and software needed to test a software component or a software system.

❖ **Software Quality**: Software quality can either be defined as

- ✓ Quality relates to the degree to which a system, system component, or process meets specified requirements.
- ✓ Quality relates to the degree to which a system, system component, or process meets customer or user needs, or expectations.

❖ **Metric:** A metric is a quantitative measure of the degree to which a system, system component, or process has a given attribute**.** A quality metric is a quantitative measurement of the degree to which an item possesses a given quality attribute. Some examples of quality metric are,

- ✓ **Correctness**—the degree to which the system performs its intended function
- ✓ **Reliability**—the degree to which the software is expected to perform its required functions under stated conditions for a stated period of time
- ✓ **Usability**—relates to the degree of effort needed to learn, operate, prepare input, and interpret output of the software
- ✓ **Integrity**—relates to the system's ability to withstand both intentional and accidental attacks
- ✓ **Portability**—relates to the ability of the software to be transferred from one environment to another
- ✓ **Maintainability**—the effort needed to make changes in the software
- ✓ **Interoperability**—the effort needed to link or couple one system to another.

1.6 SOFTWARE TESTING PRINCIPLES

Testing principles are important to test specialists and engineers because they are the foundation for developing testing knowledge and acquiring testing skills. They also provide guidance for defining testing activities.

❖ A principle can be defined as,

- ✓ A general or fundamental law.

- ✓ A rule or code of conduct.
- ✓ The laws or facts of nature underlying the working of an artificial device.

❖ In the software domain, principles may also refer to rules or codes of conduct relating to professionals, who design, develop, test, and maintain software systems.

❖ The following are a set of testing principles

Principle 1: ***Testing is the process of exercising a software component using a selected set of test cases, with the intent of revealing defects, and evaluating quality.***

❖ This principle supports testing as an execution-based activity to detect defects. It also supports the separation of testing from debugging since the intent of debugging is to locate defects and repair the software.

❖ The term ―software component‖ means any unit of software ranging in size and complexity from an individual procedure or method, to an entire software system.

❖ The term ―defects‖ represents any deviations in the software that have a negative impact on its functionality, performance, reliability, security, and/or any other of its specified quality attributes.

Principle 2: ***When the test objective is to detect defects, then a good test case is one that has a high probability of revealing yet undetected defects.***

❖ Testers must carry out testing in the same way as scientists carry out experiments.

❖ Testers need to create a hypothesis and work towards proving or disproving it, it means he/she must prove the presence or absence or a particular type of defect.

Principle 3: ***Test results should be inspected meticulously.***

❖ Testers need to carefully inspect and interpret test results. Several erroneous and costly scenarios may occur if care is not taken.

❖ A failure may be overlooked, and the test may be granted a ―pass‖ status when in reality the software has failed the test.

❖ Testing may continue based on erroneous test results.

❖ The defect may be revealed at some later stage of testing, but in that case it may be more costly and difficult to locate and repair.

Principle 4: ***A test case must contain the expected output or result.***

❖ The test case is of no value unless there is an explicit statement of the expected outputs or results. Expected outputs allow the tester to determine

- ✓ Whether a defect has been revealed,
- ✓ Pass/fail status for the test.

❖ It is very important to have a correct statement of the output so that time is not spent due to misconceptions about the outcome of a test.

- The specification of test inputs and outputs should be part of test design activities.

Principle 5: ***Test cases should be developed for both valid and invalid input conditions.***

- A tester must not assume that the software under test will always be provided with valid inputs. Inputs may be incorrect for several reasons.
- Software users may have misunderstandings, or lack information about the nature of the inputs
- They often make typographical errors even when complete/correct information is available.
- Devices may also provide invalid inputs due to erroneous conditions and malfunctions.

Principle 6: ***The probability of the existence of additional defects in a software component is proportional to the number of defects already detected in that component.***

- The higher the number of defects already detected in a component, the more likely it is to have additional defects when it undergoes further testing.
- If there are two components A and B, and testers have found 20 defects in A and 3 defects in B, then the probability of the existence of additional defects in A is higher than B.

Principle 7: ***Testing should be carried out by a group that is independent of the development group.***

- This principle is true for psychological as well as practical reasons. It is difficult for a developer to admit that software he/she has created and developed can be faulty.
- Testers must realize that
  - ✓ Developers have a great pride in their work,
  - ✓ Practically it is difficult for the developer to conceptualize where defects could be found.

Principle 8: ***Tests must be repeatable and reusable.***

- The tester needs to record the exact conditions of the test, any special events that occurred, equipment used, and a carefully note the results.
- This information is very useful to the developers when the code is returned for debugging so that they can duplicate test conditions.
- It is also useful for tests that need to be repeated after defect repair.

Principle 9: ***Testing should be planned.***

- Test plans should be developed for each level of testing. The objective for each level should be described in the associated plan. The objectives should be stated as quantitatively as possible.

Principle 10: ***Testing activities should be integrated into the software life cycle.***

- Testing activity should be integrated into the software life cycle starting as early as in the requirements analysis phase, and continue on throughout the software life cycle in parallel with development activities.

Principle 11: ***Testing is a creative and challenging task.***

- Difficulties and challenges for the tester include the following:
- A tester needs to have good knowledge of the software engineering discipline.
- A tester needs to have knowledge from both experience and education about software specification, designed, and developed.
- A tester needs to be able to manage many details.
- A tester needs to have knowledge of fault types and where faults of a certain type might occur in code construction.
- A tester needs to reason like a scientist and make hypotheses that relate to presence of specific types of defects.
- A tester needs to have a good understanding of the problem domain of the software that he/she is testing. Familiarly with a domain may come from educational, training, and work related experiences. A tester needs to create and document test cases.
- To design the test cases the tester must select inputs often from a very wide domain.
- The selected test cases should have the highest probability of revealing a defect. Familiarly with the domain is essential.
- A tester needs to design and record test procedures for running the tests.
  A tester needs to plan for testing and allocate proper resources.
- A tester needs to execute the tests and is responsible for recording results.
- A tester needs to analyse test results and decide on success or failure for a test.
- This involves understanding and keeping track of huge amount of detailed information.
- A tester needs to learn to use tools and keep updated of the newest test tools.
- A tester needs to work and cooperate with requirements engineers, designers, and developers, and often must establish a working relationship with clients and users.
- A tester needs to be educated and trained in this specialized area.

1.7 THE TESTER'S ROLE IN A SOFTWARE DEVELOPMENT ORGANIZATION

The tester's job is to

- ✓ ***Reveal defects,***
- ✓ ***Find weak points,***
- ✓ ***Inconsistent behaviour,***
- ✓ ***Circumstances where the software does not work as expected.***
- ✓ ***It is difficult for developers to effectively test their own code.***

- A tester needs very good programming experience in order to understand how code is constructed, and to know where and what types of, defects could occur.
- A tester should work with the developers to produce high-quality software that meets the customers' requirements.

- Teams of testers and developers are very common in industry, and projects should have a correct developer/tester ratio. The ratio will vary depending on
  - ✓ Available resources
  - ✓ Type of project
  - ✓ TMM level
  - ✓ Nature of the project
  - ✓ Project Schedules
- Testers also need to work with ***requirements engineers*** to make sure that requirements are testable, and to plan for system and acceptance test.
- Testers also need to work with ***designers*** to plan for integration and unit test.
- ***Test managers*** need to cooperate with ***project managers*** in order to develop reasonable test plans, and with ***upper management*** to provide input for the development and maintenance of organizational
  - ✓ Testing standards, Polices, Goals.
- Testers also need to cooperate with ***software quality assurance*** staff and software ***engineering*** process group members.
- Testers are a part of the development group. They concentrate on testing. They may be part of the software quality assurance group.
- Testers are specialists, their main function is to plan, execute, record, and analyse tests. They do not debug software.
- When defects are detected during testing, software should be returned to the developers.
- The developers locate the defect and repair the code. The developers have a detailed understanding of the code, and they can perform debugging better.
- Testers need the support of management. Testers ensure that developers release code with few or no defects, and that marketers can deliver software that satisfies the customers' requirements, and is reliable, usable, and correct.

1.8 ORIGINS OF DEFECTS

Defects have negative effects on software users. Software engineers work very hard to produce high-quality software with a low number of defects.

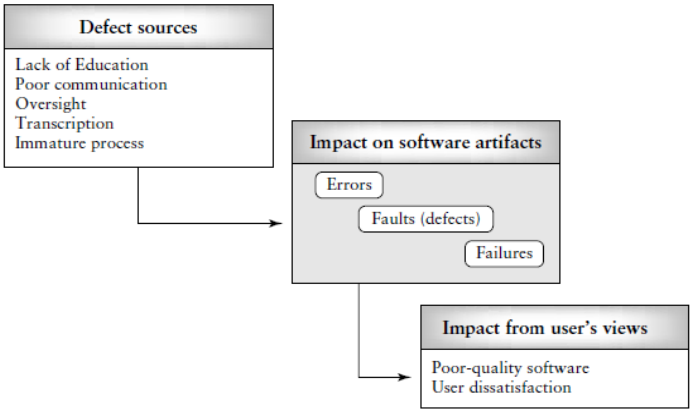


Figure 1.5: Origins of Defects

- **Education:** The software engineer did not have the proper educational background to prepare the software artifact.
- **Communication:** The software engineer was not informed about something by a colleague.
- **Oversight:** The software engineer omitted to do something.
- **Transcription:** The software engineer knows what to do, but makes a mistake in doing it.

- **Process:** The process used by the software engineer misdirected his/her actions.The impact of defect on the user ranges from a minor inconvenience to rendering the software unfit for use.
- Testers have to discover these defects before the software is in operation. The results of the tests are analyzed to determine whether the software has behaved correctly.

- In this scenario a tester develops ***hypotheses*** about possible defects. Test cases are then designed based on the hypotheses.
- The hypotheses are used to,
  - ✓ Design test cases.
  - ✓ Design test procedures.
  - ✓ Assemble test sets.
  - ✓ Select the testing levels suitable for the tests.
  - ✓ Evaluate the results of the tests.

- **FAULT MODEL:** A fault (defect) model can be described as a link between the error made, and the fault/defect in the software.

- **DEFECT REPOSITORY:** To increase the effectiveness of their testing and debugging processes, software organizations need to initiate the creation of a defect database, or defect repository. The defect repository supports storage and retrieval of defect data from all projects in a centrally accessible location.

1.9 COST OF DEFECT

The cost of defects can be measured by the impact of the defects. Earlier the defect is found lesser is the cost of defect. For example if error is found in the requirement specification then it is somewhat cheap to fix it.

- The correction to the requirement specification can be done and then it can be re-issued. In the same way when defect or error is found in the design then the design can be corrected and it can be re-issued.

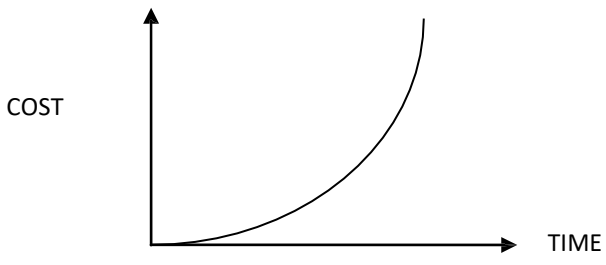


Figure 1.6: Cost of Defect

- But if the error is not caught in the specification and is not found till the user acceptance then the cost to fix those errors or defects will be way too expensive.
- If the error is made and the consequent defect is detected in the **requirements phase** then it is relatively cheap to fix it.

- Similarly if an error is made and the consequent defect is found in the **design phase** then the design can be corrected and reissued with relatively little expansive,
- The same applies for **construction phase**.
- If however, a defect is introduced in the requirement specification and it is not detected until **acceptance testing** or even once the system has been implemented then it will be much more expansive to fix.
- This is because rework will be needed in the specification and design before changes can be made in construction; because one defect in requirement may well propagate into several places in the design and code.
- All the testing work done-to that point will need to be repeated in order to reach the confidence level in the software that we require.
- It is quite often the case that defects detected at a very late stage, depending on how serious they are, and not corrected because the cost of doing is too expensive.

1.10 DEFECT CLASSES – DEFECT REPOSITORY AND TEST DESIGN

Defects can be classified in many ways. The four classes of defects are as follows,

- ✓ ***Requirements and specifications defects,***
- ✓ ***Design defects,***
- ✓ ***Code defects,***
- ✓ ***Testing defects***

Requirements and Specifications Defects

- The beginning of the software life cycle is important for ensuring high quality in the software being developed.
- Defects injected in early phases can be very difficult to remove in later phases.
- Since many requirements documents are written using a natural language representation, they may become
  - ✓ Ambiguous,
  - ✓ Contradictory,
  - ✓ Unclear,
  - ✓ Redundant,
  - ✓ Imprecise.

- Some specific requirements/specification defects are:
    1. **Functional Description Defects**
       The overall description of what the product does, and how it should behave (inputs/outputs), is incorrect, ambiguous, and/or incomplete.

    2. **Feature Defects**
       Features are described as distinguishing characteristics of a software component or system. Feature defects are due to feature descriptions that are missing, incorrect, incomplete, or unnecessary.
    3. **Feature Interaction Defects**
       These are due to an incorrect description of how the features should interact with each other.
    4. **Interface Description Defects**
       These are defects that occur in the description of how the target software is to interface with external software, hardware, and users.

Design Defects

- Design defects occur when the following are incorrectly designed,
    - ✓ System components,
    - ✓ Interactions between system components,
    - ✓ Interactions between the components and outside software/hardware, or users

- It includes defects in the design of algorithms, control, logic, data elements, module interface descriptions, and external software/hardware/user interface descriptions.
- The design defects are,
    1. **Algorithmic and Processing Defects**
       These occur when the processing steps in the algorithm as described by the pseudo code are incorrect.
    2. **Control, Logic, and Sequence Defects**
       Control defects occur when logic flow in the pseudo code is not correct.
    3. **Data Defects**
       These are associated with incorrect design of data structures.
    4. **Module Interface Description Defects**
       These defects occur because of incorrect or inconsistent usage of parameter types, incorrect number of parameters or incorrect ordering of parameters.
    5. **Functional Description Defects**
       The defects in this category include incorrect, missing, or unclear design elements.
    6. **External Interface Description Defects**

       These are derived from incorrect design descriptions for interfaces with COTS components, external software systems, databases, and hardware devices.

Coding Defects

❖ Coding defects are derived from errors in implementing the code. Coding defects classes are similar to design defect classes. Some coding defects come from a failure to understand programming language constructs, and miscommunication with the designers

1. **Algorithmic and Processing Defects**

   Code related algorithm and processing defects include
   - ✓ Unchecked overflow and underflow conditions,
   - ✓ Comparing inappropriate data types,
   - ✓ Converting one data type to another,
   - ✓ Incorrect ordering of arithmetic operators,
   - ✓ Misuse or omission of parentheses,
   - ✓ Precision loss,
   - ✓ Incorrect use of signs.

2. **Control, Logic and Sequence Defects**

   This type of defects includes incorrect expression of case statements, incorrect iteration of loops, and missing paths.

3. **Typographical Defects**

   These are mainly syntax errors, for example, incorrect spelling of a variable name that are usually detected by a compiler or self-reviews, or peer reviews.

4. **Initialization Defects**

   This type of defects occurs when initialization statements are omitted or are incorrect This may occur because of misunderstandings or lack of communication between programmers, or programmer`s and designer`s, carelessness, or misunderstanding of the programming environment.

5. **Data-Flow Defects**

   It occurs when the code does not follow the necessary data-flow conditions.

6. **Data Defects**

   These are indicated by incorrect implementation of data structures.

7. **Module Interface Defects**

   Module Interface defects occurs because of using incorrect or inconsistent parameter types, an incorrect number of parameters, or improper ordering of the parameters.

8. **Code Documentation Defects**

   When the code documentation does not describe what the program actually does, or is incomplete or ambiguous, it is called a code documentation defect.

9. **External Hardware, Software Interfaces Defects** These defects occur because of
   - ✓ System calls,
   - ✓ Links to databases,
   - ✓ Input/output sequences,
   - ✓ Memory usage,
   - ✓ Resource usage,
   - ✓ Interrupts and exception handling,
   - ✓ Data exchanges with hardware,
   - ✓ Protocols,
   - ✓ Formats,
   - ✓ Interfaces with build files,
   - ✓ Timing sequences.

Testing Defects

- ❖ Test plans, test cases, test harnesses, and test procedures can also contain defects. These defects are called testing defects.
- ❖ Defects in test plans are best detected using review techniques.

1. Test Harness Defects

In order to test software, at the unit and integration levels, auxiliary code must be developed. This is called the test harness or scaffolding code. The test harness code should be carefully designed, implemented, and tested since it is a work product and this code can be reused when new releases of the software are developed.

2. Test Case Design and Test Procedure Defects

These consist of incorrect, incomplete, missing, inappropriate test cases, and test procedures.

1.11 DEFECT EXAMPLES: THE COIN PROBLEM

Specification for the program calculate_coin_value
This program calculates the total rupees value for a set of coins. The user inputs the amount of 25p, 50p and 1rs coins. There are size different denominations of coins. The program outputs the total rupees and paise value of the coins to the user
Input : number_of_coins is an integer
Output: number_of_rupees is an integer, number_of_paise is an integer

- ❖ This is a sample informal specification for a simple program that calculates the total value of a set of coins. The program could be a component of an interactive cash register system. This simple example shows
  - ✓ Requirements/specification defects,
  - ✓ Functional description defects,
  - ✓ Interface description defects.
- ❖ The functional description defects arise because the functional description is ambiguous and incomplete. It does not state that the input and the output should be zero or greater and cannot accept negative values.
- ❖ Because of these ambiguities and specification incompleteness, a checking routine may be omitted from the design.
- ❖ A more formally stated set of preconditions and post conditions is needed with the specification.
- ❖ A precondition is a condition that must be true in order for a software component to operate properly.

Number_of_coins>=0

- ❖ A ***post condition*** is a condition that must be true when a software component completes its operation properly.

Number_of_rupees>=0, Number_of_paise>=0

- The functional description is unclear about the maximum number of coins of each denomination allowed, and the maximum number of rupees and paise allowed as output values.
- It is not clear from the specification how the user interacts with the program to provide input, and how the output is to be reported.

Design Description for the Coin Problem

```
DESIGN DESCRIPTION FOR PROGRAM CALCULATE_COIN_VALUES
PROGRAM CALCULATE_COIN_VALUES
NUMBER_OF_COINS IS INTEGER
TOTAL_COIN_VALUE IS INTEGER
NUMBER_OF_RUPEES IS INTEGER
NUMBER_OF_PAISE IS INTEGER
COIN_VALUES IS ARRAY OF SIX INTEGERS REPRESENTING
EACH COIN VALUE IN PAISE
INITIALIZED TO: 25,25,100
BEGIN
INITIALIZE TOTAL_COIN_VALUE TO ZERO
INITIALIZE LOOP_COUNTER TO ONE
WHILE LOOP_COUNTER IS LESS THAN SIX
BEGIN
OUTPUT "ENTER NUMBER OF COINS" READ
(NUMBER_OF_COINS ) TOTAL_COIN_VALUE =
TOTAL_COIN_VALUE +
NUMBER_OF_COINS * COIN_VALUE[LOOP_COUNTER]
INCREMENT LOOP_COUNTER
END
NUMBER_RUPEES = TOTAL_COIN_VALUE/100
NUMBER_OF_PAISE = TOTAL_COIN_VALUE - 100 * NUMBER_OF_RUPEES
OUTPUT (NUMBER_OF_RUPEES, NUMBER_OF_PAISE)
END
```

Design Defects in the Coin Problem

- **Control, logic, and sequencing defects.** The defect in this subclass arises from an incorrect ―while‖ loop condition (should be less than or equal to six)
- **Algorithmic and processing defects.** These arise from the lack of error checks for incorrect and/or invalid inputs, lack of a path where users can correct erroneous inputs, lack of a path for recovery from input errors.
- **Data defects.** This defect relates to an incorrect value for one of the elements of the integer array, coin_values, which should be 25, 50,100.

❖ **External interface description defects.** These are defects arising from the absence of input messages or prompts that introduce the program to the user and request inputs.

Coding Defects in The Coin Problem

- ❖ **Control, logic, and sequence defects.** These include the loop variable increment step which is out of the scope of the loop. Note that incorrect loop condition (i<6) is carried over from design and should be counted as a design defect.
- ❖ **Algorithmic and processing defects.** The division operator may cause problems if negative values are divided, although this problem could be eliminated with an input check.
- ❖ **Data Flow defects.** The variable total_coin_value is not initialized. It is used before it is defined.
- ❖ **Data Defects.** The error in initializing the array coin_values is carried over from design and should be counted as a design defect.
- ❖ **External Hardware, Software Interface Defects.** The call to the external function scanf is incorrect. The address of the variable must be provided.
- ❖ **Code Documentation Defects.** The documentation that accompanies this code is incomplete and ambiguous. It reflects the deficiencies in the external interface description and other defects that occurred during specification and design.

1.12 DEVELOPER /TESTER SUPPORT OF DEVELOPING A DEFECT REPOSITORY

To increase the effectiveness of their testing and debugging processes, software organizations need to initiate the creation of a defect database, or defect repository.

- ❖ The defect repository supports storage and retrieval of defect data from all projects in a centrally accessible location.
- ❖ A requirement for repository development should be a part of testing and/or debugging policy statements.
- ❖ A defect repository can help in implementing several TMM maturity goals including
  - ✓ Controlling and monitoring of test,
  - ✓ Software quality evaluation and control,
  - ✓ Test measurement,
  - ✓ Test process improvement.

- ❖ Defect monitoring should be done for each on-going project. The distribution of defects will change when changes are made to the process.

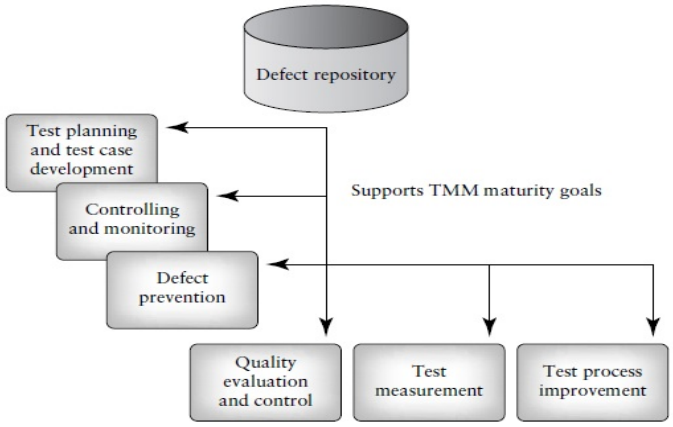


Figure 1.7: The Defect Repository and Support for TMM Maturity Goals

- The defect data is useful for test planning.
- It is a TMM level 2 maturity goal.
- It helps a tester to select applicable testing techniques, design the test cases, and allocate the amount of resources needed to detect and remove defects.
- This allows tester to estimate testing schedules and costs.
- The defect data can support debugging activities also.

QUESTION BANK

PART-A

1. **What are the objectives of software testing? [Apr-May 2016][Nov 2016][Nov 2017]**
   Testing is process of executing a program with the aim of finding the errors.
   A successful test is one which uncovers an undetected error.

2. **Mention the factors that are considered during software development.**
   Software products are
   - ✓ Built on time
   - ✓ Within budget
   - ✓ High quality

3. **Name the different levels in Testing Maturity Model. Also mention the key activity in each of the levels**
   - ✓ Level 1 – Initial
   - ✓ Level 2 – Definition
   - ✓ Level 3 – Integration
   - ✓ Level 4 – Management and measurement
   - ✓ Level 5 – Optimization

4. **List the people who are associated with testing. [Nov 2009]**
   - ✓ Software Tester
   - ✓ Software Developer
   - ✓ Project Lead/Manager
   - ✓ End User

5. **Define Software process. [May 2012]**
   Software process is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system, and its associated artefacts, such as project and test plans, design documents, code, and manuals.

6. **Give the role of process in software quality. [Nov 2009]**
   Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artefacts, such as project and test plans, design documents, code, and manuals.

7. **Define Testing.[May 2016]**
   Testing is generally described as a group of procedures carried out to evaluate some aspects of a piece of software. (or)

   Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes.

   (or)

   Testing is the process of exercising a software component using a selected set of test cases, with the intent of revealing defects, and evaluating quality.

8. What is meant by validation and verification? [May 2012] (or)
Differentiate verification and validation. [Nov 2012, 2016]

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

Validation is the process of evaluating a software system or component during, or at the end of, the development cycle in order to determine whether it satisfies specified requirements

Verification	Validation
Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behaviour.

9. Define Debugging.

Debugging is a process of finding and correcting the errors. The process of debugging is given as

- ✓ Test cases are taken first
- ✓ Each and every test cases are executed
- ✓ Find out the errors
- ✓ Repair the code
- ✓ Retest the code

10. Differentiate the process of testing and debugging.

Testing	Debugging
Finding and locating of a defect	Fixing defect
Done by Testing Team	Done by Development Team

11. Why testing is also said to be dual process?

To find errors

To check the quality attributes of the software like

- ✓ Reliability,
- ✓ Security,
- ✓ Usability,
- ✓ Correctness

12. Define Testing Axioms.

The test axioms are a context neutral set of rules for testing that identify the critical thinking process and motivations for all test approaches.

The axioms represent mechanism designed to meet the objective of the testing in scope. A mechanism may be well defined, documented process, an informal or even ad-hoc activity - but that mechanism must be understood and used by participants in the test.

13. What are the testing axioms?

- ✓ It is impossible to test a program completely
- ✓ Software testing is risk based exercise
- ✓ Testing cannot show that bugs don’t exist
- ✓ The more bugs you find, the more bugs there are

- ✓ Not all the bugs you find will be fixed
- ✓ Product specifications are never fin

14. Define errors.[May 2016]

An error is a mistake misconception misunderstanding on the part of a software developer.
Example: A programmer may act as a software developer. When the programmer is preparing the code, it is possible to type the variable name incorrectly.

15. Define the term defect or fault. [May 2016]

A fault (defect) is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification.

16. Define Failures.

A failure is the inability of a software system or component to perform its required functions within specified performance requirements.

17. Distinguish between fault and failure. [May 2009]

A **fault (defect)** is introduced into the software as the result of an error. It is an anomaly in the software that may cause it to behave incorrectly, and not according to its specification.
A **failure** is the inability of a software system or component to perform its required functions within specified performance requirements

18. Define Test Case.

A test case is a test-related item which contains the following information:

- ✓ **A set of test inputs**
- ✓ **Execution conditions**
- ✓ **Expected outputs**
- ✓ **Define review**

19. Define Test.

A test is a group of related test cases, or a group of related test cases and test procedures.

20. Define Test Oracle.

A test oracle is a document, or piece of software that allows testers to determine whether a test has been passed or failed.

21. Define Test Bed. [Nov 2017]

A test bed is an environment that contains all the hardware and software needed to test a software component or a software system.

22. Define Software Quality. [May 2008]

Quality relates to the degree to which a system, system component, or process meets specified requirements.Quality relates to the degree to which a system, system component, or process meets customer or user needs, or expectations.

23. Define metric.

A metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.

24. Define Quality Metric.

A quality metric is a quantitative measurement of the degree to which an item possesses a given quality attribute. **Examples are:**

- ✓ Correctness
- ✓ Reliability
- ✓ Usability
- ✓ Integrity
- ✓ Portability

25. Define SQA group.

The software quality assurance (SQA) group is a team of people with the necessary training and skills to ensure that all necessary actions are taken during the development process so that the resulting software conforms to established technical requirements.

26. What is called fault localization?

Debugging is also called as fault localization. The debugging process begins after testing has been carried out and the tester has noted that the software is not behaving as specified.
Debugging is the process of:

- ✓ Locating the fault or defect
- ✓ Repairing the code
- ✓ Retesting the code

27. What is test case? What are the information it contains? [Nov 2012]

A test case is a test-related item which contains the following information:

- ✓ **A set of test inputs.** These are data items received from an external source by the code under test. The external source can be hardware, software, or human.
- ✓ **Execution conditions.** These are conditions required for running the test, for example, a certain state of a database, or a configuration of a hardware device.
- ✓ **Expected outputs.** These are the specified results to be produced by the code under test.

28. Differentiate error, defect and failure.

Error is problem, Defect is a mistake, and Failure is a thing which cannot be retrieved.

29. What is the importance of testing principles?

Testing principles are important to test specialists and engineers because they are the foundation for developing testing knowledge and acquiring testing skills. They also provide guidance for defining testing activities.

30. What is the role of software tester? [May 2017]

The tester's job is to

- ✓ Reveal defects
- ✓ Find weak points
- ✓ Inconsistent behaviour
- ✓ Circumstances where the software does not work as expected

31. What is fault or defect model?

A fault (defect) model can be described as a link between the error made, and the fault/defect in the software.

32. What are the uses of hypotheses?

The hypotheses are used to,

- ✓ Design test cases.
- ✓ Design test procedures.
- ✓ Assemble test sets.
- ✓ Select the testing levels suitable for the tests.

33. What are the classifications of defects? List the defect classes. [Nov 2016][May 2017]

The four classes of defects are as follows;

- ✓ Requirements and specifications defects
- ✓ Design defects
- ✓ Code defects
- ✓ Testing defects

34. Give some examples for design defect and testing defect.

- ✓ Algorithmic and Processing Defects
- ✓ Control, Logic, and Sequence Defects
- ✓ Data Defects
- ✓ Module Interface Description Defects
- ✓ Functional Description Defects
- ✓ External Interface Description Defects

35. Define test Harness. [Nov 2012]

In order to test software, at the unit and integration levels, auxiliary code must be developed.This is called the test harness or scaffolding code. The test harness code should be carefully designed, implemented, and tested since it is a work product and this code can be reused when new releases of the software are developed.

36. Describe the basic elements you put in a defect report?

Components od defect report defect id, test case reference, project name, build found, priority, status, assigned to, submitted by, submitted date, brief title, environmental details, description of the issue.

37. What is defect repository?

Defect information is stored in the defect repository.

38. What is pre condition and post condition?

A ***pre condition*** is a condition that must be true in order for a software component to operate properly. *Number_of_coins >= 0*

A ***post condition*** is a condition that must be true when a software component completes its operation properly. *Number_of_rupees >= 0, Number_of_paise >= 0*

39. Mention some goals of TMM that are implemented by using defect repository.

A defect repository can help in implementing several TMM maturity goals including

- ✓ Controlling and monitoring of test,

- ✓ Software quality evaluation and control,
- ✓ Test measurement,
- ✓ Test process improvement.

40. What is the objective of Defect Prevention meeting?

Defect prevention meeting will help not to transfer defects to other phase of the [SDLC] Software Development Life Cycle

PART B

1. **Explain how testing is considered as an engineering activity.[8M]**
2. **Explain the role of process in software quality. [8M]**
3. **Write short notes on testing as a process.**
4. **Define correctness, reliability, integrity, interoperability. [Nov 2012][8M]**
5. **Compare and contrast terms errors, faults and failures using suitable examples.[May 2012]**
6. **Define test cases, test, metric, test oracle, test bed, software quality, software quality assurance group, review. [8M]**
7. **Explain the various software testing principles. [Nov 2012][8M] (or) Explain the principles of software testing and describe the role of tester in software development organization. [Nov 2009][8M] [Nov 2016][May 2017] [Nov 2017][16M]**
8. **Explain tester's role in software development organization. [May 2012] [8M](or) Explain the principles of software testing and describe the role of tester in software development organization. [Nov 2009][Nov 2017] [8M]**
9. **Discuss the origin of defects and explain the defect repository. [Nov 2009][8M] (or) Define defect and write the various origins of defects. [Nov 2012][8M][Nov 2016][May 2017][16M]**
10. **Discuss the defect classification in detail. [Nov 2009] [Nov 2017] [8M]**
11. **Explain the concepts of defects with the coin problem.[Nov 2012][8M]**
12. **Discuss the information in defect repository. [May 2012] [8M](or) Explain the steps in developing defect repository. [Nov 2009][8M]**
13. **Discuss the information about cost of defects**
14. **Explain in detail about testing axioms.**
15. **Discuss about defects prevention in detail.**
16. **Explain Testing Maturity Model in detail.**