**SOFTWARE ENGINEERING-BCSE2355**

**QUESTION BANK**

SECTION -A

1. **What is the prime objective of software engineering?**

Objectives of Software Engineering

1.Maintainability – the ease with which changes in a functional unit can be performed in

order to meet prescribed requirements.

2. Correctness – the extent to which software meets its specified requirements

3. Reuseability – the extent to which a module can be used in multiple applications.

4. Testability – the extent to which software facilitates both the establishment of test criteria

and the evaluation of the software with respect to those criteria.

5. Reliability – an attribute of software quality. The extent to which a program can be expected

to perform its intended function, over an arbitrary time period.

6. Portability – the ease with which software can be transferred from one computer system or

environment to another.

7. Adaptability – the ease with which software allows differing system constraints and user

needs to be satisfied by making changes to the software.

2.**What is SRS?**

A **software requirements specification** (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide .

This document lays a foundation for software engineering activities and is created when entire requirements are elicited and analyzed.

**Advantages**

Software SRS establishes the basic for agreement between the client and the supplier on what the software product will do.

1.    A SRS provides a reference for validation of the final product.

2.    A high-quality SRS is a prerequisite to high-quality software.

3.    A high-quality SRS reduces the development cost.

**Characteristics of an SRS**

1.       Correct

2.       Complete

3.       Unambiguous

4.       Verifiable

5.       Consistent

6.       Ranked for importance and/or stability

7.       Modifiable

8.       Traceable

**3.What do you mean by halstead software science?**

Halstead’s complexity measurement was developed to measure a program module’s complexity directly from source code, with emphasis on computational complexity. The Halstead’s measures are based on four scalar number derived directly from a program’s source code: n1 is number of distinct operators. n2 is number of distinct operands. N1 is total number of distinct operators. N2 is total number of distinct operands.

halstead’s uses certain measures such as program length, program vocabulary, volume, difficulty, and effort for the given algorithm. By this Halstead’s is trying to show that the program length can be calculated, volume of algorithm can be estimated.

**4.What is software maintenance?**

[Software maintenance](https://en.wikipedia.org/wiki/Software_maintenance) is the process of changing, modifying, and updating software to keep up with customer needs. Software maintenance is done after the product has launched for several reasons including improving the software overall, correcting issues or bugs, to boost performance, and more.

Software maintenance is a natural part of SDLC (software development life cycle). Software developers don’t have the luxury of launching a product and letting it run, they constantly need to be on the lookout to both correct and improve their software to remain competitive and relevant.

The four types are:  
Corrective Software Maintenance  
Preventative Software Maintenance   
Perfective Software Maintenance  
Adaptive Software Maintenance

Corrective Software Maintenance

Corrective software maintenance is the typical, classic form of maintenance (for software and anything else for that matter). Corrective software maintenance is necessary when something goes wrong in a piece of software including faults and errors. These can have a widespread impact on the functionality of the software in general and therefore must be addressed as quickly as possible.

Many times, software vendors can address issues that require corrective maintenance due to bug reports that users send in. If a company can recognize and take care of faults before users discover them, this is an added advantage that will make your company seem more reputable and reliable (no one likes an error message after all).

Preventative Software Maintenance

Preventative software maintenance is looking into the future so that your software can keep working as desired for as long as possible. This includes making necessary changes, upgrades, adaptations and more. Preventative software maintenance may address small issues which at the given time may lack significance but may turn into larger problems in the future. These are called latent faults which need to be detected and corrected to make sure that they won’t turn into effective faults.

Perfective Software Maintenance

As with any product on the market, once the software is released to the public, new issues and ideas come to the surface. Users may see the need for new features or requirements that they would like to see in the software to make it the best tool available for their needs. This is when perfective software maintenance comes into play. Perfective software maintenance aims to adjust software by adding new features as necessary and removing features that are irrelevant or not effective in the given software. This process keeps software relevant as the market, and user needs, change.

Adaptive Software Maintenance

Adaptive software maintenance has to do with the changing technologies as well as policies and rules regarding your software. These include operating system changes, cloud storage, hardware, etc. When these changes are performed, your software must adapt in order to properly meet new requirements and continue to run well.

**5.Explain waterfall and spiral model for software life cycle and discuss various activities in each phase.**

The classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.   
The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure: 



Let us now learn about each of these phases in brief detail: 

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.   
   The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification**: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.
   * **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
   * **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.
3. **Design**: The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD)
4. **Coding and Unit testing**: In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.
5. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists of three different kinds of testing activities as described below :

* + **Alpha testing:** Alpha testing is the system testing performed by the development team.
  + **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
  + **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

1. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are basically three types of maintenance :
   * **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   * **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer’s request.
   * **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

**Advantages of Classical Waterfall Model**

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model: 

* This model is very simple and is easy to understand.
* Phases in this model are processed one at a time.
* Each stage in the model is clearly defined.
* This model has very clear and well-understood milestones.
* Process, actions and results are very well documented.
* Reinforces good habits: define-before- design,   
  design-before-code.
* This model works well for smaller projects and projects where requirements are well   
  understood.

**Drawbacks of Classical Waterfall Model**

The classical waterfall model suffers from various shortcomings, basically, we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model: 

* **No feedback path:** In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.
* **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
* **No overlapping of phases:** This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase efficiency and reduce cost, phases may overlap.

**Spiral model** is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **Phase of the software development process.** The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

**The below diagram shows the different phases of the Spiral Model: –**



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

**Advantages of Spiral Model**:   
Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

**Disadvantages of Spiral Model**:   
Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

**5.Mention the values of good SRS. Explain the components of SRS.**

**Following are the characteristics of a good SRS document:**

**Correctness:**

User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

**Completeness:**

Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

**Consistency:**

Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

**Unambiguousness:**

A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

**Ranking for importance and stability:**

There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

**Modifiability:**

SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.

**Verifiability:**

A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement starting that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

**Traceability:**

One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

**Design Independence:**

There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

**Testability:**

A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

**Understandable by the customer:**

An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

**Right level of abstraction:**

If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

#### COMPONENTS OF THE SRS

**Introduction to Components of the SRS**

In previous section, we discussed various characteristics that will help in completely specification the requirements.  Here we describe some of system properties that an SRS should specify. The basic issues, an SRS must address are:

Functional requirements

Performance requirements

Design constraints

External interface requirements

Conceptually, any SRS should have these components. Now we will discuss them one by one.

**1. Functional Requirements**

Functional requirements specify what output should be produced from the given inputs. So they basically describe the connectivity between the input and output of the system. For each functional requirement:

1. A detailed description  of all the  data inputs  and their  sources, the units  of measure, and the range  of valid inputs  be specified:

2. All the  operations  to be  performed on the input data obtain  the output  should be specified, and

3. Care must be taken not to specify any algorithms that are not parts of the system but that may be needed to implement the system.

4. It must clearly state  what the  system should do if  system behaves abnormally when any  invalid input is given  or due  to some  error  during  computation. Specifically, it should specify the behaviour of the system for invalid inputs and invalid outputs.

**2. Performance Requirements (Speed Requirements)**

This part of an SRS specifies the performance constraints on the software system. All the requirements related to the performance characteristics of the system must be clearly specified. Performance requirements are typically expressed as processed transaction s per second or response time from the system for a user event or screen refresh time or a combination of these. It is a good idea to pin down performance requirements for the most used or critical transactions, user events and screens.

**2. Design Constraints**

The client environment may restrict the designer to include some design constraints that must be followed. The various design constraints are standard compliance, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

**Standard Compliance:** It specifies the requirements for the standard the system must follow. The standards may include the report format   and according procedures.

**Hardware Limitations**: The software needs some existing or predetermined hardware to operate, thus imposing restrictions on the design. Hardware limitations can includes the types of machines to be used operating system availability memory space etc.

**Fault Tolerance**:  Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive, so they should be minimized.

**Security:** Currently security requirements have become essential and major for all types of systems. Security requirements place restriction s on the use of certain commands control access to database, provide different kinds of access, requirements for different people, require the use of passwords and cryptography techniques, and maintain a log of activities in the system.

**4. External Interface Requirements**

For each external interface requirements:

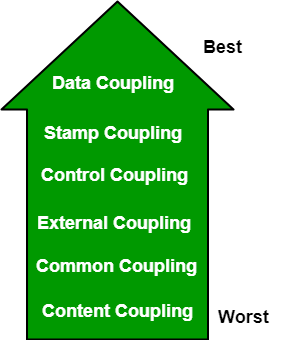
1. All the possible interactions of the software with people hardware and other software should be clearly specified,

2. The characteristics of each user interface of the software product should be specified and

3. The SRS should specify the logical characteristics of each interface between the software product and the hardware components for hardware interfacing.

**6.what is cohension and coupling? Also explain types of coupling and cohension?**

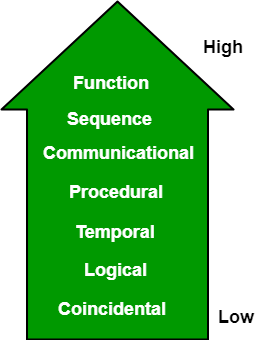
**Coupling:** Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling. 



**Types of Coupling:**

* **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don’t contain tramp data. Example-customer billing system.
* **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
* **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
* **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
* **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
* **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

**Cohesion:** Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



**Types of Cohesion:**

* **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
* **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
* **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
* **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
* **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
* **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
* **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

**7.Define software testing? Define functional and non functional testing. What are the difference between the black box testing and white box testing**.

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

1. Verification: it refers to the set of tasks that ensure that the software correctly implements a specific function.

2. Validation: it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification: “Are we building the product right?”

Validation: “Are we building the right product?”

What are different types of software testing?

Software Testing can be broadly classified into two types:

1. Manual Testing: Manual testing includes testing software manually, i.e., without using any automation tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

2. Automation Testing: Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.

Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money when compared to manual testing.

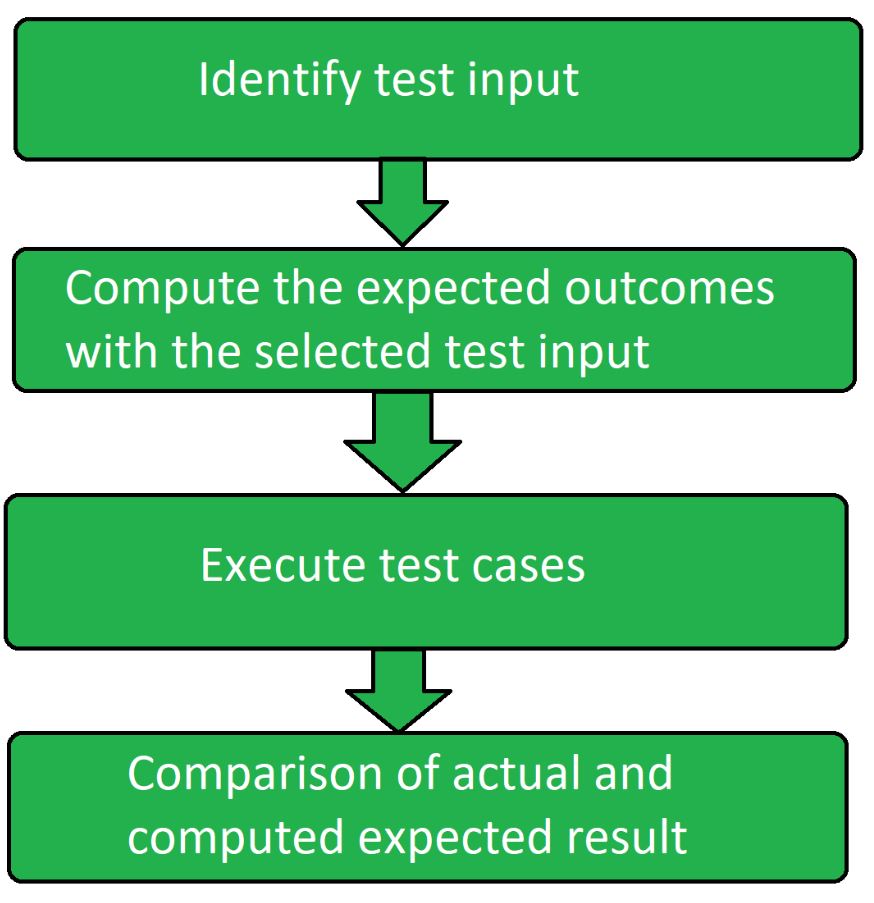
Functional Testing is a type of [Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) in which the system is tested against the functional requirements and specifications. Functional testing ensures that the requirements or specifications are properly satisfied by the application. This type of testing is particularly concerned with the result of processing. It focuses on simulation of actual system usage but does not develop any system structure assumptions.

It is basically defined as a type of testing which verifies that each function of the software application works in conformance with the requirement and specification. This testing is not concerned about the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output and comparing the actual output with the expected output. This testing focuses on checking of user interface, APIs, database, security, client or server application and functionality of the Application Under Test.

Functional testing can be manual or automated.

**Functional Testing Process:**  
Functional testing involves the following steps:

1. Identify function that is to be performed.
2. Create input data based on the specifications of function.
3. Determine the output based on the specifications of function.
4. Execute the test case.
5. Compare the actual and expected output.



**Major Functional Testing Techniques:**

* Unit Testing
* Integration Testing
* Smoke Testing
* User Acceptance Testing
* Interface Testing
* Usability Testing
* System Testing
* Regression Testing

**Functional Testing Tools:**

**1.** Selenium

**2.** QTP

**3.** JUnit

**4.** SoapUI

**5.** Watir

**Advantages of Functional Testing:**

* It ensures to deliver a bug-free product.
* It ensures to deliver a high-quality product.
* No assumptions about the structure of the system.
* This testing is focused on the specifications as per the customer usage.

**Disadvantages of Functional Testing:**

* There are high chances of performing redundant testing.
* Logical errors can be missed out in the product.
* If the requirement is not complete then performing this testing becomes difficult.

**Non-functional Testing** is a type of [Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) that is performed to verify the non-functional requirements of the application. It verifies whether the behavior of the system is as per the requirement or not. It tests all the aspects which are not tested in functional testing. Non-Functional testing is a software testing technique that checks the non-functional attributes of the system. Non-functional testing is defined as a type of software testing to check non-functional aspects of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing. Non-functional testing is as important as functional testing.

## **Objectives of Non-functional Testing**

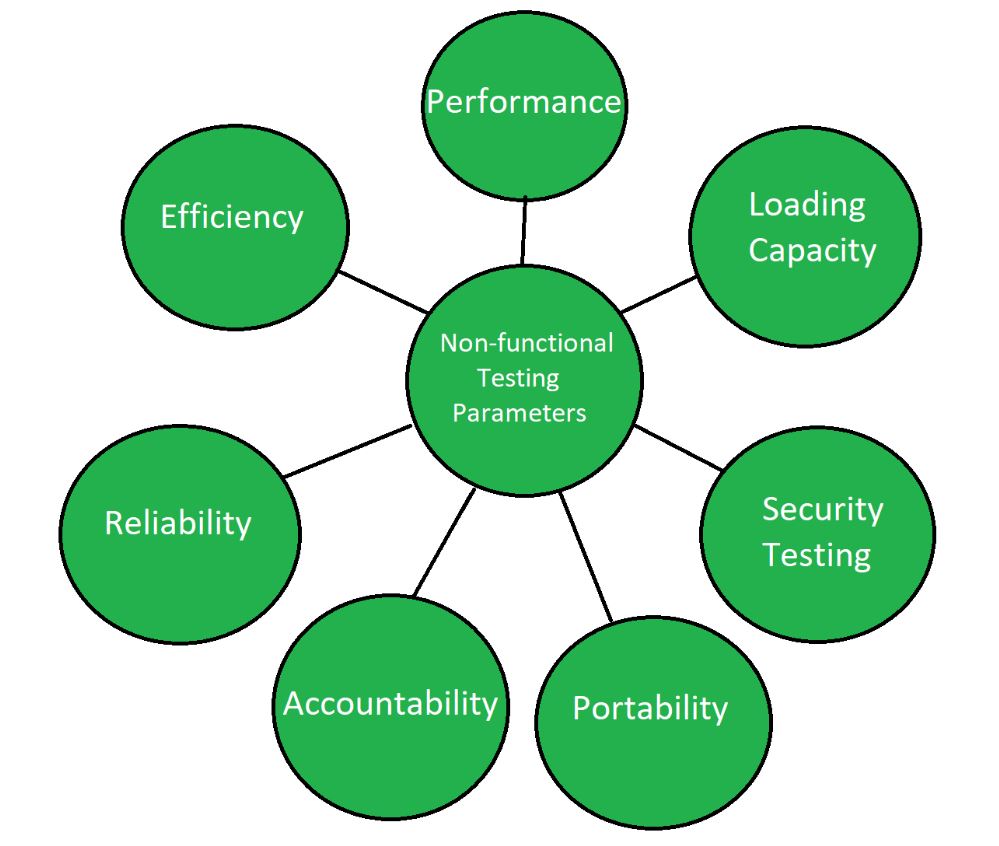
The objective of non-functional testing is:

* To increase usability, efficiency, maintainability and portability of the product.
* To help in the reduction of production risk related with non-functional aspects of the product.
* To help in the reduction of cost related with non-functional aspects of the product.
* To optimize the installation, execution and monitoring way of the product.
* To collect and produce measurements and metrics for internal research and development.
* To improve and enhance knowledge of the product behavior and technologies in use.

## **Non-Functional Testing Techniques**

* **Compatibility testing:**A type of testing to ensure that a software program or system is compatible with other software programs or systems.
* **Compliance testing:**A type of testing to ensure that a software program or system meets a specific compliance standard, such as HIPAA or Sarbanes-Oxley.
* **Endurance testing:**A type of testing to ensure that a software program or system can handle a long-term, continuous load.
* **Load testing:**A type of testing to ensure that a software program or system can handle a large number of users or transactions.
* **Performance testing:**A type of testing to ensure that a software program or system meets specific performance goals, such as response time or throughput.
* **Recovery testing:**A type of testing to ensure that a software program or system can be recovered from a failure or data loss.
* **Security testing:**A type of testing to ensure that a software program or system is secure from unauthorized access or attack.
* **Scalability testing:**A type of testing to ensure that a software program or system can be scaled up or down to meet changing needs.
* **Stress testing:**A type of testing to ensure that a software program or system can handle an unusually high load.
* **Usability testing:**A type of testing to ensure that a software program or system is easy to use.
* **Volume testing:**A type of testing to ensure that a software program or system can handle a large volume of data.

## **Non-functional Testing Parameters**

**:** 

| S. No. | Black Box Testing | White Box Testing |
| --- | --- | --- |
| 1. | It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it. | It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software. |
| 2. | Implementation of code is not needed for black box testing. | Code implementation is necessary for white box testing. |
| 3. | It is mostly done by software testers. | It is mostly done by software developers. |
| 4. | No knowledge of implementation is needed. | Knowledge of implementation is required. |
| 5. | It can be referred to as outer or external software testing. | It is the inner or the internal software testing. |
| 6. | It is a functional test of the software. | It is a structural test of the software. |
| 7. | This testing can be initiated based on the requirement specifications document. | This type of testing of software is started after a detail design document. |
| 8. | No knowledge of programming is required. | It is mandatory to have knowledge of programming. |
| 9. | It is the behavior testing of the software. | It is the logic testing of the software. |
| 10. | It is applicable to the higher levels of testing of software. | It is generally applicable to the lower levels of software testing. |
| 11. | It is also called closed testing. | It is also called as clear box testing. |
| 12. | It is least time consuming. | It is most time consuming. |
| 13. | It is not suitable or preferred for algorithm testing. | It is suitable for algorithm testing. |
| 14. | Can be done by trial and error ways and methods. | Data domains along with inner or internal boundaries can be better tested. |
| 15. | **Example:** Search something on google by using keywords | **Example:** By input to check and verify loops |
| 16. | **Black-box test design techniques-**   * Decision table testing * All-pairs testing * Equivalence partitioning * Error guessing | **White-box test design techniques-**   * Control flow testing * Data flow testing * Branch testing |
| 17. | **Types of Black Box Testing:**   * Functional Testing * Non-functional testing * Regression Testing | **Types of White Box Testing:**   * Path Testing * Loop Testing * Condition testing |
| 18. | It is less exhaustive as compared to white box testing. | It is comparatively more exhaustive than black box testing. |

**7.What are the difference between verification and validation in software development**

| Verification | Validation |
| --- | --- |
| It includes checking documents, design, codes and programs. | It includes testing and validating the actual product. |
| Verification is the static testing. | Validation is the dynamic testing. |
| It does *not* include the execution of the code. | It includes the execution of the code. |
| Methods used in verification are reviews, walkthroughs, inspections and desk-checking. | Methods used in validation are Black Box Testing, White Box Testing and non-functional testing. |
| It checks whether the software conforms to specifications or not. | It checks whether the software meets the requirements and expectations of a customer or not. |
| It can find the bugs in the early stage of the development. | It can only find the bugs that could not be found by the verification process. |
| The goal of verification is application and software architecture and specification. | The goal of validation is an actual product. |
| Quality assurance team does verification. | Validation is executed on software code with the help of testing team. |
| It comes before validation. | It comes after verification. |
| It consists of checking of documents/files and is performed by human. | It consists of execution of program and is performed by computer. |

**8.Define Software crisis? What are the causes of software crisis?**

**Software Crisis** is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to using the same workforce, same methods, same tools even though rapidly increasing in software demand, the complexity of software, and software challenges. With the increase in the complexity of software, many software problems arise because existing methods were insufficient. If we will use the same workforce, same methods, and same tools after the fast increase in software demand, software complexity, and software challenges, then there arise some problems like software budget problems, software efficiency problems, software quality problems, software managing and delivering problem, etc. This condition is called a **software crisis.**

### Causes of Software Crisis:

* The cost of owning and maintaining software was as expensive as developing the software
* At that time Projects were running over-time
* At that time Software was very inefficient
* The quality of the software was low quality
* Software often did not meet user requirements
* The average software project overshoots its schedule by half
* At that time Software was never delivered
* Non-optimal resource utilization.
* Difficult to alter, debug, and enhance.
* The software complexity is harder to change.

Let’s now understand **which factors are contributing to the software crisis.**

* Poor project management.
* Lack of adequate training in software engineering.
* Less skilled project members.
* Low productivity improvements.

### Solution of Software Crisis:

There is no single solution to the crisis. One possible solution to a software crisis is *Software Engineering* because software engineering is a systematic, disciplined, and quantifiable approach. For preventing software crises, there are some guidelines:

* Reduction in software over budget.
* The quality of software must be high.
* Less time is needed for a software project.
* Experienced and skilled people working over the software project.
* Software must be delivered.
* Software must meet user requirements.

**9. What is COCOMO Model? What are the types of COCOMO model?**

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models. The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.

Schedule: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of the constant to be used in subsequent calculations.

**1.Intermediate Model** – The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software systems. However, in reality, no system’s effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation. Classification of Cost Drivers and their attributes: (i) Product attributes –

* Required software reliability extent
* Size of the application database
* The complexity of the product
* Run-time performance constraints
* Memory constraints
* The volatility of the virtual machine environment
* Required turnabout time
* Analyst capabilitySoftware engineering capability
* Applications experience
* Virtual machine experience
* Programming language experience
* Use of software tools
* Application of software engineering methods
* Required development schedule

**2.Detailed Model** – Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver’s impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort. The Six phases of detailed COCOMO are:

Planning and requirements

System design

Detailed design

Module code and test

Integration and test

Cost Constructive model

**9. Differentiates between coupling & cohesion.**

|  |  |
| --- | --- |
| Cohesion | Coupling |
| Cohesion is the concept of intra-module. | Coupling is the concept of inter-module. |
| Cohesion represents the relationship within a module. | Coupling represents the relationships between modules. |
| Increasing cohesion is good for software. | Increasing coupling is avoided for software. |
| Cohesion represents the functional strength of modules. | Coupling represents the independence among modules. |
| Highly cohesive gives the best software. | Whereas loosely coupling gives the best software. |
| In cohesion, the module focuses on a single thing. | In coupling, modules are connected to the other modules. |
| Cohesion is created between the same module. | Coupling is created between  two different modules. |
| There are Six types of Cohesion  1. Functional Cohesion. 2. Procedural Cohesion.  3. Temporal Cohesion. 4. Sequential Cohesion. 5. Layer Cohesion. 6. Communication Cohesion. | There are Six types of Coupling  1. Common Coupling. 2. External Coupling.  3. Control Coupling. 4. Stamp Coupling. 5. Data Coupling 6. Content Coupling. |

**SECTION-B**

**1.Why do we need software development life cycle model?**

The Software Development Life Cycle (SDLC) is a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time. The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands.

**2.What are the characteristics of a software.**

The quality of a software product is determined by what it offers and how easily it can be used. Software is judged by different people on different grounds. Customers, for instance, want software that meets their specific needs. Similarly, developers engaged in designing, coding, and maintaining the software determine the quality of the software by assessing its internal characteristics. Let’s check them out…

**Functionality**

The functionality of software refers to its ability to perform and function according to design specification. In simple terms, software systems should function correctly, i.e. perform all the functions for which they are designed.

The functions refer to the features that the end user as well as the business expect as basic facilities from the system. All these functions must be integrated into the system. Many software applications out there are designed for simplicity, but ultimately, the purpose of software is to provide its users with the desired functionality.  In order to look like the best software product, it must have a clear appearance, components, and functions. However, there are also those products out there that can provide a great deal of value for your money.

**Usability (User-friendly)**

The user-friendliness of the software is characterized by its ease of use. In other words, learning how to use the software should require less effort or time. Navigating the software is extremely important since it helps determine the journey the user takes within the software.  This is imperative to ensuring visitors remain on your website and have a positive experience, which leads to an increase in sales and brand loyalty.

An important indicator of a good piece of software is its user interface, i.e., the smooth flow of its design. A product with a great UI (User Interface) design is more likely to get noticed than one without. If a software program isn’t user-friendly, users may have trouble navigating the software and using some of its features. Software should require less time or effort to learn. Ideally, a software should be easy to use even by people with no IT experience.

### **Efficiency**

Essentially, it refers to the software’s ability to utilize human and system resources such as time, effort, CPU, memory, computation power, network bandwidth, files, databases, etc., as effectively and efficiently as possible. For a software project to succeed, efficiency is crucial. In addition to meeting the needs for which the software was made, it must also provide excellent features designed to assist users in completing their tasks faster. Software should make efficient use of storage space and execute commands according to timing requirements.

In order to be efficient, a software must offer users proper value in terms of their time and cash. The market is filled with products that cater to various industries, but only a handful of products are efficient enough to benefit individuals and businesses. The medical billing software that Open Practice Solutions provides, for instance, makes billing processes much more efficient for clients than those offered by other companies.

### **Flexibility**

Software Flexibility refers to the ability of the software solution to adapt to potential or future changes in its requirements. When evaluating the flexibility of software, look at how simple it is to add, modify, or remove features without interfering with the current operation.

It is essential to keep up with rapidly changing markets, technologies, and customer needs. In software development, change is inevitable; it can arise during the development process itself or as the result of future requirements. Flexibility is therefore highly valued. Consequently, any software product must be scalable, flexible, and easily adaptable to future technology. When designing or building a software product, be sure to plan for these changes that are inevitably going to occur. Loose coupling of components is the key to creating highly flexible systems.

### **Reliability**

The reliability of a software product describes the likelihood it will operate without failure over a specified period of time under certain conditions. It determines the ability of software to maintain its level of performance (provide desired functionality) under specified conditions for a specified period of time. Generally speaking, software reliability is measured as the availability of the software. The value should not be less than 99%. In reliability testing, the goal is not perfection, but achieving a level of reliability that is acceptable before a software product is released to customers. MTTF, MTTR, MTBR, etc., are some reliability metrics that can be used to quantify the reliability of a software product.

It is regarded as one of the most important quality aspects of software quality, along with functionality, efficiency, maintainability, etc. Since software tends to be complex, it is hard to achieve software reliability.

### **Maintainability**

Maintainability refers to how easily you can repair, improve and comprehend software code. In some ways, maintaining is similar to being flexible. Maintainability deals with the modification of errors and minor alterations to software code, while flexibility focuses on major functional extensions. It also involves maintaining the services and functionality of the software.

Most of the time, developers are not the ones who maintain the software. Therefore, good documentation is crucial, which includes code documentation, interface definitions, etc. Maintainability of software products is affected by the quality of the documentation. Typically, more than half of development budgets are spent on software maintenance. Maintenance should therefore be integrated into the development lifecycle for effective software maintenance.

### **Portability**

Software portability is a critical factor that cannot be ignored. Portability refers to the ability to use software in different environments. This is the ease with which software can be ported from one platform to another without (or with minimal) changes, while obtaining similar results. As simple as it may sound, it refers to the ability of software to work on different hardware platforms without any (or little) modifications needed.

Furthermore, you should be aware that porting software to a new environment is comparatively cheaper than developing an equivalent application from scratch. There can be no doubt that portability is a crucial aspect of reducing development costs.

**Integrity**

There are multiple interpretations of software integrity. Some people tend to associate integrity with security, believing it is resistant to hacks and privacy violations. To others, high integrity means that the software cannot be modified without authorization.

Integrity is key for demonstrating the safety, security, and maintainability of your software. In addition, software that needs to be compliant with industry regulations and coding standards requires high code integrity. Achieving software integrity can be difficult. Yet, with the right practices to improve safety, security, and maintainability, the challenge can be easily overcome. In these days of increased security threats, all software must include this factor.

**2.Define error and bug.**

#### ****ERROR:****

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

#### ****BUG:****

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

**3.What are the limitations of testing?**

Software Testing Limitations Limitation is a principle that restricts the extent of any application. Software testing has also few limitations that should be considered to set realistic expectations about its benefits. Inspite of being most widely used verification technique, software testing as various following limitations:

1) Testing can be used to show the presence of errors, but never to show their absence [5]. It can only identify the known issues or errors. It gives no idea about defects still uncovered. Testing cannot guarantee that the system under test is error free.

2) Testing provides no help when we have to make a decision to either "release the product with errors for meeting the deadline" or to "release the product late compromising the deadline"

3) Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions .

4) Software testing does not help in finding root causes which resulted in injection of defects in the first place. Locating root causes of failures can help us in preventing injection of such faults in future.

**4.What are the objectives of Software Testing?**

Objectives of testing

• Executing a program with the intent of finding an error.

• To check if the system meets the requirements and be executed successfully in the Intended environment.

• To check if the system is “Fit for purpose”

• To check if the system does what it is expected to do.

**6.List the different types of testing fall under functional testing.?**

### Functional testing types

Let’s explore these types of functional tests with examples:

* Unit testing
* Component testing
* Smoke testing
* Sanity testing
* Regression testing
* Integration testing
* API testing
* UI testing
* System testing
* White-box testing
* Black-box testing
* Acceptance testing
* Alpha testing
* Beta testing
* Production testing

**Unit testing.**Before you can test an entire software program, make sure the individual parts work properly on their own. Unit testing validates the function of a unit, ensuring that the inputs (one to a few) result in the lone desired output. This testing type provides the foundation for more complex integrated software. When done right, unit testing drives higher quality application code and speeds up the development process. Developers often execute unit tests through test automation.

Unit testing example: A developer builds a calculator app. A unit test would check whether the user can input two numbers and receive an accurate sum. Separate unit tests would validate other calculator functionality, such as subtraction, multiplication and division.

**Component testing.**Also called module testing, component testing checks individual parts of an application. Similar to unit testing, component testing assesses a part of the software in isolation from the broader system. The difference between unit testing and component testing is that the former is done by developers in a white-box format to verify that program modules execute, while the latter is done by testers in a black-box format to validate individual objects or parts of the software. If other software components rely on the component under test, the QA professional might use a stub and driver to simulate interactions between those dependent components.

Component testing example: A [banking mobile app](https://www.applause.com/finance) includes an option to schedule an appointment with a banking professional. The stub provides a simulated user profile, and the driver provides a simulated schedule of available appointment times. In this functional testing example, the middle component — the one under test — finds the user’s location via GPS and displays local banking centers from which they can choose. By testing this component in isolation, the tester can ensure that the geolocation service works correctly and displays an accurate list of nearby locations.

**Smoke testing.**Smoke testing, a type of acceptance testing, provides an initial check that a new software build and its critical functionality are stable. If the smoke tests pass, the build can undergo further testing. Smoke testing, also called build verification testing, often checks whether new or critical functionality meets its objective. If the tests don’t pass, as the saying goes, “where there’s smoke, there’s fire,” and additional dev work is required.

Smoke testing example: A web app for an [insurance company](https://go.applause.com/exceptional-insurance-customer-experiences-3-keys.html?mc=null&ls=null&lc=null&cc=null&orig_mc=null&orig_ls=null&orig_lc=null&orig_cc=null&utm_campaign=null&utm_medium=null&utm_source=null&orig_utm_campaign=null&orig_utm_medium=null&orig_utm_source=null&gclid=null) adds a claims status page. Testers would apply smoke tests to verify that the existing build works on a fundamental level, such as whether a user can successfully log in, navigate to the claims status page and retrieve the status of a specific claim without the app crashing or malfunctioning.

**Sanity testing.**A type of regression testing, QA professionals perform sanity testing on new versions of stable builds to validate either new functionality or bug fixes. While similar to smoke testing in that both provide a gate check that a build is ready for more testing, sanity testing is unscripted and specifically targets the area that has undergone a code change.

Sanity testing example: A web page for a [telehealth provider](https://www.applause.com/healthcare) returns a 404 error for its mental health page. The developers fix the issue, then commit the build for testing. The QA professional performs a sanity check to determine whether the basic functionality and navigation for that specific page work as intended.

**Regression testing.**Just because functional tests pass once doesn’t mean they’ll always pass. When developers commit new code or change a feature, you run regression tests to make sure the software still functions as expected. Regression testing helps maintain a stable product while changes are made to it. Regression tests are often automated.

Regression testing example: A [clothing retailer](https://www.applause.com/retail) adds the ability to pay with customer rewards points on their mobile app. Testers might perform regression tests on other existing functionality, such as the ability to pay with credit cards and gift cards, to make sure all forms of payment work correctly.

**Integration testing.**Integration testing is often done in concert with unit testing. Through integration testing, QA professionals verify that individual modules of code work together properly as a group. Many modern applications run on microservices, self-contained applications that are designed to handle a specific task. These microservices must be able to communicate with each other, or the application won’t work as intended. Through integration testing, testers ensure these components operate and communicate together seamlessly.

Integration testing example: A credit card company includes a page where a customer can request a credit increase, which is a separate code base from login functionality. Testers might perform integration tests to make sure the system remembers the user after they navigate to the credit increase page, and again after a successful request.

**API testing.**Application programming interfaces connect different applications or systems, and they are growing in popularity as consumers expect apps to interoperate. With API testing, testers validate that API connections and responses function as intended, including how they handle data and user permissions.

API testing example: A [travel booking site](https://www.applause.com/travel) might pull pricing data from an airline company’s database via APIs. Through API testing, QA professionals can verify that the correct data type is returned in the local currency and responsive to changes in date and location.

**UI testing.**With UI testing, QA professionals interact with the graphical interface of a software program. This includes testing of UI controls like buttons, menus and text input to ensure that the experience flow and features chosen are optimal for the user experience.

UI testing example: A [wearables maker](https://go.applause.com/why-iot-devices-require-end-to-end-testing.html?mc=null&ls=null&lc=null&cc=null&orig_mc=null&orig_ls=null&orig_lc=null&orig_cc=null&utm_campaign=null&utm_medium=null&utm_source=null&orig_utm_campaign=null&orig_utm_medium=null&orig_utm_source=null&gclid=null) creates a mobile app for product setup and maintenance. As part of UI testing, the team would make sure that required fields function as expected, images display correctly and maintenance information appears in the app dashboard after use.

**System testing.**With system testing, QA professionals test the software in its entirety, as a complete product. With this type of functional testing, testers validate the complete and integrated software package to make sure it meets requirements. Where necessary, testers can provide feedback on the functionality and performance of the app or website without prior knowledge of how it was programmed. This helps teams develop test cases to be used moving forward. System testing is also referred to as end-to-end testing.

System testing example: An [automobile manufacturer](https://www.applause.com/automotive-testing) produces an in-car entertainment system that gives users functionality for voice control, GPS, a video player, Bluetooth connectivity, mobile phone pairing, touch-screen support and climate control. Testers would assess all of these features individually, but they must also test them as a complete system to ensure interoperability and a good user experience.

**White-box testing.**When the software’s internal infrastructure, code and design are visible to the developer or tester, that refers to white-box testing. This approach incorporates various functional testing types, including unit, integration and system testing. In a white-box testing approach, the organization tests several aspects of the software, such as predefined inputs and expected outputs, as well as decision branches, loops and statements in the code.

White box testing example: In this functional testing example, consider an end-to-end test for a customer who [adds payment information](https://go.applause.com/4-challenges-digital-payments-are-creating-and-how-to-address-them.html?mc=null&ls=null&lc=null&cc=null&orig_mc=null&orig_ls=null&orig_lc=null&orig_cc=null&utm_campaign=null&utm_medium=null&utm_source=null&orig_utm_campaign=null&orig_utm_medium=null&orig_utm_source=null&gclid=null) to a retailer’s app. Developers and testers would conduct tests in a white-box format to ensure that sensitive data, such as a credit card number, is stored in a PCI-compliant manner. White-box tests might also ensure that purchase information flows to a machine learning algorithm to generate predictions, the purchase correctly generates rewards points, and the inventory system deducts the items from the stock count.

**Black-box testing.**Contrary to white-box testing, black-box testing involves testing against a system where the internal code, paths and infrastructure are not visible. Thus, testers use this method to validate expected outputs against specific inputs. Any time where a QA professional doesn’t look into the code before testing can be considered black box. With black-box testing, the organization can test the software in the same way a customer would experience it. Black-box testing encompasses a variety of non-functional and functional testing types, depending on the objective of the test.

Black box testing example: On a [streaming television platform](https://www.applause.com/media), the tester toggles the search functionality and executes a search for a specific actor. The tester then verifies that the search feature returns logical (expected) outputs, including television shows that the actor appeared in, or suggested titles similar to that actor’s well-known works.

**Acceptance testing.**The purpose of acceptance testing is purely to ensure that the end user can achieve the goals set in the business requirements. Rather than focus on functionality of specific features, acceptance testing involves reviewing the feature-complete application flow and end-to-end experience. User acceptance testing (UAT) and beta testing, subsets of acceptance testing, involve end users to conduct their analysis of the finished product. From there, the organization can evaluate that feedback and make changes.

Acceptance testing example: A software company releases a product that enables its users to manage big data. Upon release of a new version of the software, a group of that company’s most significant users conducts user acceptance testing to determine whether the new version meets their primary needs and how the product can be improved.

**Alpha testing.**Another subset of acceptance testing, alpha testing uses internal team members to evaluate the product. These team members should be knowledgeable of the project but not directly involved in its development or testing. Where some builds might still be somewhat unstable, alpha testing provides an immediate subset of testers to root out major bugs before the software is seen by external users.

Alpha testing example: In this functional testing example, a casino games provider releases a new version of its app that includes video poker. The organization compiles a cross-functional group of internal users that test whether the app functions correctly on their devices and how the user experience can improve.

**Beta testing.**After the internal team tests the product and fixes bugs, beta testing occurs with a select group of end users. Beta testing serves as a soft launch, enabling you to get feedback from real users who have no prior knowledge of the app. Beta testing enables you to gather feedback from unbiased users who may interact with the product differently than you intended, perhaps identifying critical unknown bugs before release to a wide user base.

Beta testing example: A [restaurant chain](https://go.applause.com/5-things-customers-demand-from-qsr-apps-website.html?mc=null&ls=null&lc=null&cc=null&orig_mc=null&orig_ls=null&orig_lc=null&orig_cc=null&utm_campaign=null&utm_medium=null&utm_source=null&orig_utm_campaign=null&orig_utm_medium=null&orig_utm_source=null&gclid=null) releases a new mobile order and pickup system. Before the company releases the functionality to all of its mobile app users, it tests the app with a small number of dedicated customers and provides them with rewards for participating.

**Production testing.**Once the product goes public, it is in a live production environment where any user can interact with it in any way — you no longer can control everything from the testing environment to the number of people using the product. Production testing is part of continuous testing and shift-right testing, which attempts to discover and triage user-reported defects as quickly as possible. By testing in production, the organization can test beyond the scripted test cases in a varied environment. With production testing, the organization can confirm product functionality and stability.

Production testing example: A fitness equipment manufacturer can monitor user-reported defects and device metrics to make sure its internet-connected treadmills, elliptical and stair-climbing machines function as they should — upon delivery and continuously.

**7.What do you mean by Software maintenance**

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Software maintenance is also an important part of the Software Development Life Cycle(SDLC). To update the software application and do all modifications in software application so as to improve performance is the main focus of software maintenance. Software is a model that run on the basis of real world. so, whenever any change requires in the software that means the need of real world changes wherever possible.

**Need for Maintenance –**   
Software Maintenance must be performed in order to:

* Correct faults.
* Improve the design.
* Implement enhancements.
* Interface with other systems.
* Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
* Migrate legacy software.
* Retire software.
* Requirement of user changes.
* Run the code fast

**Challenges in Software Maintenance:**

The various challenges in software maintenance are given below:

* The popular age of any software program is taken into consideration up to ten to fifteen years. As software program renovation is open ended and might maintain for decades making it very expensive.
* Older software program’s, which had been intended to paintings on sluggish machines with much less reminiscence and garage ability can not maintain themselves tough in opposition to newly coming more advantageous software program on contemporary-day hardware.
* Changes are frequently left undocumented which can also additionally reason greater conflicts in future.
* As era advances, it turns into high priced to preserve vintage software program.
* Often adjustments made can without problems harm the authentic shape of the software program, making it difficult for any next adjustments.
* There is lack of Code Comments.

**Categories of Software Maintenance –**   
Maintenance can be divided into the following: 

1. **Corrective maintenance:**   
   Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.
2. **Adaptive maintenance:**   
   This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.
3. **Perfective maintenance:**   
   A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.
4. **Preventive maintenance:**   
   This type of maintenance includes modifications and updations to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

**8.Relate Quality assurance with Quality control. Justify your answer that QA is not QC.**

**9.Define software Quality.**

# **Software Quality**

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

**Example:** Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

**The modern view of a quality associated with a software product several quality methods such as the following:**

**Portability:** A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

**Usability:** A software product has better usability if various categories of users can easily invoke the functions of the product.

**Reusability:** A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

**Correctness:** A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

**Maintainability:** A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

**10.Define Quality Standards.**

Quality standards are defined as documents that provide requirements, specifications, guidelines, or characteristics that can be used consistently to ensure that materials, products, processes, and services are fit for their purpose.

Standards provide organizations with the shared vision, understanding, procedures, and vocabulary needed to meet the expectations of their stakeholders. Because standards present precise descriptions and terminology, they offer an objective and authoritative basis for organizations and consumers around the world to communicate and conduct business.

**[](https://asq.org/quality-progress/articles/under-the-wire?id=af58f42dc217470b9ee6c38a0d2c042d)  
Principles of Quality Standards**

## **WHO USES QUALITY STANDARDS?**

Organizations turn to standards for guidelines, definitions, and procedures that help them achieve objectives such as:

* Satisfying their customers’ quality requirements
* Ensuring their products and services are safe
* Complying with regulations
* Meeting environmental objectives
* Protecting products against climatic or other adverse conditions
* Ensuring that internal processes are defined and controlled

## **WHY ARE STANDARDS IMPORTANT?**

**For businesses:** Standards are important to the bottom line of every organization. Successful companies recognize standards as business tools that should be managed alongside quality, safety, intellectual property, and environmental policies. Standardization leads to lower costs by reducing redundancy, minimizing errors or [recalls](https://asq.org/quality-resources/recalls), and reducing time to market.

**For the global economy:** Businesses and organizations complying to quality standards helps products, services, and personnel cross borders and also ensures that products manufactured in one country can be sold and used in another.

**For consumers:** Many quality management standards provide safeguards for users of products and services, but standardization can also make consumers’ lives simpler. A product or service based on an international standard will be compatible with more products or services worldwide, which increases the number of choices available across the globe.

**11.Illustrate the need of quality standards**

Quality standards are designed to ensure companies meet the minimum requirements to become an integral part of almost every industry from food to automotive to healthcare. It’s clear that those standards are here indefinitely.

Some organizations struggle with the concepts of quality standards or they view it as a complex system. But with a little bit of information, help and a clear understanding of why quality standards can benefit them, organizations can learn to embrace quality standards rather than buck against them.

## **The Intent of Quality Standards**

Quality isn’t just about profits and loss or beating out a competitor. It’s about safety, delivering on a promise and meeting the very basics of customer expectations. But, by meeting quality standards, companies often reap better profits and reduce losses. Those that exceed quality standards stand out above their competitors and further their potential for profit and consumer loyalty. Therein rests the benefit of quality standards.

## **Quality Standards Offer a Formula for Success**

Quality can be an obscure concept at first because what one might see as quality someone else may not. Hence, the need and purpose of quality standards is crucial. ISO standards, such as ISO 9001, ISO 14001, and ISO 27001, serve as a framework for businesses. Clearly defined standards and requirements make it easier for companies to meet what their consumers consider “quality” and they improve the overall vision of what the company should work toward.

But what good are quality standards if your company isn’t clear on how they apply to your specific product or service? Quality standards are only meant to serve as a framework. There are still walls to be built, fixtures to be mounted, and a floor to be laid. This is where quality management systems come into play.

Still struggling with quality standards? Partner with QAD and embrace quality standards.

## **The Role of Quality Management Systems**

Not to be confused with quality standards, a [quality management system (QMS)](https://www.qad.com/blog/2018/03/beginners-guide-to-quality-management-systems) is a tool used by companies wishing to effectively meet (and possibly even exceed) required quality standards. Designed to prevent losses and improve compliance, QMS offers a method by which companies measure and monitor products, services, processes, procedures, production, and employees to ensure they are meeting what’s expected.

The most commonly used QMS compliance practices include:

* Total Quality Management (TQM)
* Continuous Quality Management (CQI)
* Six Sigma

**12.Explain spiral Model.**

**Spiral model** is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **Phase of the software development process.** The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

**The below diagram shows the different phases of the Spiral Model: –**



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

**Advantages of Spiral Model**:   
Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

**Disadvantages of Spiral Model**:   
Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

**13.Which is more important – the product or process? Justify your answer.**

**14.Discuss the various methods of software gathering.**

  In most cases, stakeholders are not aware of all the alternatives that exist when it comes to developing a specific product. Immersed in the current status quo, their vision tends to be limited. It’s hard for users to detach from the way they’re currently doing things—to imagine possibilities that are a significant departure from what they have now.

Furthermore, there’s no silver bullet. No single requirement gathering technique will help you elicit a complete set of requirements that will fill every gap and stand up to scrutiny during validation of custom software development.

## **WHAT IS REQUIREMENTS GATHERING?**

Requirements gathering is a crucial part of any custom software development project, large or small. It is essential to understanding and fulfilling the needs of the customers.

The process of requirement gathering for your software solution include identifying and documenting the necessary requirements of customers, users, stakeholders etc. related to the project. This knowledge will be used to develop solutions in the form of products, services, software etc.

Methods used to gather this data for your software solution may include techniques such as interviewing, brainstorming, focus groups, questionnaires etc.

### **DIFFERENT TYPES OF REQUIREMENTS**

The main types of requirements are:

·       Functional Requirements

·       Performance Requirements

·       System Technical Requirements

·       Specifications

### **STEPS OF REQUIREMENT GATHERING**

Step 1: Understand Pain Behind the Requirement

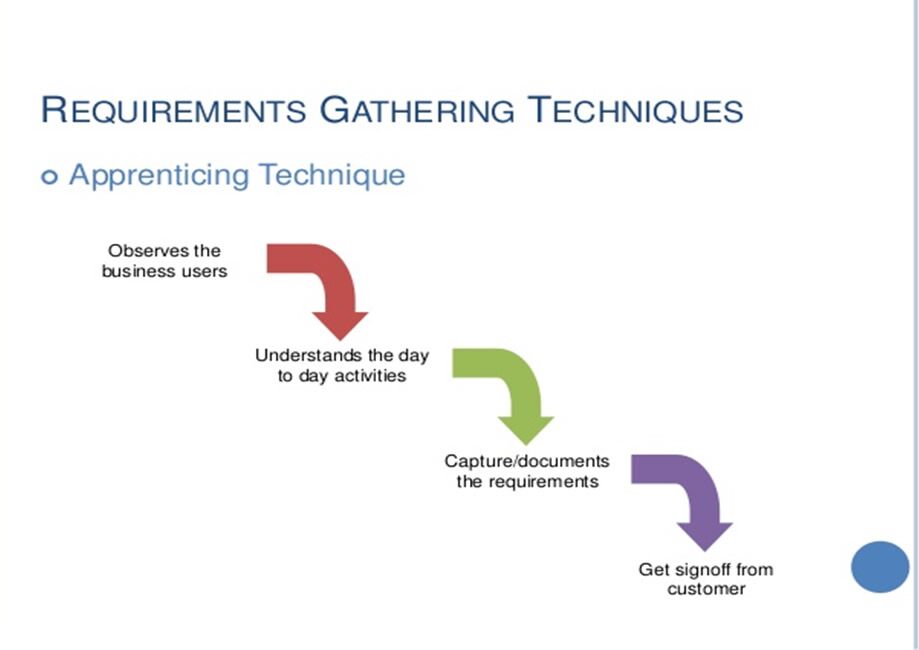
Step 2: Eliminate Language Ambiguity

Step 3: Identify Corner Cases

Step 4: Write User Stories

Step 5: Create a Definition Of “Done

**TECHNIQUES FOR GATHERING REQUIREMENTS**



* **One-on-one interviews**: Sit down with the client and ask them what they need. Asking open-ended questions and getting the client talking about their hopes for the project is a good way to understand their needs.
* **Group interviews**: Similar to the one-on-one, but with a group of people, the same types of questions are asked, but with more of the users in the room, they may pick up on other’s statements and expand or correct them. This can be helpful, but too many cooks spoil the broth and these meetings can stray from the goal.
* **Use of questionnaires**: Providing questionnaires to the customer to fill out can be a good starting point or supplemental approach to interviewing.
* **Use cases:** A use case of a story about how a certain process in the software should work, they may be easier for users to communicate clearly.
* **Brainstorming:** Brainstorming is used in requirement gathering to get as many ideas as possible from group of people. Generally used to identify possible solutions to problems, and clarify details of opportunities. The following basic rules for brainstorming ensure better results:

o  Start out by clearly stating the objective of the brainstorming session

o  Generate as many ideas as possible

o  Let your imagination soar

o  Do not allow criticism or debate while you are gathering information

o  After you have gathered information, reshape and combine ideas

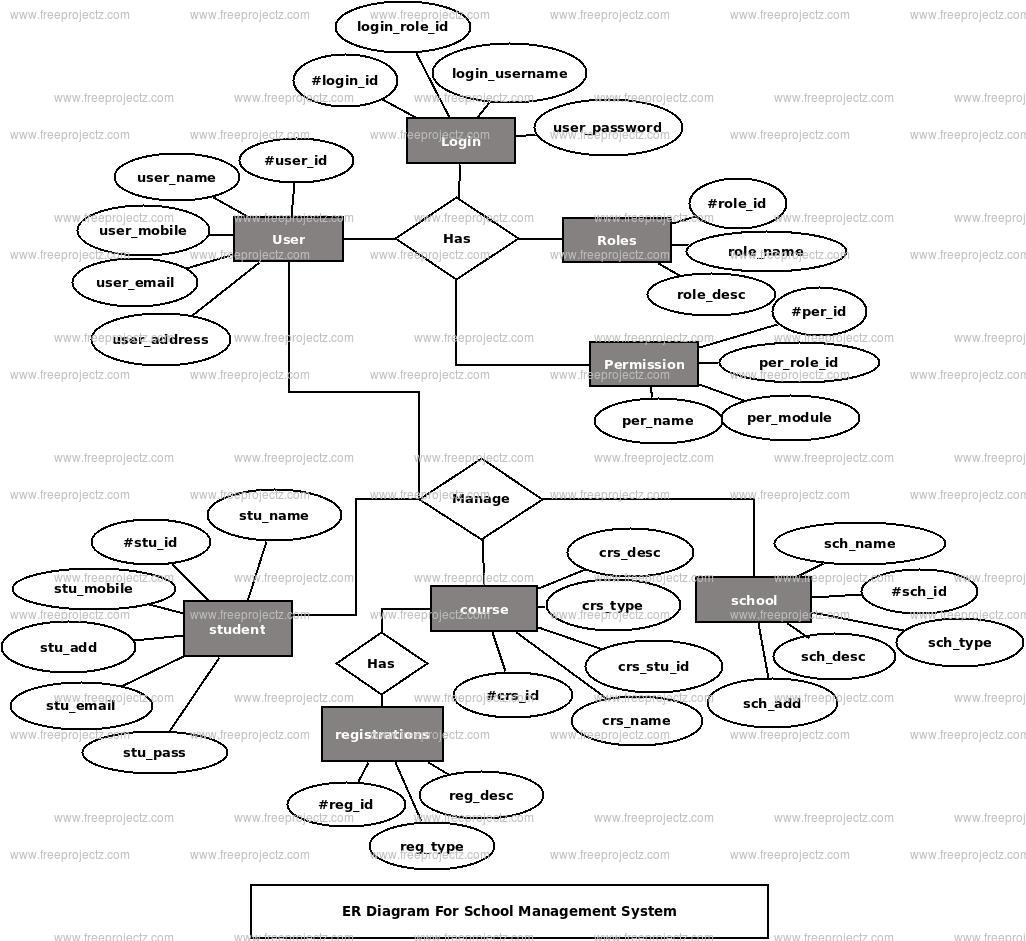
* **Document analysis:** Reviewing the documentation of an existing system can help when creating AS–IS process document, as well as driving gap analysis for scoping of migration projects. In an ideal world, we would even be reviewing the requirements that drove creation of the existing system – a starting point for documenting current requirements. Nuggets of information are often buried in existing documents that help us ask questions as part of validating requirement completeness.
* **Prototyping:** Prototyping is a relatively modern technique for gathering requirements. In this approach, you gather preliminary requirements that you use to build an initial version of the solution – a prototype. You show this to the client, who then gives you additional requirements. You change the application and cycle around with the client again. This repetitive process continues until the product meets the critical mass of business needs or for an agreed number of iterations.
* **Reverse engineering:** When a migration project does not have access to sufficient documentation of the existing system, reverse engineering will identify what the system does. It will not identify what the system should do, and will not identify when the system does the wrong thing.

As a software developer, there are many more techniques and methods of gathering good requirements; the above are only a few. There are many online resources for requirements gathering and development methodologies in general. When it comes to application development, a development team should be versed in how to gather as many of the requirements to start developing a software product.

Cost and time proved to be the two most important factors when determining which method is to be used. Many methods are used in supplement of each other.

**15.Make an ER diagram for school management system.**

This School Management System ER Diagram example depicts the relationship between elements such as student information and department information, as well as how they are related. The ERD example demonstrates the relationship between a student, a department, and information about the department head, courses, and more. These types of ERD are also useful in other fields. Administrators can use these ERDs to keep track of their employees, students, or clients. ER diagrams are classified into several types based on their application



**16.What is software testing? Is it possible to do complete testing?**

**SECTION A QUESTION 7**

**17.Briefly discuss the Test case design, Test Case & Test suite.**

## **Test cases**

A typical use of test case might be to use the same test script for testing multiple configurations. For instance, if you want to test a login script on three different browsers, such as Firefox, Internet Explorer, and Safari, you can create three different test case execution records in that test case. In a test case that is called Test Browsers you might include three testing scenarios:

* Test case execution record 1: Firefox and log-in test script
* Test case execution record 2: Internet Explorer and log-in test script
* Test case execution record 3: Safari and log-in test script

## **Test suites**

If each test case represents a piece of a scenario, such as the elements that simulate a completing a transaction, use a test suite. For instance, a test suite might contain four test cases, each with a separate test script:

* Test case 1: Login
* Test case 2: Add New Products
* Test case 3: Checkout
* Test case 4: Logout

Test suites can identify gaps in a testing effort where the successful completion of one test case must occur before you begin the next test case. For instance, you cannot add new products to a shopping cart before you successfully log in to the application. When you run a test suite in sequential mode, you can choose to stop the suite execution if a single test case does not pass. Stopping the execution is useful if running a test case in a test suite depends on the success of previous test cases.

Test suites are also useful for the following types of tests:

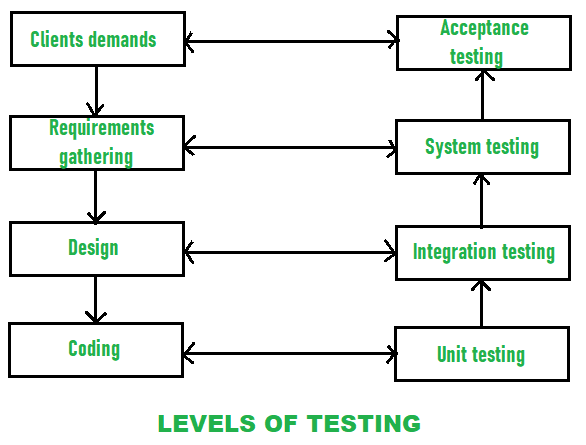
* Build verification tests: A collection of test cases that perform a basic validation of most the functional areas in the product. The tests are executed after each product build and before the build is promoted for use by a larger audience.
* Smoke tests: A collection of test cases that ensure basic product functionality. Typically, smoke tests are the first level of testing that is performed after changes are made to the system under test.
* End-to-End integration tests: A collection of test cases that cross product boundaries and ensure that the integration points between products are exercised and validated.
* Functional verification tests: A collection of test cases that focus on a specific product function. Executing this type of test with a test suite ensures that several aspects of a specific feature are tested.
* Regression tests: A collection of test cases that are used to make a regression pass over functional product areas.

**18.Explain the levels of testing**.

[Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) is an activity performed to identify errors so that errors can be removed to obtain a product with greater quality. To assure and maintain the quality of software and to represents the ultimate review of specification, design, and coding, Software testing is required. There are different levels of testing :

1. [Unit Testing](https://www.geeksforgeeks.org/unit-testing-software-testing/)**:**  
   In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that if they are fit for use by the developers. It is the smallest testable part of the software.
2. [Integration Testing](https://www.geeksforgeeks.org/software-engineering-integration-testing/)**:**  
   In this testing, two or more modules which are unit tested are integrated to test i.e. technique interacting components and are then verified if these integrated modules work as per the expectation or not and interface errors are also detected.
3. [System Testing](https://www.geeksforgeeks.org/system-testing/)**:**  
   In system testing, complete and integrated Softwares are tested i.e. all the system elements forming the system is tested as a whole to meet the requirements of the system.
4. [Acceptance Testing](https://www.geeksforgeeks.org/acceptance-testing-software-testing/)**:**  
   It is a kind of testing conducted to ensure whether the requirement of the users are fulfilled prior to its delivery and the software works correctly in the user’s working environment.

These testing can be conducted at various stages of software development. The levels of testing along with the corresponding software development phase is shown by the following diagram –



While performing the software testing, following [Testing principles](https://www.geeksforgeeks.org/types-software-testing/) must be applied by every software engineer:

* The requirements of customers should be traceable and identified by all different tests.
* Planning of tests that how tests will be conducted should be done long before the beginning of the test.
* The Pareto principle can be applied to software testing- 80% of all errors identified during testing will likely be traceable to 20% of all program modules.
* Testing should begin “in the small” and progress toward testing “in the large”.
* Exhaustive testing which simply means to test all the possible combinations of data is not possible.
* Testing conducted should be most effective and for this purpose, an independent third party is required.

**SECTION-C**

**1.What is the main difference between Quality Assurance, Quality Control and Software Testing?**

## Quality assurance

**Quality assurance**is process oriented. It is all about preventing defects by ensuring the processes used to manage and create deliverables works. Not only does it work, but is consistently followed by the team. Moreover, QA is about engineering processes that assure quality is achieved in an effective and efficient way.

For instance, if a defect is found and fixed, there is no guaranteeing it won’t pop back up. The role of QA is to identify the process that allowed the error to occur and re-engineer the system so that these defects won’t appear for the second time. The QA process verifies that the product will continue to function as the customer expects.

Though QC is absolutely necessary, QA is perhaps more important. By the time you reach the QC stage, for instance, fixing bugs becomes an expensive issue. Because of that, focusing efforts on improved QA processes is one of the best investments an organization can make.

Examples of QA include process definition and implementation, training, audits and selection of tools.

Top of Form



Bottom of Form

## **Quality control**‍

**Quality control**, alternatively, is product oriented. It is the function of software quality that determines the ending result is what was expected. Whereas QA is proactive, QC is reactive. QC detects bugs by inspecting and testing the product. This involves checking the product against a predetermined set of requirements and validating that the product meets those requirements.

Examples of QC include technical reviews, software testing and code inspections.

## Testing

**Testing** is a subset of QC. It is the process of executing a system in order to detect bugs in the product so that they get fixed. Testing is an integral part of QC as it helps demonstrate that the product runs the way it is expected and designed for.

To summarize, think of everything as an assembly line. QA can be thought of as the process to ensure the assembly line actually works, while QC is when the products coming off the assembly line are checked to verify they meet the required specifications.

Ultimately, both QA and QC are required for ensuring a successful product. When used together, they can help detect inefficient processes and identify bugs in the product. Moreover, QA and QC can help to develop and deliver a consistently high-quality product to your customers.

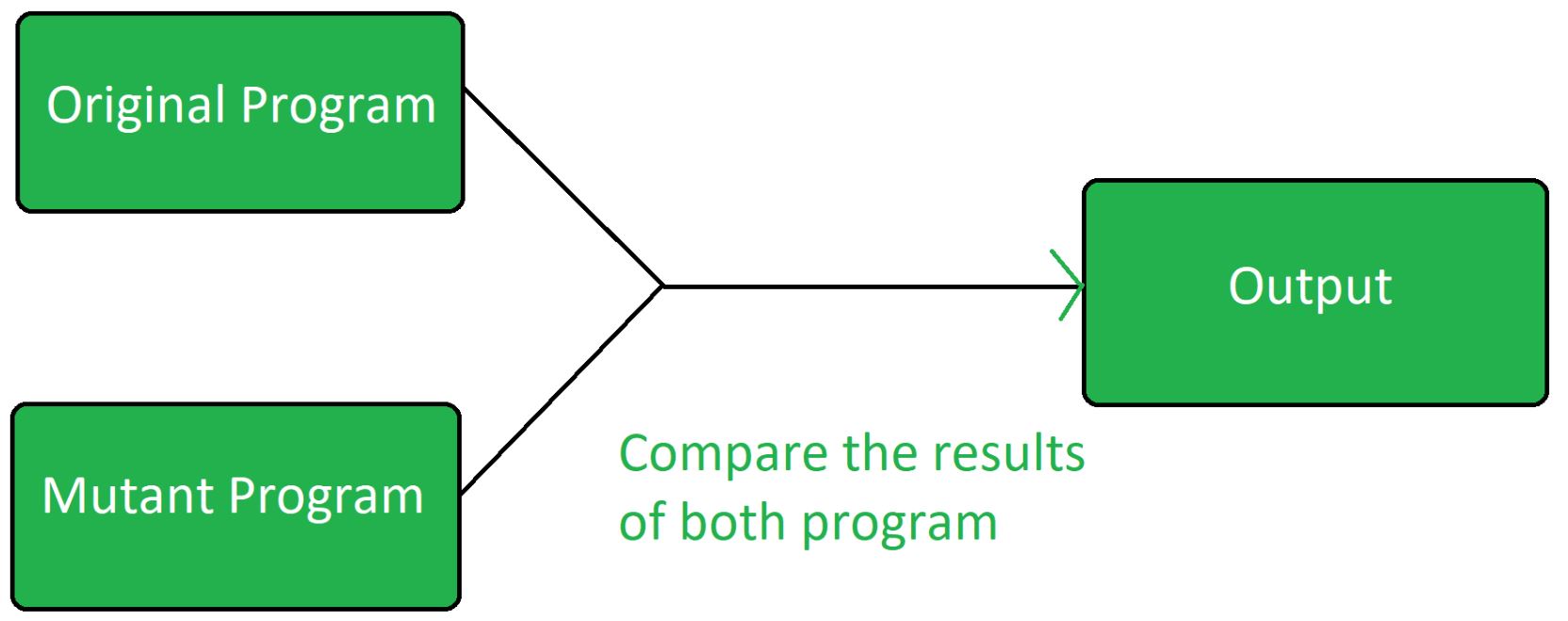
**2.Explain the waterfall model with its advantages and disadvantages.**

**SECTION A QUESTION 5**

**3.Explain Mutation testing and its types.**

**Mutation Testing** is a type of [Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) that is performed to design new software tests and also evaluate the quality of already existing software tests. Mutation testing is related to modification a program in small ways. It focuses to help the tester develop effective tests or locate weaknesses in the test data used for the program.

**History of Mutation Testing:**   
*Richard Lipton* proposed the mutation testing in 1971 for the first time. Although high cost reduced the use of mutation testing but now it is widely used for languages such as Java and XML.



Mutation Testing is a [White Box Testing](https://www.geeksforgeeks.org/software-engineering-white-box-testing/).

Mutation testing can be applied to design models, specifications, databases, tests, and XML. It is a structural testing technique, which uses the structure of the code to guide the testing process. It can be described as the process of rewriting the source code in small ways in order to remove the redundancies in the source code.

**Objective of Mutation Testing:**   
The objective of mutation testing is:

* To identify pieces of code that are not tested properly.
* To identify hidden defects that can’t be detected using other testing methods.
* To discover new kinds of errors or bugs.
* To calculate the mutation score.
* To study error propagation and state infection in the program.
* To assess the quality of the test cases.

**Types of Mutation Testing:**   
Mutation testing is basically of 3 types:

**1. Value Mutations:**   
In this type of testing the values are changed to detect errors in the program. Basically a small value is changed to a larger value or a larger value is changed to a smaller value. In this testing basically constants are changed.   
**2. Decision Mutations:**   
In decisions mutations are logical or arithmetic operators are changed to detect errors in the program.

**3. Statement Mutations:**   
In statement mutations a statement is deleted or it is replaces by some other statement.

d = 20;

**Tools used for Mutation Testing :**

* Judy
* Jester
* Jumble
* PIT
* MuClipse.

**Advantages of Mutation Testing:**

* It brings a good level of error detection in the program.
* It discovers ambiguities in the source code.
* It finds and solves the issues of loopholes in the program.
* It helps the testers to write or automate the better test cases.
* It provides more efficient programming source code.

**Disadvantages of Mutation Testing:**

* It is highly costly and time-consuming.
* It is not able for Black Box Testing.
* Some, mutations are complex and hence it is difficult to implement or run against various test cases.
* Here, the team members who are performing the tests should have good programming knowledge.
* Selection of correct automation tool is important to test the programs.

**4.Explain dynamic testing**

**Dynamic Testing** is a type of [Software Testing](https://www.geeksforgeeks.org/software-testing-basics/) which is performed to analyze the dynamic behavior of the code. It includes the testing of the software for the input values and output values that are analyzed. Dynamic Testing is basically performed to describe the dynamic behavior of code. It refers to the observation of the physical response from the system to variables that are not constant and change with time. To perform dynamic testing the software should be compiled and run. It includes working with the software by giving input values and checking if the output is as expected by executing particular test cases which can be done with either manually or with automation process. In 2 V’s i.e., Verification and Validation, Validation is Dynamic Testing.

## **Levels of Dynamic Testing**

There are various levels of Dynamic Testing. They are:

* Unit Testing
* Integration Testing
* System Testing
* Acceptance Testing

## **Dynamic Testing Process Phase**   **Advantages of Dynamic Testing**

* It discloses very difficult and complex defects.
* It detects the defects that can’t be detected by static testing.
* It increases the quality of the software product or application being tested.
* Dynamic testing detects security threats and ensure the better secure application.
* It can be used to test the functionality of the software at the early stages of development.
* It is easy to implement and does not require any special tools or expertise.
* It can be used to test the software with different input values.
* It can be used to test the software with different data sets.
* It can be used to test the software with different user profiles.
* It can be used to test the functionality of the code.
* It can be used to test the performance of the code.
* It can be used to test the security of the code.

## **Disadvantages of Dynamic Testing**

* It is a time consuming process as in dynamic testing whole code is executed.
* It increases the budget of the software as dynamic testing is costly.
* Dynamic testing may require more resources than static testing.
* Dynamic testing may be less effective than static testing in some cases.
* It is difficult to cover all the test scenarios.
* It is difficult to find out the root cause of the defects.

**5.A Illustrate equivalence class partitioning and boundary value analysis using suitable examples.**

**6.Explain DD-path with a suitable example**

**7.Explain Cause effect graphing Technique**

**Cause Effect Graphing based technique** is a technique in which a graph is used to represent the situations of combinations of input conditions. The graph is then converted to a decision table to obtain the test cases. Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions. But since there may be some critical behaviour to be tested when some combinations of input conditions are considered, that is why cause-effect graphing technique is used.

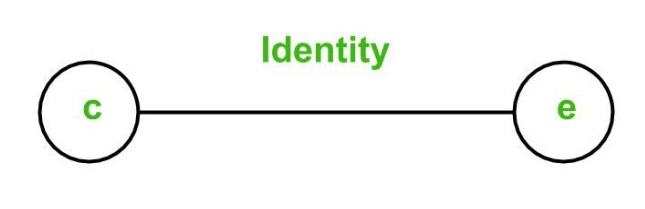
**Steps used in deriving test cases using this technique are:**

1. **Division of specification:**  
   Since it is difficult to work with cause-effect graphs of large specifications as they are complex, the specifications are divided into small workable pieces and then converted into cause-effect graphs separately.
2. **Identification of cause and effects:**  
   This involves identifying the causes(distinct input conditions) and effects(output conditions) in the specification.
3. **Transforming the specifications into a cause-effect graph:**  
   The causes and effects are linked together using Boolean expressions to obtain a cause-effect graph. Constraints are also added between causes and effects if possible.
4. **Conversion into decision table:**  
   The cause-effect graph is then converted into a limited entry decision table
5. **Deriving test cases:**  
   Each column of the decision-table is converted into a test case.

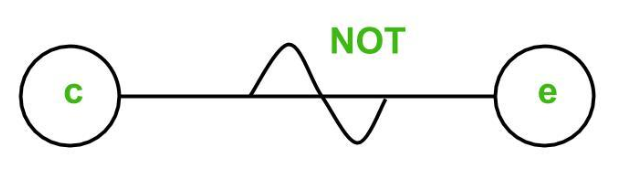
**Basic Notations used in Cause-effect graph:**  
Here **c** represents **cause** and **e** represents **effect**.

The following notations are always **used between a cause and an effect**:

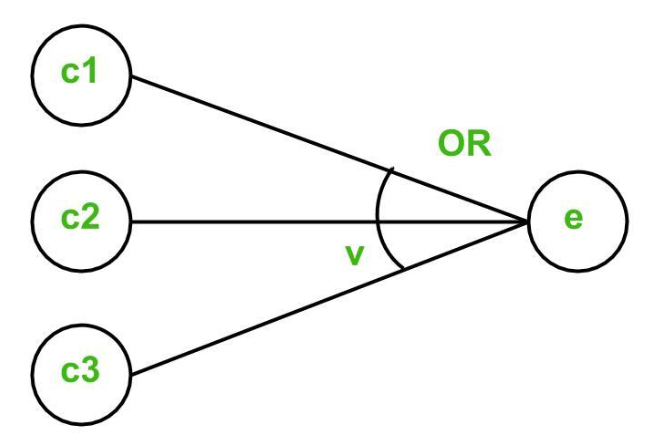
1. **Identity Function:** if c is 1, then e is 1. Else e is 0.



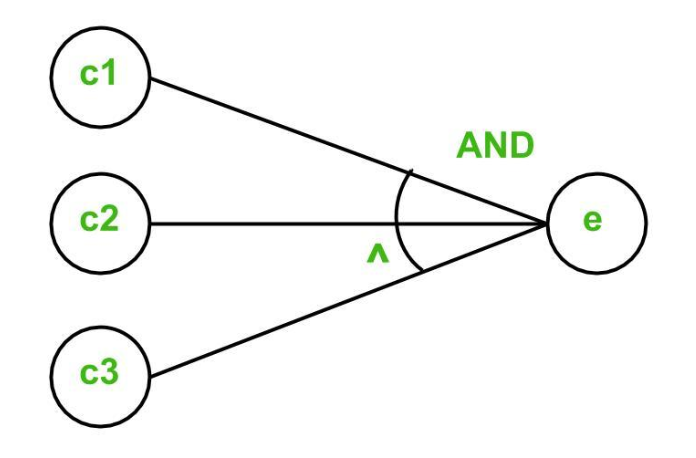
1. **NOT Function:** if c is 1, then e is 0. Else e is 1.



1. **OR Function:** if c1 or c2 or c3 is 1, then e is 1. Else e is 0.

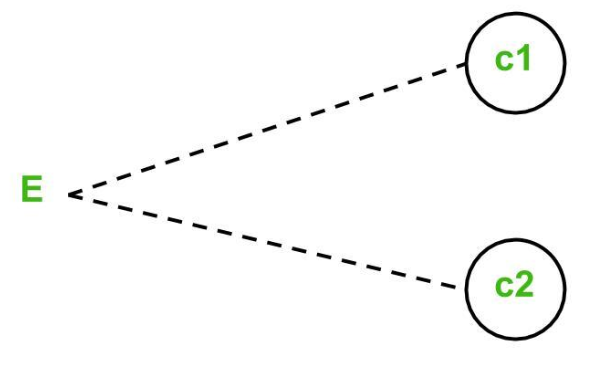


1. **AND Function:** if both c1 and c2 and c3 is 1, then e is 1. Else e is 0.

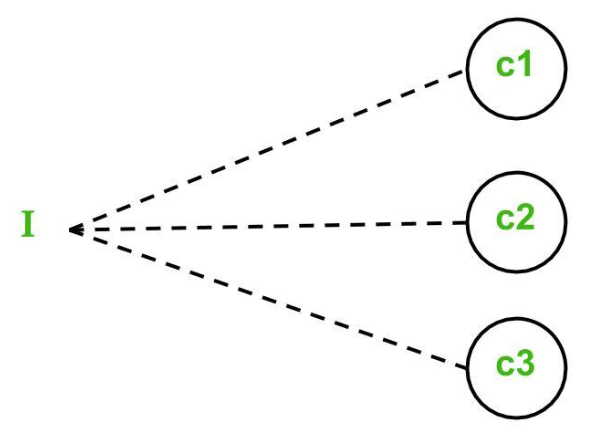


To represent some impossible combinations of causes or impossible combinations of effects, constraints are used. The following **constraints** are used in cause-effect graphs:

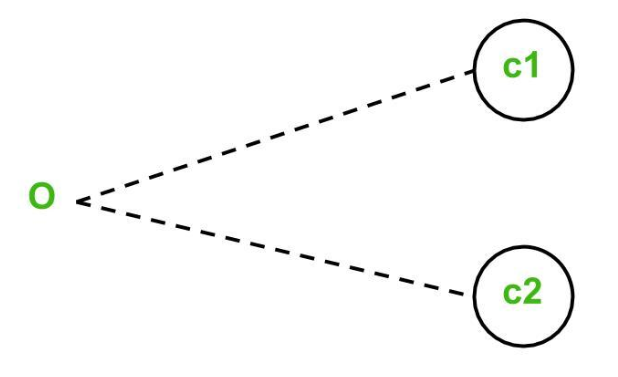
1. **Exclusive constraint** or **E-constraint:** This constraint exists between causes. It states that either c1 or c2 can be 1, i.e., c1 and c2 cannot be 1 simultaneously.



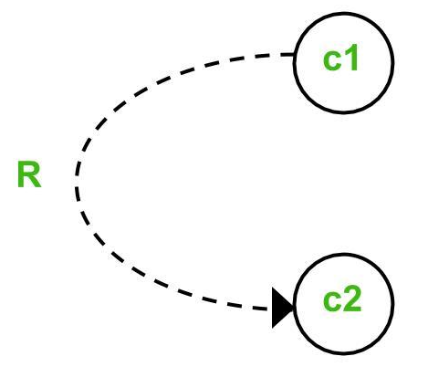
1. **Inclusive constraint**or **I-constraint:** This constraint exists between causes. It states that atleast one of c1, c2 and c3 must always be 1, i.e., c1, c2 and c3 cannot be 0 simultaneously.



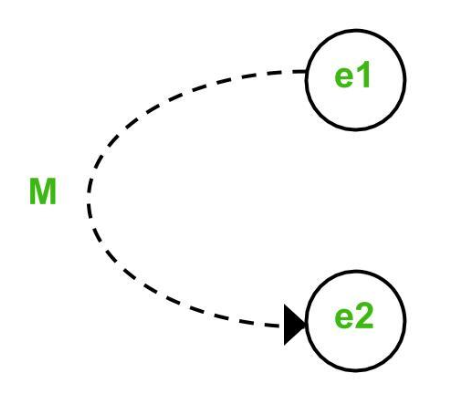
1. **One and Only One constraint** or **O-constraint:** This constraint exists between causes. It states that one and only one of c1 and c2 must be 1.



1. **Requires constraint**or **R-constraint:** This constraint exists between causes. It states that for c1 to be 1, c2 must be 1. It is impossible for c1 to be 1 and c2 to be 0.



1. **Mask constraint** or **M-constraint:** This constraint exists between effects. It states that if effect e1 is 1, the effect e2 is forced to be 0.

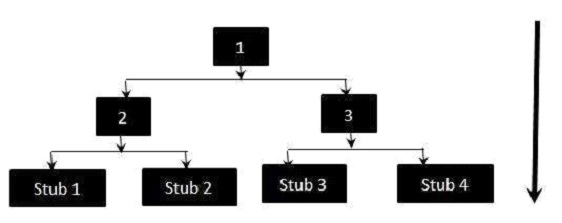


**8.Explain the bottom-up and top-down testing approach**

## **Top-down Integration Testing**

* It is also known as incremental integration testing.
* The higher level modules are first tested after which the lower level modules are tested.
* Once it is done, they are integrated.
* The higher level modules are the main modules and the lower level modules are the submodules.
* It uses stubs to simulate the sub-modules.
* If the sub-module hasn’t been fully developed, the stub acts like a replacement to it.
* It is useful in cases where significant defect occurs at the top of the program.
* The main module is designed first and then the submodules or subroutines are called from it.
* It is implemented on structure or procedure-oriented programming languages.
* It is a simple testing technique.
* It works on big to small components.
* Stub modules need to be produced.

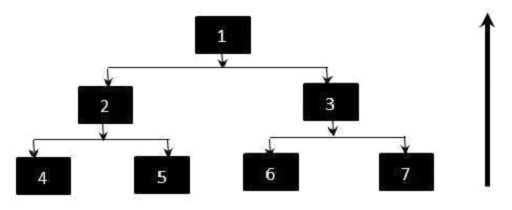
Here’s the flow diagram for stub −



## **Bottom-up Integration Testing**

* The lower level modules are tested first after which the higher level modules are tested.
* The lower level modules are the sub-modules and higher level modules are the main modules.
* It uses test drivers to initiate and pass the required data to the sub-modules.
* The test drivers are used to simulate the main module.
* If the main module hasn’t been developed yet, the driver works as a replacement to it.
* It is useful in cases where the crucial flaws are identified at the bottom of the program.
* It is generally implemented on object-oriented programming languages.
* It is highly complex and data intensive.
* It works on small to big components.
* The driver modules need to be produced.

Here’s the flow diagram −



**9.Explain the types of software maintenance. Why do we need software maintenance?**

**Section A Question 4**

**10.What are automated testing tools? What are the characteristics of modern tools?**

A test automation tool is a tool that helps teams and organizations automate their software testing needs, reducing the need for human intervention and thus achieving greater speed, reliability, and efficiency.  There’s more to it than that, though.

there are many different types of test automation tools. Just the sheer number of available options can make the experience of evaluating and choosing the best tool an overwhelming one.

To help you out, in this section we’ll briefly walk you through some of the ways in which we can categorize testing tools.

* **Codeless vs. code-based vs. hybrid tools.** There are test automation tools that require coding skills, while there are tools that don’t. There are also hybrid tools that bring together the best of both worlds. They allow testers and other professionals with no coding skills to create test cases with the use of some visual tool. Then, engineers can enhance those test cases with the use of a language such as [JavaScript](https://www.testim.io/blog/javascript-end-to-end-testing/).
* **Commercial vs. open-source.**Test automation tools can vary wildly when it comes to their pricing schemes and licenses. There are tools that are completely free (as in beer) and open-source. Others are closed-source but offer free versions or at least a free trial. Additionally, it’s been increasingly common for test automation tools to be offered in a SaaS model, in which the client pays a monthly or annual subscription.
* **Desktop vs web vs mobile.** Test automation tools also differ when it comes to the different types of software they support. You can have tools that target desktop (e.g. Windows) applications. Nowadays, it’s more common to immediately think of web and mobile apps when testing tools come up. Web testing, in itself, is a huge field that can be subdivided into many different types.
* **Production vs. non-production testing.** Finally, nowadays it’s increasingly more common and beneficial to perform some kinds of testing on the application after it’s in production. Techniques such as synthetic and non-synthetic monitoring, chaos engineering, A/B testing, canary releases, and [load](https://www.testim.io/blog/artillery-load-testing-introduction-see-how-your-code-scales/) and performance testing in production are a few that come to mind.

Following are characteristics of modern testing tools

1. It should use one or more testing strategy for performing testing on host as well as on target platform.
2. It should support GUI based test preparation.
3. It should provide complete code coverage and create test documentation in various formats (HTML/DOC/RTF...).
4. These tools should able to adopt the underlying hardware.
5. It should be easy to use.

6. Finally it should provide a clear report on test case, steps, test case status (PASS/FAIL).

**9.Explain the following:**

**(i)Reverse engineering**

**Software Reverse Engineering** is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

**Reverse Engineering Goals:** 

* Cope with Complexity.
* Recover lost information.
* Detect side effects.
* Synthesise higher abstraction.
* Facilitate Reuse.

**(ii)Re-engineering**

**Software Re-Engineering** is the examination and alteration of a system to reconstitute it in a new form. The principle of Re-Engineering when applied to the software development process is called software re-engineering. It positively affects software cost, quality, customer service, and delivery speed. In Software Re-engineering, we are improving the software to make it more efficient and effective.

**Software Re-Engineering Activities:** 

**1. Inventory Analysis**

**2. Document reconstructing**

**3. Reverse Engineering**

**4. Code Reconstructing**

**5. Data Restructuring**

**6. Forward Engineering**