Unit-1

Software Testing

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Testing as an Engineering activity

This is an exciting time to be a software developer. Software systems are becoming more challenging to build. They are playing an increasingly important role in society. People with software development skills are in demand. New methods, techniques, and tools are becoming available to support development and maintenance tasks. Because **software now has such an important role in our lives both economically and socially, there is pressure for software professionals to focus on quality issues. Poor quality software that can cause loss of life or property is no longer acceptable to society. Failures can result in catastrophic losses. Conditions demand software development staffs with interest and training in the areas of software product and process quality.** Highly qualified staff ensure that software products are built on time, within budget, and are of the highest quality with respect to attributes such as reliability, correctness, usability, and the ability to meet all user requirements. Using an engineering approach to software development implies that:

- the development process is well understood;
- projects are planned;
- life cycle models are defined and adhered to;
- standards are in place for product and process;
- measurements are employed to evaluate product and process quality;
- components are reused;
- validation and verification processes play a key role in quality determination;
- engineers have proper education, training, and certification.

Role of process in software quality

The need for software products of high quality has pressured those in the profession to identify and quantify quality factors such as **usability, testability, maintainability, and reliability, and to identify engineering practices that support the production of quality products** having these favorable attributes. Among the practices identified that contribute to the development of high-quality software are project planning, requirements management, development of formal specifications, structured design with use of information hiding and encapsulation, design and code reuse, inspections and reviews, product and process measures, education and training of software professionals, development and application of CASE tools, use of effective testing techniques, and integration of testing activities into the entire life cycle. In addition to identifying these individual best technical and managerial practices, software researchers realized that it was important to integrate them within the context of a high-quality software development process.

Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artifacts, such as project and test plans, design documents, code, and manuals. Process of software quality is represented graphically in Figure 1.1

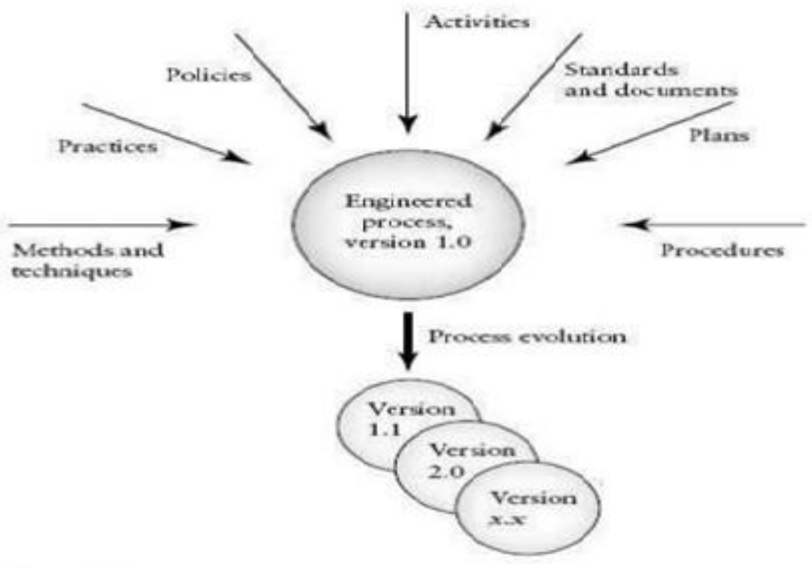

Figure 1.1 process in software quality

It also was clear that adding individual practices to an existing software development process in an ad hoc way was not satisfactory. The software development process, like most engineering artifacts, must be engineered. That is, it must be designed, implemented, evaluated, and maintained. As in other engineering disciplines, a software development process must evolve in a consistent and predictable manner, and the best technical and managerial practices must be integrated in a systematic way. These models allow an organization to evaluate its current software process and to capture an understanding of its state. Strong support for incremental process improvement is provided by the models, consistent with historical process evolution and the application of quality principles. The models have received much attention from industry, and resources have been invested in process improvement efforts with many successes recorded.

All the software process improvement models that have had wide acceptance in industry are high-level models, in the sense that they focus on the software process as a whole and do not offer adequate support to evaluate and improve specific software development sub processes such as design and testing. Most software engineers would agree that testing is a vital component of a quality software process, and is one of the most challenging and costly activities carried out during software development and maintenance.

Testing as a Process

Whenever we get any new project there is an initial project familiarity meeting. In this meeting, we basically discuss on who is the client? what is the project duration and when is its delivery? Who are all involved in the project i.e. manager, Tech leads, QA leads, developers, testers etc.? From the SRS (software requirement specification) project plan is developed. The responsibility of testers is to create a software test plan from this SRS and project plan. Developers start coding from the design. The project work is divided into different modules and these project modules are distributed among the developers.

In the meantime, **the responsibility of a tester is to create test scenario and write test cases according to the assigned modules. We try to cover almost all the functional test cases from SRS. The data can be maintained manually in some excel test case templates or bug tracking tools.** When developers finish the individual modules, those modules are assigned to the testers. Smoke testing is performed on these modules and if they fail this test, modules are reassigned to the respective developers for a fix.

For passed modules, manual testing is carried out from the written test cases. If any bug is found that gets assigned to the module developer and get logged in bug tracking tool. On bug fix, a tester does bug verification and regression testing of all related modules. **If bug passes the verification it is marked as verified and marked as closed. Otherwise above mentioned bug cycle gets repeated**. Different tests are performed on individual modules and integration testing on module integration. These tests include Compatibility testing i.e. testing application on different hardware, OS versions, software platform, different browsers etc.

Load and stress testing are also carried out according to SRS. Finally, system testing is performed by creating virtual client environment. On passing all the test cases test report is prepared and the decision is taken to release the product!

So this was a brief outline of the process of project life cycle.

Details of each testing step: Given below are the details of each testing step that is carried out in each software quality and testing life cycle specified by IEEE and ISO standards**:**

#1) SRS Review: Review of the software requirement specifications
#2) Objectives are set for Major releases
#3) Target Date planned for the Releases
#4) Detailed Project Plan is built. This includes the decision on Design Specifications
#5) Develop Test Plan based on Design Specifications
#6) Test Plan: This includes objectives, the methodology adopted while testing, features to be tested and not to be tested, risk criteria, testing schedule, multi-platform support and the resource allocation for testing.
#7) Test Specifications
This document includes technical details (Software requirements) required prior to testing.
#8) Writing of Test Cases

- Smoke (BVT) test cases
- Sanity Test cases
- Regression Test Cases
- Negative Test Cases
- Extended Test Cases

#9) Development – Modules are developed one by one
#10) Installers Binding: Installers are built around the individual product.
#11) Build procedure : A build includes Installers of the available products – multiple platforms.
#12) Testing: Smoke Test (BVT): Basic application test to take decision on further testing

- Testing of new features
- Cross-browser and cross-platform testing
- Stress testing and memory leakage testing.

#13) Test Summary Report

- Bug report and other reports are created

#14) Code freezing

- No more new features are added at this point.

#15) Testing

- Build and regression testing.

#16) Decision to release the product
#17) Post-release scenario for further objectives.

Software Testing Principles

1) Exhaustive testing is not possible

Yes!Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application. And the million dollar question is, how do you determine this risk ? To answer this let's do an exercise. In your opinion, Which operation is most likely to cause your Operating system to fail?

I am sure most of you would have guessed, Opening 10 different application all at the same time. So if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly which brings us to our next principle Defect Clustering

2) Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules. By experience, you can identify such risky modules. But this approach has its own problems. If the same tests are repeated over and over again , eventually the same test cases will no longer find new bugs.

3) Pesticide Paradox

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects. To overcome this, the test cases need to be regularly reviewed & revised , adding new & different test cases to help find more defects. Testers cannot simply depend on existing test techniques. **He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug free**. To drive home this point , let's see this video of public launch of Windows 98.

4) Testing shows presence of defects: Hence, testing principle states that - Testing talks about the presence of defects and don’t talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness. But what if , you work extra hard , taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients. This leads us to our next principle, which states that- Absence of Error.

5) Absence of Error: It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. Absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements. To solve this problem , the next principle of testing states that Early Testing.

6) Early Testing: Early Testing - **Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages.** It is much cheaper to fix a defect in early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined.

7) Testing is context dependent

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software’s are not identical. You might use a different approach, methodologies, techniques and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.

Summary of the Seven Testing Principles: Summary of Testing Principles is shown in Table 1.1.

Table 1.1 Testing Principles

Principle 1	Testing shows presence of defects
Principle 2	Exhaustive testing is impossible
Principle 3	Early Testing
Principle 4	Defect Clustering

Principle 5	Pesticide Paradox
Principle 6	Testing is context dependent
Principle 7	Absence of errors - fallacy

Myth: "Principles are just for reference. I will not use them in practise."
This is so very untrue. Test Principles will help **you create an effective test strategy and draft error catching test cases.** But learning testing principles is just like learning to drive for the first time.

Initially while you learn to drive, you pay attention to each and everything like gear shifts, speed, clutch handling, etc. But with experience, you just focus on driving the rest comes naturally. Such that you even hold conversations with other passengers in the car. Same is true for testing principles. Experienced testers have internalized these principles to a level that they apply them even without thinking. Hence the myth that the principles are not use in practice is simply not true.

<u>Tester Role in Software development</u>

A Software tester (software test engineer) **should be capable of designing test suites and should have the ability to understand usability issues. Such a tester is expected to have sound knowledge of software test design and test execution methodologies**. It is very important for a software tester to have great communication skills so that he can interact with the development team efficiently. The roles and responsibilities for a usability software tester are as follows:

1. A Software Tester is responsible for designing testing scenarios for usability testing.
2. He is responsible for conducting the testing, thereafter analyze the results and then submit his observations to the development team.
3. He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.
4. Software Testers are often responsible for creating test-product documentation and also has to participate in testing related walk through.

A software tester has different sets of roles and responsibilities. He should have in depth knowledge about software testing. He should have a good understanding about the system which means technical (GUI or non-GUI human interactions) as well as functional product aspects. In order to create test cases it is important that the software tester is aware of various testing techniques and which approach is best for a particular system. He should know what are various phases of software testing and how testing should be carried out in each phase. The responsibilities of the software tester include:

1. Creation of test designs, test processes, test cases and test data.
2. Carry out testing as per the defined procedures.
3. Participate in walkthroughs of testing procedures.
4. Prepare all reports related to software testing carried out.
5. Ensure that all tested related work is carried out as per the defined standards and procedures.

Software Test Manager Role

Managing or leading a test team is not an easy job. The company expects the test manager to know testing methodologies in detail. A test manager has to take very important decisions regarding the testing environment that is required, how information flow would be managed and how testing procedure would go hand in hand with development. He should have sound knowledge about both manual as well as automated testing so that he can decide how both the methodologies can be put together to test the software. A test manager should have sound knowledge about the business area and the client's requirement, based on that he should be able to design a test strategy, test goal and objectives. He should be good at project planning, task and people coordination, and he should be familiar with various types of testing tools. Many people get confused between the roles and responsibilities of a test manager and test lead. For a clarification, a test lead is supposed to have a rich technical experience which includes, programming, handling database technologies and various operating systems, whereas he may not be as strong as Software Test Manager regarding test project management and coordination. The **responsibilities of the test manager** are as follows:

1. Since the test manager represents the team he is responsible for all interdepartmental meetings.
2. Interaction with the customers whenever required.
3. A test manager is responsible for recruiting software testing staff. He has to supervise all testing activities carried out by the team and identify team members who require more training.
4. Schedule testing activities, create budget for testing and prepare test effort estimations.
5. Selection of right test tools after interacting with the vendors. Integration of testing and development activities.
6. Check the quality of requirements, how well they are defined.
7. Trace test procedures with the help of test traceability matrix.

Software Test Automator Role

Software test automator or an automated test engineer should have very good understanding of what he needs to test- GUI designs, load or stress testing. He should be proficient in automation of software testing, and he should be able to design test suites accordingly. A software test automator should be comfortable using various kinds of automation tools and should be capable of upgrading their skills with changing trends. He should also have programming skills so that he is able to write test scripts without any issues. The responsibilities of a tester at this position are as follows:

1. He should be able to understand the requirement and design test procedures and test cases for automated software testing.
2. Design automated test scripts that are reusable.
3. Ensure that all automated testing related activities are carried out as per the standards defined by the company.

Software Testing Artifacts - Test Reports

What is Artifact?

An artifact is one of many kinds of tangible by-products produced during the development of software. Some artifacts (e.g., use cases, class diagrams, and other Unified Modeling Language (UML) models, requirements and design documents) help describe the function, architecture, and design of software. Other artifacts are concerned with the process of development itself—such as project plans, business cases, and risk assessments.

Terms used in Artifacts:

Wrong: When requirements are implemented not in the right way. This defect is a variance from the given specification. It is Wrong!

Missing: A requirement of the customer that was not fulfilled. This is a variance from the specifications, an indication that a specification was not implemented, or a requirement of the customer was not noted correctly.

Extra: A requirement incorporated into the product that was not given by the end customer. This is always a variance from the specification, but may be an attribute desired by the user of the product. However, it is considered a defect because it's a variance from the existing requirements.

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

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

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

Test Reports: Test team sent out numerous test reports to different stakeholders for various purposes. Test reports are used to communicate the test aspects clearly among stakeholders. Test report information's referred and analyzed in future to develop better projects in future. Information on test report is decided as per the purpose and the audience. Before preparing the test report tester should know whom the audiences for the particular test report are. Then tester should prepare the test report according the end purpose of the particular test report.

Why Test Reports Are Important?

- It offers knowledge transfers and sharing experience among team members, management and client to improvement.
- **Test reports facilitate to maintain the test record with the required detail**. This gives information availability.
- Test report saves the time and effort to put on a project. The information is available to access quickly and effortless.
- This gives the facility to keep the information and will reduce the ambiguity and distraction.
- **Versioning** provide us the details on what are the changes done on particular test matter.
- Since we maintain test reports, the **other members can review and give the feedback for betterment in the project.**
- It keeps a record on the testing activities carried out. Client will like to see the process and the test carried out.
- We shall analyze the test report and use the lesson learn from it to the future projects.
- It serves as a proof of evident to the testing carried out.
- *Test reports helps to make better decision.*

Test Result Report:

In test result report, document the detailed test result. Usually test result report is sent periodically. Test result report provides the information of test execution and test result in detail. It also contain conditions, assumptions, constrain of test. Also mention which test item is tested by whom. Detail of remaining work is marked in the test result report to show how much remaining work to be done in the project is shown in Table 1.2.

Table 1.2 Test Result Report

Item	**Description**
Test item	Specific test item or software. (E.g. Software version or module.)
Description of the test item	Describe the detail of test item.
Test result	**Whether the test items is pass, failed, blocked or on hold.**
Test sate	Whether the test items is completed, in progress or not started.
Defect(s) detail	If the test item is failed then mention the defect detail. Add a linkage to the defect.

Total percentage completed	State the completed test percentage.
Summary of the test result	Mention the number of executed test cases, defect count, total test completion percentage.

Test Execution Report:

Test execution report contain the execution detail and the result of test execution. This is generally prepared to send to higher management from testing team to show the state of the test execution and test progress. When we deliver the software to client, we shall send the full test execution detail in brief as shown in Table 1.3. This would give a better understanding on the executed test and coverage to client.

Table 1.3 Test Execution Report

Element	**Description**
Test item	Item, module or software that is been testing.
Scenario (Possibilities with conditions)	Scenario of the test.
Responsible tester	Who tested the particular test item?
Date of test execution	Date of the test executed.
Status	Status of test execution.
Test environment	Detail of test environment. Detail of hardware, software and tools.
Risk and constrain	Any risk that impact the test execution and any constrain.
Test execution summary	This provide detail of test execution by software section, test type, defect rage etc.
Defect summary and detail	Defect count. Rate of open and closed defect. Severity and status wise defects.
Test cases detail	Planned and executed test cases. Status and result of execution and test case.
Test progress	Over all progress of test and which cycle the test execution is going on.

Test Summary Report:

Test summary report provides a detail of testing carried out throughout the software development life cycle. The items in a test summary report vary from organization to organization and also vary for different projects. Below are some common items stating in a test summary report is shown in Table 1.4. Information on test report is based on the audience of the test report. Audience can be client, management, business analyst, developers, testing team and organizational level members etc.

Table 1.4 Test Summary Report

Items	**Content**
Introduction	Write the purpose of the test summary report. An introduction on particular test’s final state.
Overview	Overview of testing and tested software.
Scope	Explain the scope of the testing.
Out of scope	Any out of scopes in testing

Test types	All different type of tests executed.
Defect summary	Defects count severity wise, software area wise, assigned developer wise, test category wise etc. If there is any limitation or open defect, clearly describe it.
Exit criteria	Explain the exit criteria. Explain the factors considered to closure of test.
Coverage of testing	Describe the covered areas in the software. State the test coverage against requirement.
Test execution summary	Executed test cases and their result.
Test matrixes	Detail and explanation of used test matrixes and the calculated value of those matrixes.
Lesson learn	Any lesson learn in the specific project. This can be utilized in the future project.
Conclusion	Conclusion of the testing and software project.

User Acceptance Testing: User acceptance test report (Table 1.5) is created while conducting the user acceptance test and after conducting the user acceptance testing. It state the detail of conducted user acceptance test and the result of the user acceptance test. It also list the defect uncounted in user acceptance testing. User acceptance testing created during and after the user acceptance testing by the people who conduct the user acceptance test. This will be amazed to check test matrices like defect leakage rate, missed requirement etc.

Table 1.5 User Acceptance Testing

Element	**Description**
Test coverage	
Item tested	Item that tested.
Test area	Tested item on which area.
Test description	Describe the test item in detail.
Test date	Date which test is conducted.
UAT result	
UAT item	Encountered item during user acceptance testing.
Type of UAT item	Whether the item raised during the user acceptance testing is defect or change request or enhancement or improvement.
Date raised	Date the UAT item found.
Severity	Severity of the UAT item. Level of impact to the system by the identified UAT item.
Priority	Priority of UAT item. How soon the UAT item should be fixed.
Raised by	Person who raised the UAT item
Status	Status of the UAT item. Whether it’s fixed or not fixed or need more clarification like wise.
Comment	Any comment for the UAT item
Test environment	Test environment detail where the user acceptance testing was done.
Defect summary	Provide defect detail and count as a summary.

Limitations of Testing

Limitations are:

- You cannot test a program completely
- We can only test against system requirements
- May not detect errors in the requirements.
- Incomplete or ambiguous requirements may lead to inadequate or incorrect testing.
- Exhaustive (total) testing is impossible in present scenario.
- Time and budget constraints normally require very careful planning of the testing effort.
- Compromise between thoroughness and budget.
- Test results are used to make business decisions for release dates.
- Even if you do find the last bug, you'll never know it
- You will run out of time before you run out of test cases
- You cannot test every path
- You cannot test every valid input
- You cannot test every invalid input

Challenges in Software Testing

Software Testing has lot of challenges both in manual as well as in automation. Generally in manual testing scenario developers through the build to test team assuming the responsible test team or tester will pick the build and will come to ask what the build is about? This is the case in organizations not following so-called 'processes'. Tester is the middleman between developing team and the customers, handling the pressure from both the sides. And I assume most of our readers are smart enough to handle this pressure. Aren't you?

This is not the case always. Some times testers may add complications in testing process due to their unskilled way of working. **In this post I have added most of the testing challenges created due to testing staff, developing staff, testing processes and wrong management decisions. So here we go with the top challenges:**

1) Testing the complete application: Is it possible? I think impossible. There are millions of test combinations. It's not possible to test each and every combination both in manual as well as in automation testing. If you try all these combinations you will never ship the product ;-)

2) Misunderstanding of company processes: Some times you just don't pay proper attention what the company-defined processes are and these are for what purposes. There are some myths in testers that they should only go with company processes even these processes are not applicable for their current testing scenario. This results in incomplete and inappropriate application testing.

3) Relationship with developers: Big challenge. Requires very skilled tester to handle this relation positively and even by completing the work in testers way. There are simply hundreds of excuses developers or testers can make when they are not agree with some points. For this tester also requires good communication, troubleshooting and analyzing skill.

4) Regression testing: When project goes on expanding the regression testing work simply becomes uncontrolled. Pressure to handle the current functionality changes, previous working functionality checks and bug tracking.

5) Lack of skilled testers: I will call this as 'wrong management decision' while selecting or training testers for their project task in hand. These unskilled fellows may add more chaos than simplifying the testing work. This results into incomplete, insufficient and ad-hoc testing throughout the testing life cycle.

6) Testing always under time constraint: Hey tester, we want to ship this product by this weekend, are you ready for completion? When this order comes from boss, tester simply focuses on task completion and not on the test coverage and quality of work. There is huge list of tasks that you need to complete within specified time. This includes writing, executing, automating and reviewing the test cases.

7) **Which tests to execute first?** If you are facing the challenge stated in point no 6, then how will you take decision which test cases should be executed and with what priority? Which tests are important over others? This requires good experience to work under pressure.

8) **Understanding the requirements:** Some times testers are responsible for communicating with customers for understanding the requirements. What if tester fails to understand the requirements? Will he be able to test the application properly? Definitely No! Testers require good listening and understanding capabilities.

9) **Automation testing:** Many sub challenges – Should automate the testing work? Till what level automation should be done? Do you have sufficient and skilled resources for automation? Is time permissible for automating the test cases? Decision of automation or manual testing will need to address the pros and cons of each process.

10) **Decision to stop the testing:** When to stop testing? Very difficult decision. Requires core judgment of testing processes and importance of each process. Also requires 'on the fly' decision ability.

11) **One test team under multiple projects:** Challenging to keep track of each task. Communication challenges. Many times results in failure of one or both the projects.

12) **Reuse of Test scripts:** Application development methods are changing rapidly, making it difficult to manage the test tools and test scripts. Test script migration or reuse is very essential but difficult task.

13) **Testers focusing on finding easy bugs:** If organization is rewarding testers based on number of bugs (very bad approach to judge testers performance) then some testers only concentrate on finding easy bugs those don't require deep understanding and testing. A hard or subtle bug remains unnoticed in such testing approach.

14) **To cope with attrition:** Increasing salaries and benefits making many employees leave the company at very short career intervals. Managements are facing hard problems to cope with attrition rate. Challenges – New testers require project training from the beginning, complex projects are difficult to understand, delay in shipping date! These are some top software testing challenges we face daily. Project success or failure depends largely on how you address these basic issues.

Testing and Debugging

Testing means verifying correct behavior. Testing can be done at all stages of module development: requirements analysis, interface design, algorithm design, implementation, and integration with other modules. In the following, attention will be directed at implementation testing. Implementation testing is not restricted to execution testing. An implementation can also be tested using correctness proofs, code tracing, and peer reviews, as described below.

Debugging **is a cyclic** activity involving execution testing and code correction. The testing that is done during debugging has a different aim than final module testing. Final module testing aims to demonstrate correctness, whereas testing during debugging is primarily aimed at locating errors. This difference has a significant effect on the choice of testing strategies.

Preconditions for Effective Debugging

In order to avoid excessive time spent on debugging, the programmer should be mentally prepared for the effort. The following steps are useful to prepare for debugging.

- **Understand the design and algorithm** - If you are working on a module and you do not understand its design or its algorithms, then debugging will be very difficult. If you don't understand the design then you can't test the module because you do not know what it is supposed to do. If you don't understand the algorithms then you will find it very difficult to locate the errors that are revealed by testing. A second reason for the importance of understanding algorithms is that you may need that understanding in order to construct good test cases. This is especially true for algorithms for complex data structures.

- **Check correctness** - There are several methods for checking correctness of an implementation prior to execution.
- **Correctness proofs** - One useful code check is to examine code using the logical methods of correctness proofs. For example, if you know preconditions, invariants, terminating conditions, and post conditions for a loop then there are some easy checks that you can make. Does the precondition, together with any loop entry code imply that the invariant is initially true? Does the loop body preserve the invariant? Does execution of the loop body make progress towards loop termination? Does the invariant, together with the loop terminating condition and loop exit code, imply the post condition? Even if these checks don't find all errors, you will often gain a better understanding of the algorithm by making the checks.
- **Code tracing** - Often, errors can be detected by tracing through the execution of various calls to module services, starting with a variety of initial conditions for the module. For poorly understood psychological reasons, tracing works best if you are describing your tracing to someone else. In order to be effective, tracing of a procedure or function should be done assuming that calls to other procedures and functions work correctly, even if they are recursive calls. If you trace into a called procedure or function then you will find yourself dealing with too many levels of abstraction. This usually leads to confusion. If there is any doubt about the called procedures and functions then they can be traced separately to verify that they perform according to specifications. Again, tracing may not catch all errors, but it can enhance your understanding of algorithms.
- **Peer reviews** - A peer review involves having a peer examine your code for errors. To be effective, the peer should either already be familiar with the algorithm, or should be given the algorithm and code in advance. When the reviewer meets with the code writer, the code writer should present the code with explanations of how it correctly implements the algorithm. If the reviewer doesn't understand or disagrees with part of the implementation, they discuss that part until both are in agreement about whether or not it is an error. The reviewer's role is only as an aid to detecting errors. It is left to the implementer to correct them. Much of the benefit of a peer review derives from the psychology of presenting how something works. Often the code writer discovers his or her own errors during the review. In any case, it is useful to have an outsider review your work in order to get a different perspective and to discover blind spots that seem to be inherent in evaluating your own work. Like code tracing, peer reviews can be time consuming. For class work, a peer review of an entire module is not likely to pay for itself in terms of instructional value. So reviews should be restricted to short segments of code. For commercial programming, however, quality of the code is much more important. Thus peer reviews are a significant part of a software quality assurance program.
- **Anticipate errors** - Unfortunately, humans make errors with correctness arguments and sometimes miss cases in code tracing, and peers don't always catch errors either. So a programmer should be prepared for some errors remaining in the code after the steps listed above. Hopefully, there won't be too many.
- **Requirements for Debugging:** To effectively debug code you need two capabilities. First, you need to be able to efficiently call on the services provided by the module. Then you need to be able to get information back about results of the calls, changes in the internal state of the module, error conditions, and what the module was doing when an error occurred.
- **Driving the module:** To effectively debug a module, it is necessary to have some method for calling upon the services provided by the module. There are two common methods for doing this.
- **Hardwired drivers -** A hardwired driver is a main program module that contains a fixed sequence of calls to the services provided by the module that is being tested. The sequence of calls can be modified by rewriting the driver code and recompiling it. For testing modules whose behavior is determined by a vary small number of cases, hardwired drivers offer the advantage of being easy to construct. If there are too many cases, though, they have the shortcoming that a considerable effort is involved in modifying the sequence of calls.
- **Command interpreters -** A command interpreter drives the module under test by reading input and interpreting it as commands to execute calls to module services. Command interpreters can be

designed so that the commands can either be entered interactively or read from a file. Interactive command interpretation is often of great value in early stages of debugging, whereas batch mode usually is better for later stages of debugging and final testing. The primary disadvantage of command interpreters is the complexity of writing one, including the possibility that a lot of time can be spent debugging the interpreter code. This is mitigated by the fact that most of the difficult code is reusable, and can be easily adapted for testing different kinds of modules. For almost all data structure modules, the flexibility offered by command interpreters makes them a preferred choice.

Obtaining information about the module

Being able to control the sequence of calls to module services has little value unless you can also obtain information about the effects of those calls. If the services generate output then some information is available without any further effort. However, for many modules, including data structure modules, the primary effect of calls to services is a change in the internal state of the module. This leads to needs for three kinds of information for debugging.

- **Module state** - Data structure modules generally have services for inserting and deleting data. These services almost never generate output on their own, and often do not return any information through parameters. Therefore, in order to test or debug the module, the programmer must add code that provides information about changes in the internal module state. Usually, the programmer adds procedures that can display the data contents of the module. These procedures are made available to the driver module, but are usually removed or made private when testing is complete. For debugging, it is useful to have procedures that show internal structure as well as content.
- **Module errors -** When a module has a complex internal state, with incorrect code it is usually possible for invalid states to arise. Also, it is possible that private subroutines are called incorrectly. Both of these situations are module errors. When practical, code can be added to the module to detect these errors.
- **Execution state -** In order to locate the cause of module errors, it is necessary to know what services and private subroutines have been called when the error occurs. This is the execution state of the module. One common method for determining the execution state is the addition of debugging print statements that indicate entry and exit from segments of code.

Principles of Debugging

- **Report error conditions immediately** - Much debugging time is spent zeroing in on the cause of errors. The earlier an error is detected, the easier it is to find the cause. If an incorrect module state is detected as soon as it arises then the cause can often be determined with minimal effort. If it is not detected until the symptoms appear in the client interface then may be difficult to narrow down the list of possible causes.
- **Maximize useful information and ease of interpretation** - It is obvious that maximizing useful information is desirable, and that it should be easy to interpret. Ease of interpretation is important in data structures. Some module errors cannot easily be detected by adding code checks because they depend on the entire structure. Thus it is important to be able to display the structure in a form that can be easily scanned for correctness.
- **Minimize useless and distracting information -** Too much information can be as much of a handicap as too little. If you have to work with a printout that shows entry and exit from every procedure in a module then you will find it very difficult to find the first place where something went wrong. Ideally, module execution state reports should be issued only when an error has occurred. As a general rule, debugging information that says "the problem is here" should be preferred in favor of reports that say "the problem is not here".
- **Avoid complex one-use testing code** - One reason why it is counterproductive to add module correctness checks for errors that involve the entire structure is that the code to do so can be quite complex. It is very discouraging to spend several hours debugging a problem, only to find that the error

was in the debugging code, not the module under test. Complex testing code is only practical if the difficult parts of the code are reusable.

Debugging Aids

Aids built into programming language

- **Assert statements** - Some Pascal compilers and all C compilers that meet the ANSI standard have assert procedures. The assert procedure has a single parameter, which is a Boolean expression. When a call to assert is executed the expression is evaluated. If it evaluates to true then nothing happens. If it evaluates to false then the program terminates with an error message. The assert procedure can be used for detecting and reporting error conditions.
- **Tracebacks** - Many Pascal compilers generate code that results in tracebacks whenever a runtime error occurs. A traceback is a report of the sequence of subroutines that are currently active. Sometimes a traceback will also indicate line numbers in the active subroutines. If available, a traceback reveals where the runtime error occurred, but it is up to the programmer to determine where the cause lies.
- **General purpose debuggers** - Many computer systems or compilers come with debugging programs. For example, most UNIX operating systems have general purpose debuggers such as sdb and dbx. Debugging programs provide capabilities for stepping through a program line-by-line and running a program with breakpoints set by the user. When a line with a breakpoint is about to be executed the program is interrupted so that the user can examine or modify program data. Debugging programs also can provide tracebacks in case of run-time errors. Debuggers are often difficult to learn to use effectively. If they are the only tool used for debugging then it is likely that they will not save much time. For example, debugging a data structure module with a debugger, but without a good test driver, will likely result in spending a lot of time getting piecemeal information about errors.

Debugging Techniques:

Incremental testing: In a good design for a complex module, the code is broken up into numerous subroutines, most of which are no more than 10 to 15 lines long. For a module designed in this way, incremental testing offers significant advantages. For incremental testing, the subroutines are classified in levels, with the lowest level subroutines being those that do not call other subroutines. If subroutine A calls subroutine B then A is a higher level subroutine than B. The incremental testing strategy is to test the subroutines individually, working from the lowest level to higher levels. To do testing at the lower levels, the test driver must either be capable of calling the low level subroutines directly, or else the programmer must be able to provide several test input cases, each of which only involves a small number of low level subroutines. Devising these test cases requires a thorough understanding of the module algorithms, along with a good imagination. The strength of incremental testing is that at any time in the process, there are only a small number of places where errors can arise. This automatically makes debugging information more meaningful and leads to quicker determination of the cause of an error. A second reason for incremental testing is that it greatly reduces the chances of having to deal with two or more errors at the same time. Multiple errors often will generate confusing error indications.

Sanity checks: Low level code in complex data structure is often written with the assumption that the higher level code correctly implements the desired algorithm. For example, the low level code may be written with the assumption that a certain variable or parameter cannot be NULL. Even if that assumption is justified by the algorithm, it may still be a good idea to put in a test to see if the condition is satisfied because the higher level code may be implemented incorrectly. This kind of check is called a sanity check. If an assert procedure is available then it can be used for the checks. The advantage of sanity checks is that they give early detection of errors.

Boolean constants for turning debugging code on or off: If debugging code is added to a module then it is often profitable to enclose it in an if statement that is controlled by a Boolean constant added to the module. By doing this, the debugging code can easily be turned off, yet be readily available if needed later. Different constants should be used for different stages of testing so that useless information is minimized.

Error variables for controlling program behavior after errors: When debugging print statements are added to code, there is the possibility of a tremendous explosion of useless information. The problem is that a print statement by itself will be executed whether or not there is an error. Thus, if the error does not appear until a large number of subroutine calls have been made then most of the messages are just telling you everything is okay so far. This problem is greatly magnified if the added code is displaying the internal structure of a data structure. Assuming that the module has sanity checks for error detection, an error Boolean variable can be added to the module. It should be initialized to false, indicating that there is no error. For most data structures, there is a Create operation for initialization. The error variable can be initialized at the same time. Instead of exiting the sanity checks are modified so that they set the error variable to true. Then debug code can be enclosed in if statements so that information is only printed when errors have been detected. One possible application of this method is obtaining traceback information when it is not otherwise available.

Traceback techniques: To obtain a traceback, use an error Boolean set by sanity checks. At various places in the module add debug code controlled by the error variable that prints the current position. Usually it is more economical to first run the code with a terminating sanity check. Then you only need to add the controlled debug code at places where the subroutine that contains the sanity check is called.

Correcting Code Errors: For the correction of errors detected by testing, the is one very important principle to keep in mind: **fix the cause, not the symptom**. Suppose that you run some code and get a segmentation fault. After some checking you determine that a NULL pointer was passed into a procedure that did not check for NULL, but tried to reference through the pointer anyway. Should you add a NULL pointer check to the procedure, enclosing the entire body of the procedure in an if statement? This question cannot be answered without an understanding of the design and algorithm. It may be that if the algorithm is correctly implemented then the pointer cannot be NULL, so the procedure does not make the check. If that is the case then adding the if statement does not fix the cause of the problem. Instead, it makes matters worse by covering up the symptoms. The problem will surely appear somewhere else, but now the symptoms will be further removed from the cause. Code such as the pointer NULL check should be added only if you are sure that it should be part of the algorithm. If you add a NULL pointer check that is not required by the algorithm then it should report an error condition. In other words, it should be a sanity check.

Verification and Validation

In few days back we have seen article about “V-Model”. In the V Model Software Development Life Cycle, based on requirement specification document the development & testing activity is started. The V-model is also called as **Verification and Validation** model. The testing activity is perform in the each phase of Software Testing Life Cycle. In the first half of the model validations testing activity is integrated in each phase like review user requirements, System Design document & in the next half the Verification testing activity is come in picture.

In interviews most of the interviewers are asking questions on “What is *Difference between Verification and Validation?”* Lots of people use verification and validation interchangeably but both have different meanings. So in this article I am adding few differences about Verification & Validation.

Difference between software **Verification and Validation is shown in Table 1.6.**

Table 1.6 Difference between software Verification and Validation

Verification	Validation
Are we building the system right?	Are we building the right system?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	**Validation** is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.

The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.
Following activities are involved in **Verification**: Reviews, Meetings and Inspections.	Following activities are involved in **Validation**: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check whether implementation software is as per specification document or not.	**Validation** is carried out by testing team.
Execution of code is not comes under **Verification**.	Execution of code is comes under **Validation**.
Verification process explains whether the outputs are according to inputs or not.	**Validation** process describes whether the software is accepted by the user or not.
Verification is carried out before the Validation.	**Validation** activity is carried out just after the Verification.
Following items are evaluated during **Verification**: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during **Validation**: Actual product or Software under test.
Cost of errors caught in**Verification** is less than errors found in Validation.	Cost of errors caught in**Validation** is more than errors found in Verification.
It is basically manually checking the of documents and files like requirement specifications etc.	It is basically checking of developed program based on the requirement specifications documents & files.

Conclusion on difference of ***Verification and Validation in software testing***:

1. Both Verification and Validation are essential and balancing to each other.
2. Different error filters are provided by each of them.
3. Both are used to finds a defect in different way, Verification is used to identify the errors in requirement specifications & validation is used to find the defects in the implemented Software application.

Test Levels

Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

1) **Unit testing**: A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately. The aim is to test each part of the software by separating it. It checks those components are fulfilling functionalities or not. This kind of testing is performed by developers.

2) **Integration testing:** Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing. Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

3) **System testing:** System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing. System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

4) **Acceptance testing:** Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

Other Types of Testing:

Alpha Testing Vs Beta testing:

Table 1.7 shows the differences between Alpha and Beta Testing:

Table 1.7 Alpha Testing Vs Beta testing

Alpha Testing	**Beta Testing**
Alpha testing performed by Testers who are usually internal employees of the organization	Beta testing is performed by Clients or End Users who are not employees of the organization
Alpha Testing performed at developer's site	Beta testing is performed at a client location or end user of the product
Reliability and Security Testing are not performed in-depth Alpha Testing	Reliability, Security, Robustness are checked during Beta Testing
Alpha testing involves both the white box and black box techniques	Beta Testing typically uses Black Box Testing
Alpha testing requires a lab environment or testing environment	Beta testing doesn't require any lab environment or testing environment. The software is made available to the public and is said to be real time environment
Long execution cycle may be required for Alpha testing	Only a few weeks of execution are required for Beta testing
Critical issues or fixes can be addressed by developers immediately in Alpha testing	Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product
Alpha testing is to ensure the quality of the product before moving to Beta testing	Beta testing also concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real time users.

Types of Beta Testing: There are different types of Beta tests in software testing, and they are as follows:

Traditional Beta testing: Product is distributed to the target market, and related data is gathered in all aspects. This data can be used for Product improvement.

Public Beta Testing: Product is publicly released to the outside world via online channels and data can be gathered from anyone. Based on feedback, product improvements can be done. For example, Microsoft conducted the largest of all Beta Tests for its OS -- Windows 8 before officially releasing it.

Technical Beta Testing: Product is released to the internal group of an organization and gathers feedback/data from the employees of the organization.

Focused Beta: Product is released to the market for gathering feedback on specific features of the program. For example, important functionality of the software.

Post-release Beta: Product is released to the market and data is gathered to make improvements for the future release of the product.

Software Quality?

Quality is meeting the requirement, expectation and needs of the customer being free from defects, lacks and substantial variants. There are standards needs to follow to satisfy the customer requirements. **Software Control/Quality Control** is known as QC and focuses on identifying defect. QC ensures that the approaches, techniques, methods and processes are designed in the project are following correctly. QC activities monitor and verify that the project deliverables meet the defined quality standards. Quality Control is a reactive process and is detection in nature. It recognizes the defects. Quality Control has to complete after Quality Assurance. **Control** is to test or verify actual results by comparing it with the defined standards.

Software Assurance?

Assurance is provided by organization management, it means giving a positive declaration on a product which obtains confidence for the outcome. It gives a security that the product will work without any glitches as per the expectations or requests. **Quality Assurance** is known as QA and focuses on preventing defect. Quality Assurance ensures that the approaches, techniques, methods and processes are designed for the projects are implemented correctly. Quality assurance activities monitor and verify that the processes used to manage and create the deliverables have been followed and are operative. Quality Assurance is a proactive process and is Prevention in nature. It recognizes flaws in the process. Quality Assurance has to complete before Quality Control.

Difference between quality assurance and quality control

Many people think QA and QC are same and interchangeable but this is not true. Both are tightly linked and sometimes it is very difficult to identify the differences. Fact is both are related to each other but they are different in origins. QA and QC both are part of Quality Management however QA is focusing on preventing defect while QC is focusing on identifying the defect. Parallel difference between these terms are shown in Table. 1.8

Table 1.8 Quality Assurance Vs Quality Control

Quality Assurance	**Quality Control**
It is a process which deliberate on providing assurance that quality request will be achieved.	QC is a process which deliberates on fulfilling the quality request.
A QA aim is to prevent the defect.	A QC aim is to identify and improve the defects.
QA is the technique of managing the quality.	QC is method to verify the quality.
QA does not involve executing the program.	QC always involves executing the program.
All team members are responsible for QA.	Testing team is responsible for QC.
QA e.g. Verification.	QC e.g. Validation.
QA means Planning for doing a process.	QC Means Action for executing the planned process.

Statistical Technique used on QA is known as Statistical Process Control (SPC.)	Statistical Technique used on QC is known as Statistical Quality Control (SPC.)
QA makes sure you are doing the right things.	QC makes sure the results of what you've done are what you expected.
QA Defines standards and methodologies to followed in order to meet the customer requirements.	QC ensures that the standards are followed while working on the product.
QA is the process to create the deliverables.	QC is the process to verify that deliverables.
QA is responsible for full software development life cycle.	QC is responsible for software testing life cycle.

Quality Assurance Analyst

Quality Analyst plays important role in software companies, from start to end quality analyst will be part of the team working on project. Quality analyst job during project lifespan:

- Assuring client requirements are well understood with clear documentation.
- Project initiation with assurance of quality resources that'll be needed throughout the project for achieving clients expectations.
- Cost Estimation & quality vigilance & auditing.
- Analysis of quality design & improvising of tools.
- Checking the coding quality is matching upto clients expectations & assuring the quality output of it.
- Removing defects & conducting external auditing.
- Once project is ready they check the feasibility on various aspect.
- After delivering getting a quality feedback & checking the quality of maintenance.
- Quality Analyst in Business analytics focusing companies (Not all but few companies like Mckinsey, AT Kearney,JP morgan ,BCG etc.) : Since in analytics focusing company you'll find every second employee as an analyst so quality analyst job won't be much differ from other team members (even this post doesn't exist in some companies). Most of the analytics companies don't give any kinda product but focuses on giving business solutions & business models to the clients. These companies will perform Quality analysis on 3 major aspects as follows :
- 1)Business Process Management
  2)Subject Matter expertise
  3)Information analytics.

Quality Factors

Flexibility and Extensibility: Flexibility is the ability of software to add/modify/remove functionality without damaging current system. Extensibility is the ability of software to add functionality without damaging system, so it may be thought as a subset of flexibility. Those functionality changes may occur according to changing requirements or an obligation if development process is one of the iterative methods. Change is inevitable in software development and so, this is one of the most important properties of quality software:

Maintainability and Readability: Maintainability is a little similar with flexibility but it focuses on modifications about error corrections and minor function modifications, not major functional extensibilities. It can be supported with useful interface definitions, documentations and also self-documenting code and/or code documentation. The more correct and useful documentation exists, the more maintainability can be performed.

Performance and Efficiency: Performance is mostly about response time of the software. This response time should be in acceptable intervals (e.g. max. a few seconds), and should not increase if transaction count increases. And also, resources are expensive. Efficiency must be supported with resource utilization. As an exaggerated example, ability of performing a simple function only by using a 32 processor machine or 1 TB disk space is not acceptable. Optimal source/performance ratio must be aimed.

Scalability: A scalable system responds user actions in an acceptable amount of time, even if load increases. Of course more hardware may be added for handling increasing user transaction, but the architecture should not change while doing this. This is called vertical scalability. Ability of running on multiple, increasing count of machines is multiple processing. If the software can perform that type of processing, this is called horizontal scalability. A preferred scalable system should suit both of these methods.

Availability, Robustness, Fault Tolerance and Reliability: A robust software should not lose its availability even in most failure states. Even if some components are broken down, it may continue running. Besides, even if whole application crashes, it may recover itself using backup hardware and data with fault tolerance approaches. There should always be B and even C, D .. plans. Reliability also stands for the integrity and consistency of the software even under high load conditions. So it is relevant with availability and scalability. An unreliable system is also unsalable.

Usability and Accessibility: User interfaces are the only visible parts of software according to the viewpoint of user. So, simplicity, taking less time to complete a job, fast learn ability etc. are very important in this case. The most well known principle for this property is KISS (Keep It Simple Stupid). Simple is always the best. A usable software should also support different accessibility types of control for people with disabilities.

Platform Compatibility and Portability: A quality software should run on as much various platforms as it can. So, more people can make use of it. In different contexts we may mention different platforms, this may be OS platforms, browser types etc. And portability is about adapting software that can run on different platforms, for being more platform compatible. In this sense, portability is also related with flexibility

Testability and Manageability: Quality software requires quality testing. Source code should be tested with the most coverage and with the most efficient testing methods. This can be performed by using encapsulation, interfaces, patterns, low coupling etc. techniques correctly. Besides testability, a qualified software should be manageable after deployment. It may be monitored for e.g. performance or data usage status, or may enable developer to configure system easily. Creating a successful logging system is another very important issue about manageability.

Security: Security is a very important issue on software development, especially for web or mobile based ones which may have millions of users with the ability of remote accessing to system. You should construct a security policy and apply it correctly by leaving no entry points. This may include authorization and authentication techniques, network attack protections, data encryption and so on. all possible types of security leaks should be considered, otherwise one day only one attack may crash your whole application and whole company.

Functionality and Correctness: Functionality (or correctness) is the conformity of the software with actual requirements and specifications. In fact this is the precondition attribute of an application, and maybe not a quality factor but we wanted to point that as the last quality factor, for taking attention: Quality factors are not meaningful when we are talking about unfunctional software. First, perform desired functionality and produce correct software, then apply quality factors on it. If you can perform both paralelly, it is the best.

Quality Management

Quality has been defined as "the degree to which something meets or exceeds the expectations of its consumers." The precise definition of quality can vary between industries and organizations. The processes and measurements used for quality assurance at a small manufacturing organization are entirely different from the checks and balances necessary for quality control in highly regulated industries such as pharmacy or medical devices. Quality is a dynamic concept which is ultimately defined by customer expectations and satisfaction. QMS are designed to provide a framework for organizations to create and maintain customer relationships by understanding the customer’s preferences and needs. Customer satisfaction is achieved with QMS through the alignment of people, process, and technology throughout the product lifecycle. One of the world’s most broadly adopted QMS, ISO 9001:2015, includes a series of quality principles which are frequently reflected in other QMS standards:

1. Customer focus
2. Leadership
3. Engagement of people
4. Process approach
5. Continuous improvement
6. Evidence-based decision making
7. Relationship management

Organizations must adopt an interdisciplinary series of quality controls to achieve these principles. The nine core elements of a QMS should include quality objectives, a quality manual, organizational responsibilities, data management, and other practices.

1. **Quality Objectives:** The creation of quality objectives is a common requirement of QMS standards, including ISO 9001. These objectives are designed to encourage organizations to define strategic goals and a purpose for the QMS. Objectives translate an organization's vision into practice by creating a link between customer requirements and specific, measurable, and attainable goals. Well-written objectives lend purpose to a QMS initiative and establish a customer-centric culture in an organization. A pharmaceutical startup in the research phase may have identified a customer need for affordable therapeutics to treat a common skin condition. Since the product is being developed, the organization may create a quality policy with a stated goal "To develop a safe, effective treatment for eczema patients which is available at a lower cost than alternatives."

Quality objectives for this organization could include:

- To obtain total compliance with staff training requirements and raise average assessment scores from 90% to 95%.
- To successfully implement a QMS software within three months and eliminate paper and spreadsheet-based record keeping methods within six months.
- To achieve a successful initial synthesis of the drug and complete all necessary processes for FDA initial review within 12 months.

Quality objectives should provide a clear vision for every member of the organization to understand the company's purpose and the value of a QMS. The objectives should provide a clear metric for measuring progress against strategic goals, including the timeline for achievement and a measurable parameter of improvement.

2. **Quality Manual:** A quality manual is defined as the first documentation of a QMS. It states the motivation for adopting a QMS framework and the role of quality within the organization. ISO 9000 requirements for a quality manual prescribe that this document should:

- Describe the scope of the QMS
- Detail the requirements of the QMS standard or framework
- List any elements of the QMS which are excluded from the implementation
- Reference specific quality procedures used within the organization
- Provide visual documentation of critical processes via flowchart
- Explain the organization's quality policies and objectives

3. **Organizational Structure and Responsibilities:** A QMS should include a clear and updated model of the organization's structure and responsibilities of all individuals within the organization. Documentation of structure and responsibilities should include visual guides such as flowcharts and clear documentation. Within the context of a QMS, the organization is broadly defined in World Health Organization guidance as both people and structure. For a life sciences company in the early phases of the product development lifecycle, initial efforts to identify organizational components may reveal a list similar to the following:

- Personnel
- Equipment
- Information Systems
- Tools for Assessment
- Facilities
- Purchasing & Inventory

- Process Controls
- Documents & Records

Documenting organizational structure should address the entire product lifecycle using techniques such as flowcharts which depict the "path of workflow." Defining responsibilities requires an organizational chart with clearly defined roles which can be linked to standard operating procedures (SOPs).

4. Data Management: Data is at the core of modern approaches to total quality management. Data quality and availability are critical to the success of a QMS framework to drive continuous improvement and preventative quality control activities. Organizations with ineffective data management practices can experience inconsistent product quality, operating inefficiencies, compliance risks, poor customer satisfaction, and low profitability.

An organization must be able to provide meaningful data evidence of effective quality controls. Data management systems should support continuous improvement efforts and corrective actions by defining the types of data that are gathered by the organization and third-party sources. The policy for data management should address data types, sources, collection methods, responsibilities, storage, disposal, and analysis. The types of data required to demonstrate effective QMS performance can vary significantly between organizations. However, at a minimum it should include the following sources:

- Customer Satisfaction
- Supplier Performance
- Product and Process Monitoring
- Non-Conformances
- Trends
- Preventative or Corrective Action

5. Processes: QMS are inherently process-driven approaches to quality control and assurance. Standards for quality management require organizations to identify and define all organizational processes which use any resource to transform inputs into outputs. Virtually every responsibility in the organization can be tied to a process, including purchasing. Initial efforts to define processes should create a high-level picture of how processes serve the organization and intersect with resources such as employees, machines, or technology. After identifying processes, organizations can begin to define standards and success metrics:

- Identify organizational processes
- Define process standards
- Establish methods for measuring success
- Document a standardized approach to ensuring quality output
- Drive continual improvement

6. Customer Satisfaction with Product Quality: A core component of QMS is the requirement for organizations to monitor customer satisfaction to determine if quality objectives are achieved. Some standards do not prescribe specific methods for measuring customer satisfaction since the definition of product quality and available data can vary significantly between organizations. A first step to establishing monitoring systems for customer satisfaction should be the definition of appropriate methods for measuring customer attitudes and complaints. This could include:

- Satisfaction Surveys
- Complaints Procedures
- Analytical Applications to measure satisfaction trends
- Management Review of customer satisfaction

7. Continuous Improvement: Continuous improvement and adaptation are necessary for organizations to drive benefits with the QMS and maintain customer satisfaction. QMS dictate that continual improvement is an organization-wide responsibility. However, ISO 9001 is clear that leadership should play a core role in implementing a quality-driven culture. Clause 5.1.1 states "top management shall demonstrate leadership and commitment with respect to the quality management system by taking accountability for effectiveness." Designing organizational processes to meet QMS standards for continuous improvement requires clear documentation of controls across the organization. Improvement documentation should encompass, at a minimum:

- Quality Planning Procedures
- Compliance Requirements
- Safety Design
- Risk-based Thinking
- Corrective Action (CAPA)
- Gradual and Breakthrough Improvement
- Innovation
- Assessment of the QMS

8. Quality Instruments: The control and calibration of tools used to measure quality are integral to the success of a QMS. If machines or equipment are used to validate products or processes, this equipment must be carefully controlled and calibrated according to industry standards. Depending on the instrument, this could involve periodic calibrations or calibration before every measurement. The QMS system design within an organization should dictate a clear policy for the maintenance of quality instruments based on nationally or internationally recognized standards for each piece of quality equipment. This documentation should address:

- Intervals for instrument calibration
- Recognized Standards for instrument calibration
- Manufacturer Instructions for adjustment
- Procedures for identifying and documenting calibration
- Controls against tampering or adjustment post-calibration
- Methods to protect instruments and equipment from damage

In addition to these requirements, the QMS should address effective documentation of calibration results, including procedures for maintaining complete records of activities and calibration results.

9. Document Control: The definition of a document in a quality-driven organization is broad, according to ISO. It includes all records of:

- Communications
- Evidence
- QMS Conformity
- Knowledge Sharing

QMS dictate standards for the types of documentation which are necessary to support quality management at a minimum, which may not be reflective of all the documents needed for accurate quality control. This generally includes quality objectives, a quality manual, procedures, process documentation, and records keeping. Document management systems must contain all evidence necessary to prove QMS performance objectively.

Methods of Quality Management

Quality improvement methods comprise three components: product improvement, process improvement, and people-based improvement. There are numerous methods of quality management and techniques that can be utilized. They include Kaizen, Zero Defect Programs, Six Sigma, Quality Circle, Taguchi Methods, the Toyota Production System, Kansei Engineering, TRIZ, BPR, OQRM, ISO, and Top Down & Bottom Up approaches among others.

Quality Management – Example :A model example of great quality management is the implementation of the Kanban system by Toyota Corporation. Kanban is an inventory control system that was developed by Taiichi Ohno to create visibility for both the suppliers and buyers to help limit the upsurge of excess inventory on the production line at any given point in time. Toyota used the concept to execute its Just-in-Time (JIT) system, which helps align raw material orders from suppliers directly with the production schedules. Toyota’s assembly line increased efficiency aa the company received just enough inventories on hand to meet customer orders as they were being generated.

Principles of Quality Management: There are several principles of quality management that the International Standard for Quality Management adopts. These principles are used by top management to guide an organization's processes towards improved performance. They include:

1. Customer Focus: The primary focus of any organization should be to meet and exceed the customers' expectations and needs. When an organization can understand the customers' current and future needs and cater to them, that results in customer loyalty, which in turn increases revenue. The business is also able to identify new customer opportunities and satisfy them. When business processes are more efficient, quality is higher and more customers can be satisfied.

2. Leadership: Good leadership results in an organization's success. Great leadership establishes unity and purpose among the workforce and shareholders. Creating a thriving company culture provides an internal environment that allows employees to fully realize their potential and get actively involved in achieving company objectives. Leaders should involve the employees in setting clear organizational goals and objectives. This motivates employees, who may significantly improve their productivity and loyalty.

3. Engagement of People: Staff involvement is another fundamental principle. The management engages staff in creating and delivering value whether they are full-time, part-time, outsourced, or in-house. An organization should encourage the employees to constantly improve their skills and maintain consistency. This principle also involves empowering the employees, involving them in decision making and recognizing their achievements. When people are valued, they work to their best potential because it boosts their confidence and motivation. When employees are wholly involved, it makes them feel empowered and accountable for their actions.

4. Process Approach: The performance of an organization is crucial according to the process approach principle. The approach principle emphasizes achieving efficiency and effectiveness in the organizational processes. The approach entails an understanding that good processes result in improved consistency, quicker activities, reduced costs, waste removal, and continuous improvement. An organization is enhanced when leaders can manage and control the inputs and the outputs of an organization, as well as the processes used to produce the outputs.

5. Continuous Improvement: Every organization should come up with an objective to be actively involved in continuous improvement. Businesses that improve continually experience improved performance, organizational flexibility, and increased ability to embrace new opportunities. Businesses should be able to create new processes continually and adapt to new market situations.

6. Evidence-based Decision Making: Businesses should adopt a factual approach to decision-making. Businesses that make decisions based on verified and analyzed data have an improved understanding of the marketplace. They are able to perform tasks that produce desired results and justify their past decisions. Factual decision making is vital to help understand the cause-and-effect relationships of different things and explain potential unintended results and consequences.

7. Relationship Management: Relationship management is about creating mutually beneficial relations with suppliers and retailers. Different interested parties can impact a company's performance. The organization should manage the supply chain process well and promote the relationship between the organization and its suppliers to optimize their impact on the company's performance. When an organization manages its relationship with interested parties well, it is more likely to achieve sustained business collaboration and success.

Core Components of Quality

The core elements of a QMS should include quality objectives, a quality manual, organizational responsibilities, data management, and other practices.

1. Quality Objectives: The creation of quality objectives is a common requirement of QMS standards, including ISO 9001. These objectives are designed to encourage organizations to define strategic goals and a purpose for the QMS. Objectives translate an organization's vision into practice by creating a link between customer requirements and specific, measurable, and attainable goals. Well-written objectives lend purpose to a

QMS initiative and establish a customer-centric culture in an organization. A pharmaceutical startup in the research phase may have identified a customer need for affordable therapeutics to treat a common skin condition. Since the product is being developed, the organization may create a quality policy with a stated goal "To develop a safe, effective treatment for eczema patients which is available at a lower cost than alternatives."
Quality objectives for this organization could include:

- To obtain total compliance with staff training requirements and raise average assessment scores from 90% to 95%.
- To successfully implement a QMS software within three months and eliminate paper and spreadsheet-based record keeping methods within six months.
- To achieve a successful initial synthesis of the drug and complete all necessary processes for FDA initial review within 12 months.

Quality objectives should provide a clear vision for every member of the organization to understand the company's purpose and the value of a QMS. The objectives should provide a clear metric for measuring progress against strategic goals, including the timeline for achievement and a measurable parameter of improvement.

2. Quality Manual: A quality manual is defined as the first documentation of a QMS. It states the motivation for adopting a QMS framework and the role of quality within the organization. ISO 9000 requirements for a quality manual prescribe that this document should:

- Describe the scope of the QMS
- Detail the requirements of the QMS standard or framework
- List any elements of the QMS which are excluded from the implementation
- Reference specific quality procedures used within the organization
- Provide visual documentation of critical processes via flowchart
- Explain the organization’s quality policies and objectives

3. Organizational Structure and Responsibilities: A QMS should include a clear and updated model of the organization's structure and responsibilities of all individuals within the organization. Documentation of structure and responsibilities should include visual guides such as flowcharts and clear documentation. Within the context of a QMS, the organization is broadly defined in World Health Organization guidance as both people and structure. For a life sciences company in the early phases of the product development lifecycle, initial efforts to identify organizational components may reveal a list similar to the following:

- Personnel
- Equipment
- Information Systems
- Tools for Assessment
- Facilities
- Purchasing & Inventory
- Process Controls
- Documents & Records

Documenting organizational structure should address the entire product lifecycle using techniques such as flowcharts which depict the “path of workflow.” Defining responsibilities requires an organizational chart with clearly defined roles which can be linked to standard operating procedures (SOPs).

4. Data Management: Data is at the core of modern approaches to total quality management. Data quality and availability are critical to the success of a QMS framework to drive continuous improvement and preventative quality control activities. Organizations with ineffective data management practices can experience inconsistent product quality, operating inefficiencies, compliance risks, poor customer satisfaction, and low profitability. An organization must be able to provide meaningful data evidence of effective quality controls. Data management systems should support continuous improvement efforts and corrective actions by defining the types of data that are gathered by the organization and third-party sources. The policy for data management should address data types, sources, collection methods, responsibilities, storage, disposal, and analysis.
The types of data required to demonstrate effective QMS performance can vary significantly between organizations. However, at a minimum it should include the following sources:

- Customer Satisfaction
- Supplier Performance
- Product and Process Monitoring
- Non-Conformances
- Trends
- Preventative or Corrective Action

5. Processes: QMS are inherently process-driven approaches to quality control and assurance. Standards for quality management require organizations to identify and define all organizational processes which use any resource to transform inputs into outputs. Virtually every responsibility in the organization can be tied to a process, including purchasing. Initial efforts to define processes should create a high-level picture of how processes serve the organization and intersect with resources such as employees, machines, or technology. After identifying processes, organizations can begin to define standards and success metrics:

- Identify organizational processes
- Define process standards
- Establish methods for measuring success
- Document a standardized approach to ensuring quality output
- Drive continual improvement

6. Customer Satisfaction with Product Quality: A core component of QMS is the requirement for organizations to monitor customer satisfaction to determine if quality objectives are achieved. Some standards do not prescribe specific methods for measuring customer satisfaction since the definition of product quality and available data can vary significantly between organizations.

A first step to establishing monitoring systems for customer satisfaction should be the definition of appropriate methods for measuring customer attitudes and complaints. This could include:

- Satisfaction Surveys
- Complaints Procedures
- Analytical Applications to measure satisfaction trends
- Management Review of customer satisfaction

7. Continuous Improvement: Continuous improvement and adaptation are necessary for organizations to drive benefits with the QMS and maintain customer satisfaction. QMS dictate that continual improvement is an organization-wide responsibility. However, ISO 9001 is clear that leadership should play a core role in implementing a quality-driven culture. Clause 5.1.1 states "top management shall demonstrate leadership and commitment with respect to the quality management system by taking accountability for effectiveness."

Designing organizational processes to meet QMS standards for continuous improvement requires clear documentation of controls across the organization. Improvement documentation should encompass, at a minimum:

- Quality Planning Procedures
- Compliance Requirements
- Safety Design
- Risk-based Thinking
- Corrective Action (CAPA)
- Gradual and Breakthrough Improvement
- Innovation
- Assessment of the QMS

8. Quality Instruments: The control and calibration of tools used to measure quality are integral to the success of a QMS. If machines or equipment are used to validate products or processes, this equipment must be carefully controlled and calibrated according to industry standards. Depending on the instrument, this could involve periodic calibrations or calibration before every measurement. The QMS system design within an organization should dictate a clear policy for the maintenance of quality instruments based on nationally or internationally recognized standards for each piece of quality equipment.

This documentation should address:

- Intervals for instrument calibration
- Recognized Standards for instrument calibration
- Manufacturer Instructions for adjustment
- Procedures for identifying and documenting calibration
- Controls against tampering or adjustment post-calibration
- Methods to protect instruments and equipment from damage

In addition to these requirements, the QMS should address effective documentation of calibration results, including procedures for maintaining complete records of activities and calibration results.

9. Document Control: The definition of a document in a quality-driven organization is broad, according to ISO. It includes all records of:

- Communications
- Evidence
- QMS Conformity
- Knowledge Sharing

QMS dictate standards for the types of documentation which are necessary to support quality management at a minimum, which may not be reflective of all the documents needed for accurate quality control. This generally includes quality objectives, a quality manual, procedures, process documentation, and records keeping. Document management systems must contain all evidence necessary to prove QMS performance objectively.

Effective records-keeping is crucial to the success of the QMS, the ability to obtain certification with QMS standards, and for regulatory compliance. During QMS design, organizations should create specific definitions of records within the organization and policies for document creation, retention, and editing. While QMS standards do not typically prescribe a method for document management, being able to capture and retain all supporting evidence is generally best accomplished with Quality Management Systems software.

Benefits of Quality Management:

- It helps an organization achieve greater consistency in tasks and activities that are involved in the production of products and services.
- It increases efficiency in processes, reduces wastage, and improves the use of time and other resources.
- It helps improve customer satisfaction.
- It enables businesses to market their business effectively and exploit new markets.
- It makes it easier for businesses to integrate new employees, and thus helps businesses manage growth more seamlessly.
- It enables a business to continuously improve their products, processes, and systems.

Cost Aspect of Quality

Cost of quality (COQ) is defined as a methodology that allows an organization to determine the extent to which its resources are used for activities that prevent poor quality, that appraise the quality of the organization’s products or services, and that result from internal and external failures. Having such information allows an organization to determine the potential savings to be gained by implementing process improvements.

- Cost of poor quality (COPQ)
- Appraisal costs
- Internal failure costs
- External failure costs
- Prevention costs
- COQ and organizational objectives
- COQ resources

What is cost of poor quality ? Cost of poor quality (COPQ) is defined as the costs associated with providing poor quality products or services. There are three categories:

1. Appraisal costs are costs incurred to determine the degree of conformance to quality requirements.
2. Internal failure costs are costs associated with defects found before the customer receives the product or service.

3. External failure costs are costs associated with defects found after the customer receives the product or service.

Quality-related activities that incur costs may be divided into prevention costs, appraisal costs, and internal and external failure costs.

Appraisal costs: Appraisal costs are associated with measuring and monitoring activities related to quality. These costs are associated with the suppliers' and customers' evaluation of purchased materials, processes, products, and services to ensure that they conform to specifications. They could include:

- Verification: Checking of incoming material, process setup, and products against agreed specifications
- Quality audits: Confirmation that the quality system is functioning correctly
- Supplier rating: Assessment and approval of suppliers of products and services

Internal failure costs: Internal failure costs are incurred to remedy defects discovered before the product or service is delivered to the customer. These costs occur when the results of work fail to reach design quality standards and are detected before they are transferred to the customer. They could include:

- Waste: Performance of unnecessary work or holding of stock as a result of errors, poor organization, or communication
- Scrap: Defective product or material that cannot be repaired, used, or sold
- Rework or rectification: Correction of defective material or errors
- Failure analysis: Activity required to establish the causes of internal product or service failure

External failure costs: External failure costs are incurred to remedy defects discovered by customers. These costs occur when products or services that fail to reach design quality standards are not detected until after transfer to the customer. They could include:

- Repairs and servicing: Of both returned products and those in the field
- Warranty claims: Failed products that are replaced or services that are re-performed under a guarantee
- Complaints: All work and costs associated with handling and servicing customers' complaints
- Returns: Handling and investigation of rejected or recalled products, including transport costs

Prevention Costs: Prevention costs are incurred to prevent or avoid quality problems. These costs are associated with the design, implementation, and maintenance of the quality management system. They are planned and incurred before actual operation, and they could include:

- Product or service requirements: Establishment of specifications for incoming materials, processes, finished products, and services
- Quality planning: Creation of plans for quality, reliability, operations, production, and inspection
- Quality assurance: Creation and maintenance of the quality system
- Training: Development, preparation, and maintenance of programs

Cost of Quality and Organizational Objectives : The costs of doing a quality job, conducting quality improvements, and achieving goals must be carefully managed so that the long-term effect of quality on the organization is a desirable one. These costs must be a true measure of the quality effort, and they are best determined from an analysis of the costs of quality. Such an analysis provides a method of assessing the effectiveness of the management of quality and a means of determining problem areas, opportunities, savings, and action priorities. Cost of quality is also an important communication tool. Philip Crosby demonstrated what a powerful tool it could be to raise awareness of the importance of quality. He referred to the measure as the "price of nonconformance" and argued that organizations choose to pay for poor quality. Many organizations will have true quality-related costs as high as 15-20% of sales revenue, some going as high as 40% of total operations. A general rule of thumb is that costs of poor quality in a thriving company will be about 10-15% of operations. Effective quality improvement programs can reduce this substantially, thus making a direct contribution to profits.

The quality cost system, once established, should become dynamic and have a positive impact on the achievement of the organization's mission, goals, and objectives.

Cost of Quality Resources

Using Cost of Quality to Improve Business Results (PDF) Since centering improvement efforts on cost of quality, CRC Industries has reduced failure dollars as a percentage of sales and saved hundreds of thousands of dollars.

Cost of Quality: Why More Organizations Do Not Use It Effectively (World Conference on Quality and Improvement) Quality managers in organizations that do not track cost of quality cite as reasons a lack of management support for quality control, time and cost of COQ tracking, lack of knowledge of how to track data, and lack of basic cost data.

Cost of Quality (COQ): Which Collection System Should Be Used? (World Conference on Quality and Improvement) This article identifies the various COQ systems available and the benefits and disadvantages of using each system.