All Online Learning
www.allonlinelearning.com

Unit 4

Lecture 1. Testing Objectives, Testing Principles, Test Plans, Test Cases, Types of Testing, Levels of Testing, Test Strategies

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

- It can also be stated as the **process of validating and verifying** that a software program or application or product:
  - Meets the business and technical requirements that guided it's design and development
  - Works as expected
  - Can be implemented with the same characteristic.

Let's break the definition of **Software testing** into the following parts:

1) Process: Testing is a process rather than a single activity.

2) All Life Cycle Activities: Testing is a process that's take place throughout the Software Development Life Cycle (SDLC).

- The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it's referred as **"verifying the test basis via the test design"**.
- The **test basis** includes documents such as the requirements and design specifications.

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

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

5) Planning: We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.

6) Preparation: We need to choose what testing we will do, by selecting test conditions and designing test cases.

7) Evaluation: During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.

8) Software products and related work products: Along with the testing of code the testing of requirement and design specifications and also the related documents like operation, user and training material is equally important.

Why software testing needed?

Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.

Since we assume that our work may have mistakes, hence we all need to check our own work. However some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.

Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

There are several reasons which clearly tell us as why Software Testing is important and what are the major things that we should consider while testing of any product or application.

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.
2. It's essential since it makes sure of the Customer's reliability and their satisfaction in the application.
3. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence.
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
7. It's required to stay in the business.

Testing Objectives:

What are software testing objectives and purpose?

Software Testing has different goals and objectives. The major objectives of Software testing are as follows:

- Finding defects which may get created by the programmer while developing the software.

- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it's performing well and as per the specifications.

While determining the coverage the test cases should be designed well with maximum possibilities of finding the errors or bugs. The test cases should be very effective. This objective can be measured by the number of defects reported per test cases. Higher the number of the defects reported the more effective are the test cases.

Once the delivery is made to the end users or the customers they should be able to operate it without any complaints. In order to make this happen the tester should know as how the customers are going to use this product and accordingly they should write down the test scenarios and design the test cases. This will help a lot in fulfilling all the customer's requirements.

Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use.

Thus we have categorized as-

Three main objectives of software testing are –

1. **Verification.** Verification confirms that software meets its technical specifications i. e. the technical definition of function in terms measurable output value.
2. **Validation**. Validation confirms that the software meets the business requirements. i. e. provides details on how data will be summarized and displayed.
3. **Defect finding**. It is variance between actual and expected output.

Testing Principles:

Principle 1 A necessary part of the test case is a definition of the expected output of result.

Test case consists of two components:

–Description of the initial data of the program.

–A brief description of the correct output of the program for that set of input data.

Principle 2 A programmer should avoid attempting to test his or her own program.

Due to psychological issues, when a programmer has designed and coded the program, it is hard to sudden change in the program.

Principle 3 A programming organization should not test its own programs.

Principle 4 Thoroughly inspect the result of each test.

Due to errors found on later test are often missed in the result of earlier test.

Principle 5 Test cases must be written for input conditions.

Principle 6 Examine a program to see if it does not do what it is supposed to do and whether a program does what is not supposed to do.

The output of the program must halt on every input with the correct output. This is possible in two ways-

* The programmer must understand what requirement of the programmer is and code has been written according user per respective.

*The code must not generate any unexpected results.

Principle 7 Avoid through away test cases unless a program is truly through away program.

Principle 8 Do not plan testing effort under tacit assumption that no error will be found.

Program testing should not be defined as program function correctly rather it Will be defined as intend of finding error.

Principle 9 The probability of existence of more errors in a section is proportional to no errors found in that section.

Principle 10 Testing is extremely crating and intellectually challenging task.

It is probably true that creativity required in testing a large program exceeds the creativity in designing program. For testing, develop a reasonable set of test cases for a program but these methodologies require a significant amount of creativity.

We can summaries Principles of Testing as:

There are seven principles of testing. They are as follows:

1) Testing shows presence of defects: Testing can show the defects are present, but cannot prove that there are no defects. Even after testing the application or product thoroughly we cannot say that the product is 100% defect free. Testing always reduces the number of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

2) Exhaustive testing is impossible: Testing everything including all combinations of inputs and preconditions is not possible. So, instead of doing the exhaustive testing we can use risks and priorities to focus testing efforts. For example: In an application in one screen there are 15 input fields, each having 5 possible values, then to test all the valid combinations you would need 30 517 578 125 (5^{15}) tests. This is very unlikely that the project timescales would allow for this number of tests. So, accessing and managing risk is one of the most important activities and reason for testing in any project.

3) Early testing: In the software development life cycle testing activities should start as early as possible and should be focused on defined objectives.

4) Defect clustering: A small number of modules contains most of the defects discovered during pre-release testing or shows the most operational failures.

5) Pesticide paradox: If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. To overcome this “Pesticide Paradox”, it is really very important to review the test cases regularly and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

6) Testing is context depending: Testing is basically context dependent. Different kinds of sites are tested differently. For example, safety – critical software is tested differently from an e-commerce site.

7) Absence – of – errors fallacy: If the system built is unusable and does not fulfill the user’s needs and expectations then finding and fixing defects does not help.

Software test plan:

A test plan is a document describing the scope, approach, objectives, resources, and schedule of a software testing effort. It identifies the items to be tested, items not be tested, who will do the testing, the test approach followed, what will be the pass/fail criteria, training needs for team, the testing schedule etc

IEEE 829 test plan structure

IEEE 829-2008, also known as the 829 Standard for Software Test Documentation, is an IEEE standard that specifies the form of a set of documents for use in defined stages of software testing, each stage potentially producing its own separate type of document.

1.Test plan identifier
2. Introduction
3. Test items
4. Features to be tested
5. Features not to be tested
6. Approach
7. Item pass/fail criteria
8. Suspension criteria and resumption requirements
9. Test deliverables
10. Testing tasks
11. Environmental needs
12. Responsibilities
13. Staffing and training needs
14. Schedule
15. Risks and contingencies
16. Approvals

How to write test plan?

1. Writing a good test plan is very easy, you just need to have an organized approach.
2. You need to understand the purpose of testing before you proceed with the Test Plan. The most important things that should be kept in mind before writing test plan are:
3. **1. What is in scope of Testing.**
   2. What is out of scope of Testing.
   3. What are the test objectives.
   4. What are the budget limitations.
   5. What are the project deadlines.
   6. What is the test execution schedule.
   7. What are the project risks.
   8. What are the product risks.
4. Based on the factors mentioned above you should select test strategy and decide how to split test work into various test levels. You should try to reduce gap between levels, master test plan addresses the details which deal with inter-level co-ordination.
5. Another thing that should be remembered while writing a good test plan is that you should always plan to co-ordinate your testing work with rest of the project activities, like what are the test dependencies which include hardware availability, software availability etc. and the test deliverables required after the testing is complete.
6. A good test plan should specifically mention who will do what, when and how.

7. You should also clearly mention what will be the test deliverables, how precisely should the testers write the test cases, test design etc.
8. Test plan should also contain entry criteria and exit criteria, i.e. When can you enter the test level or phase and when can you exit from test phase.

Now we can summarize as-

The test plan is a mandatory document. You cannot test without one. According to "ANSI and IEEE standards for software Test Documentation", the following components should be covered in the test plan as given in table below-

Component	Description	Purpose
Responsibilities	**Specific people and their assignments**	**Assigns responsibilities and keeps everyone on track and focused.**
Assumptions	**Code, system status and availability**	**Avoids misunderstanding about schedules**
Test	**Testing scope, schedule, duration and prioritization.**	**Outlines the entire process and maps specific tested**
Communication	**Communication Plans who, what, when, how**	**Everyone knows what they need to know when they need to know it**
Risk Analysis	**Critical terms that will be tested**	**Provides focus by identifying areas critical for success.**
Defect Reporting	**How defects will be logged and documented**	**Tells how to document a defect so that it can be reproduced, fixed and retested**
Environment	**The Technical Environment, data work are and interfaces in testing**	**Reduces or eliminate misunderstandings and source of potential delay**

Test cases:

A test case is a documentation which specifies input values, expected output and the preconditions for executing the test.

It is well-documented procedure to test the functionality of a feature in the system. It means, a test case is document that describes an input action or event and expected result, in order to determine if a feature of an applications working correctly. A test case should contain particulars as test cases identifier, name, setup, objectives etc.

Test Case ID Identification number given to each test case.

Purpose Defines why the case is being designated.

Test Case ID,

Purpose,

Preconditions,

Inputs,

Preconditions. Preconditions for running the inputs in system can be defined, if required in a test case.

Inputs. Inputs should not be hypothetical. Actual inputs must be provided.

Expected Output. These are the outputs which should be produced when there is no system failure.

IEEE Standard 610 (1990) defines test case as follows:

A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

Boris Beizer (1995, p. 3) defines a test as:

A sequence of one or more subtests executed as a sequence because the outcome and/or final state of one subtest is the input and/or initial state of the next. The word 'test' is used to include subtests, tests proper, and test suites.

Objectives behind writing and executing the test cases:

Below mentioned are some of the objectives behind running the test cases.

1. Find the defects in software products
2. Verify that the software meets the end user requirements
3. Improve software quality
4. Minimize the maintenance and software support costs
5. Avoid post deployment risks
6. Compliance with processes
7. Help management to make software delivery decisions

Types of Testing

1. **White Box Testing**. Testing is based on the internal structure of the piece of software. It is also know as structural testing/glass-box testing/clean-box testing/internal testing.
2. **Black Box Testing**. Testing is based on the analysis of the specification of a piece of software without reference to its internal workings. The goal is to test how well the components conform to its published requirements for components.
3. **Unit Testing**. It is first level of dynamic testing. It confirms the behaviors of a single module are according its functional specifications.

4. **Acceptance Testing**. Testing is conducted to enable a user/customer to determine whether to accept software product. It is normally performed to validate the software meets a set of agreed acceptance criteria
5. **Integration Testing**. Upon completion of unit testing, integration testing begins, which black-box is testing. The purpose is to ensure that distinct components of application still work in accordance to customer requirements.
6. **Regression Testing**. The objective of regression testing is to ensure software remains intact. A baseline set of data and scripts will be maintained and executed to verify changes introduced during release have not "undone" previous code.
7. **System Testing**. Upon completion of integration testing, the test team will begin system testing. This testing focuses upon entire integrated system. The purpose is to test the validity for specific users and environments. The validity of whole system is checked against user requirements.
8. **Installation Testing.** Installation testing confirms that the application under test recovers from expected or unexpected or unexpected events without loss of data or functionality. Events can include storage of disk space, unexpected loss of communication, or power out conditions.
9. **Alpha Testing**. Alpha testing of an application is done when development almost on completion. Minor design changes may be introduced as a result of such testing. It is performed by end users and others but not by testers.
10. **Beta Testing.** Beta testing is done when development and testing is essentially completed and final bugs and problems need to be found before final release, it is also performed by end users.

Level of testing:

Testing levels are basically to identify missing areas and prevent overlap and repetition between the development life cycle phases. In software development life cycle models there are defined phases like requirement gathering and analysis, design, coding or implementation, testing and deployment. Each phase goes through the testing. Hence there are various levels of testing. The various levels of testing are:

1. **Unit testing:** It is basically done by the developers to make sure that their code is working fine and meet the user specifications. They test their piece of code which they have written like classes, functions, interfaces and procedures.
2. **Component testing:** It is also called as module testing. The basic difference between the unit testing and component testing is in unit testing the developers test their piece of code but in component testing the whole component is tested. For example, in a student record application there are two modules one which will save the records of the students and other module is to upload the results of the students. Both the modules are developed separately and when they are tested one by one then we call this as a component or module testing.
3. **Integration testing:** Integration testing is done when two modules are integrated, in order to test the behavior and functionality of both the modules after integration. Below are few types of integration testing:
    - Big bang integration testing
    - Top down
    - Bottom up
    - Functional incremental
4. **Component integration testing:** In the example above when both the modules and components are integrated then the testing done is called as Component integration testing. This testing is basically done to ensure that the code should not break after integrating the two modules.
5. **System integration testing:** System integration testing (SIT) is a testing where testers basically test that in the same environment all the related systems should maintain data integrity and can operate in coordination with other systems.
6. **System testing:** In system testing the testers basically test the compatibility of the application with the system.
7. **Acceptance testing:** Acceptance testing are basically done to ensure that the requirements of the specification are met.
8. **Alpha testing:** Alpha testing is done at the developer's site. It is done at the end of the development process
9. **Beta testing:** Beta testing is done at the customers site. It is done just before the launch of the product.

Test Strategies:

The choice of **test approaches** or **test strategy** is one of the most powerful factor in the success of the test effort and the accuracy of the test plans and estimates. This factor is under the control of the testers and test leaders.

Let's survey the major types of test strategies that are commonly found:

- **Analytical:** Let us take an example to understand this. The risk-based strategy involves performing a risk analysis using project documents and stakeholder input, then planning, estimating, designing, and prioritizing the tests based on risk. Another analytical test strategy is the requirements-based strategy, where an analysis of the requirements specification forms the basis for planning, estimating and designing tests. Analytical test strategies have in common the use of some formal or informal analytical technique, usually during the requirements and design stages of the project.

- **Model-based:** Let us take an example to understand this. You can build mathematical models for loading and response for e commerce servers, and test based on that model. If the behavior of the system under test conforms to that predicted by the model, the system is deemed to be working. Model-based test strategies have in common the creation or selection of some formal or informal model for critical system behaviors, usually during the requirements and design stages of the project.

- **Methodical:** Let us take an example to understand this. You might have a checklist that you have put together over the years that suggests the major areas of testing to run or you might follow an industry-standard for software quality, such as ISO 9126, for your outline of major test areas. You then methodically design, implement and execute tests following this outline. Methodical test strategies have in common the adherence to a pre-planned, systematized approach that has been developed in-house, assembled from various concepts developed inhouse and gathered from outside, or adapted significantly from outside ideas and may have an early or late point of involvement for testing.

- **Process – or standard-compliant:** Let us take an example to understand this. You might adopt the IEEE 829 standard for your testing, using books such as [Craig, 2002] or [Drabick, 2004] to fill in the methodological gaps. Alternatively, you might adopt one of the agile methodologies such as Extreme Programming. Process- or standard-compliant strategies have in common reliance upon an externally developed approach to testing, often with little – if any – customization and may have an early or late point of involvement for testing.

- **Dynamic:** Let us take an example to understand this. You might create a lightweight set of testing guide lines that focus on rapid adaptation or known weaknesses in software. Dynamic strategies, such as **exploratory testing,** have in common concentrating on finding as many defects as possible during test execution and adapting to the realities of the system under test as it is when delivered, and they typically emphasize the later stages of testing. See, for example, the attack based approach of [Whittaker, 2002] and [Whittaker, 2003] and the exploratory approach of [Kaner *et al.,* 2002].

- **Consultative or directed:** Let us take an example to understand this. You might ask the users or developers of the system to tell you what to test or even rely on them to do the testing. Consultative or directed strategies have in common the reliance on a group of non-testers to guide or perform the testing effort and typically emphasize the later stages of testing simply due to the lack of recognition of the value of early testing.

- **Regression-averse:** Let us take an example to understand this. You might try to automate all the tests of system functionality so that, whenever anything changes, you can re-run every test to ensure nothing has broken. Regression-averse strategies have in common a set of procedures – usually automated – that allow them to detect regression defects. A regression-averse strategy may involve automating functional tests prior to release of the function, in which case it requires early testing, but sometimes the testing is almost entirely focused on testing functions that already have been released, which is in some sense a form of post release test involvement.

Some of these strategies are more preventive, others more reactive. For example, analytical test strategies involve upfront analysis of the test basis, and tend to identify problems in the test basis prior to test execution. This allows the early – and cheap – removal of defects. That is a strength of preventive approaches.

Dynamic test strategies focus on the test execution period. Such strategies allow the location of defects and defect clusters that might have been hard to anticipate until you have the actual system in front of you. That is a strength of reactive approaches.

Rather than see the choice of strategies, particularly the preventive or reactive strategies, as an either/or situation, we'll let you in on the worst-kept secret of testing (and many other disciplines): There is no one best way. We suggest that you adopt whatever test approaches make the most sense in your particular situation, and feel free to borrow and blend.

How do you know which strategies to pick or blend for the best chance of success? There are many factors to consider, but let us highlight a few of the most important:

- **Risks:** Risk management is very important during testing, so consider the risks and the level of risk. For a well-established application that is evolving slowly, regression is an important risk, so regression-averse strategies make sense. For a new application, a risk analysis may reveal different risks if you pick a risk-based analytical strategy.
- **Skills:** Consider which skills your testers possess and lack because strategies must not only be chosen, they must also be executed. . A standard compliant strategy is a smart choice when you lack the time and skills in your team to create your own approach.
- **Objectives:** Testing must satisfy the needs and requirements of stakeholders to be successful. If the objective is to find as many defects as possible with a minimal amount of up-front time and effort invested – for example, at a typical independent test lab – then a dynamic strategy makes sense.
- **Regulations:** Sometimes you must satisfy not only stakeholders, but also regulators. In this case, you may need to plan a methodical test strategy that satisfies these regulators that you have met all their requirements.
- **Product:** Some products like, weapons systems and contract-development software tend to have well-specified requirements. This leads to synergy with a requirements-based analytical strategy.
- **Business:** Business considerations and business continuity are often important. If you can use a legacy system as a model for a new system, you can use a model-based strategy.

You must choose testing strategies with an eye towards the factors mentioned earlier, the schedule, budget, and feature constraints of the project and the realities of the organization and its politics.

We mentioned above that a good team can sometimes triumph over a situation where materials, process and delaying factors are ranged against its success. However, talented execution of an unwise strategy is the equivalent of going very fast down a highway in the wrong direction. Therefore, you must make smart choices in terms of testing strategies.

Lecture 2: Program Correctness, Program Verification & validation, Testing Automation & Testing Tools,

Establishing Program Correctness

Correctness involves

- establishing correctness of individual modules
- establishing correctness of system subsection
- establishing correctness of complete system

Usually integrated with coding and other activities

- at any time some modules will be coded, some not
- test the completed ones while working on others
- if tests show errors, have to recode, etc

Configuration Management System tracks what goes on

- records state of cork of modules
- notes which modules versions used to build release

Development/Testing Environment supports all activities

- distributed IDE on steroids

Techniques for Establishing Program Correctness

(Defect) program testing

– Verifies correctness on point test vectors
- regression testing . . .
- usually develop test vectors from bug reports
- bugs are reported with test vector to display it

– Validates correctness if test data supplied by customer

Program inspection

only useful for verification

- group walk-throughs catch problems

formal methods can establish absolute correctness

- specialists operate provers to demonstrate correctness
- requires formal specification to check against

(Defect) Program Testing

- Commonly used cyclic process
  - test
    - select test vectors (set of test data)
    - compare computed vs expected outputs
  - debug

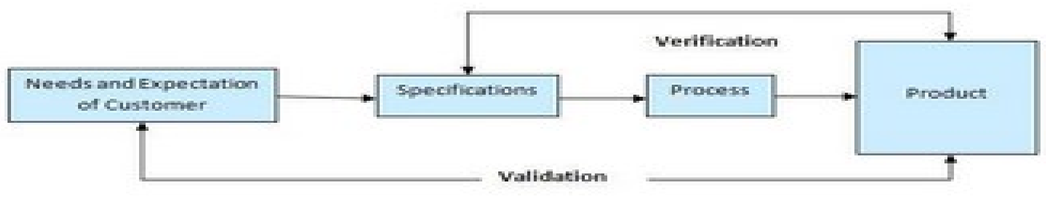
- symptom identified in testing tracked to cause
- recode
    - error cause is corrected
- Cyclic process because . . .
    - rewriting is not perfect and/or causes new errors
    - often correcting one bug unmasks another
- Expensive and time-consuming process
    - 2003 study showed takes 30-40% of development time/effort

Defect Testing Strategies

- Defect testing takes place at different levels
    - module testing
        - each module tested independently
        - in large modules, components tested (unit test)
    - subsystem testing
        - sets of modules tested for correct interaction
    - system testing
        - complete system is tested against use scenarios
- Tests must be repeatable
    - repeating old tests passed earlier is regression testing
        - necessary to prevent inadvertently restoring old errors

Program Verification:

- Makes sure that the product is designed to deliver all functionality to the customer.
- Verification is done at the starting of the development process. It includes reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.
- Suppose you are building a table. Here the verification is about checking all the parts of the table, whether all the four legs are of correct size or not. If one leg of table is not of the right size it will imbalance the end product. Similar behavior is also noticed in case of the software product or application. If any feature of software product or application is not up to the mark or if any defect is found then it will result into the failure of the end product. Hence, verification is very important. It takes place at the starting of the development process.



Software verification and validation

- It answers the questions like: **Am I building the product right?**
- Am I accessing the data right (in the right place; in the right way).
- It is a Low level activity
- Performed during development on key artifacts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards.
- Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

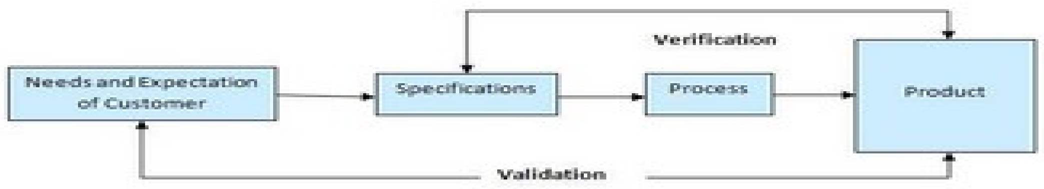
According to the Capability Maturity Model(CMMI-SW v1.1) we can also define verification as the process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610].

Advantages of Software Verification :

1. Verification helps in lowering down the count of the defect in the later stages of development.
2. Verifying the product at the starting phase of the development will help in understanding the product in a better way.
3. It reduces the chances of failures in the software application or product.
4. It helps in building the product as per the customer specifications and needs.

Validation:

- Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs.
- Validation is done at the end of the development process and takes place after verifications are completed.
- It answers the question like: **Am I building the right product?**
- Am I accessing the right data (in terms of the data required to satisfy the requirement).
- It is a High level activity.
- Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment.
- Determination of correctness of the final software product by a development project with respect to the user needs and requirements.



Software verification and validation

According to the Capability Maturity Model(CMMI-SW v1.1) we can also define validation as The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610].

A product can pass while verification, as it is done on the paper and no running or functional application is required. But, when same points which were verified on the paper is actually developed then the running application or product can fail while validation. This may happen because when a product or application is build as per the specification but these specifications are not up to the mark hence they fail to address the user requirements.

Advantages of Validation:

1. During verification if some defects are missed then during validation process it can be caught as failures.
2. If during verification some specification is misunderstood and development had happened then during validation process while executing that functionality the difference between the actual result and expected result can be understood.
3. Validation is done during testing like feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.
4. Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.

Validation is basically done by the testers during the testing. While validating the product if some deviation is found in the actual result from the expected result then a bug is reported or an incident is raised. Not all incidents are bugs. But all bugs are incidents. Incidents can also be of type 'Question' where the functionality is not clear to the tester.

Hence, validation helps in unfolding the exact functionality of the features and helps the testers to understand the product in much better way. It helps in making the product more user friendly.

Testing Automation & Testing Tools:

Continuous Integration (CI) is a practice in Software Engineering, where all the developers local working code base will be merged to share with a common repository several times during the product development. It was first adopted as a part of Extreme Programming (XP). The main purpose of CI is prevent developers stepping over each other code and eliminate integration issues. CI works in tandem with other best practices like Configuration management, compilation, software build, deployment, and testing which are bundled into a single automated and repeatable process. Due to rapid integration of code, it is more likely that defects surface faster than it could compare to normal manual integration.

CI has the following built in automation to check the validity of the code that was checked in:

- **Static code analysis:** Reporting the results of static code execution

- **Compile:** Generating the executable files by linking the code and compiling after
- **Unit test:** Writing unit tests, executing them, checking code coverage and reporting the results
- **Deploy:** Build the code and install it into a test/production environment.
- **Integration test:** Providing results by executing the integration tests.
- **Report (dashboard):** Indicating the status of key parameters by posting Red, Green, and Yellow to a publicly visible location.

The Cycle of CI has been shown below in the figure 3

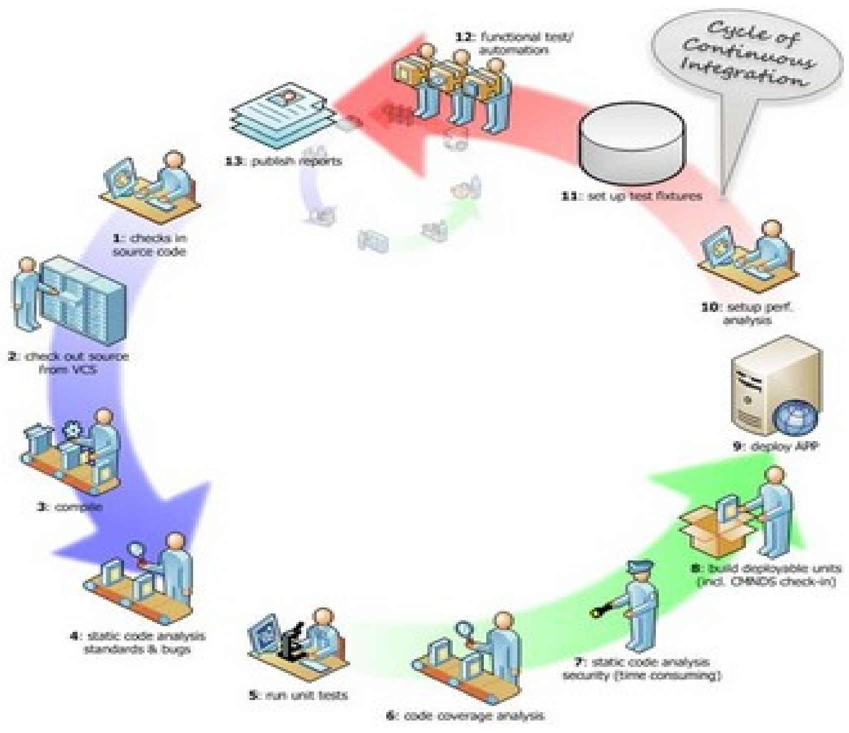

Cycle of Continuous Integration
12: functional test/ automation
13: publish reports
11: set up test fixtures
1: checks in source code
10: setup perf. analysis
2: check out source from VCS
9: deploy APP
3: compile
8: build deployable units (incl. CMNDS check-in)
4: static code analysis standards & bugs
7: static code analysis security (time consuming)
5: run unit tests
6: code coverage analysis

Figure 3: Cycle of Continuous Integration

CI helps testers to perform automation effectively to uncover defects at a faster rate and improve the regression results. The automation will also give the percentage of test coverage area. The automation report also covers the number of user stories and functionality mapped to the product increment. Automation also reduces the manual testing effort of the testers and makes their life easy. Testers can also manually test the areas of failure and collaborate with the developers to fix the defects.

There are several tools that are being used in organizations as a part of continuous integration to build the automation scripts. Few examples of the tools are JUnit, Selenium, SONAR etc. CI automation will reduce the time to deliver, since it replaces the traditional manual testing. Build tools are linked to automation server and build server and deploy to testing, staging and production environments. Organizations started using these build tools to replace the traditional quality control.

CI provides the following benefits to its users

- Enables a quick feedback mechanism on the build results
- Helps collaboration between various product teams within the same organization
- Decreases the risk of regression since the code is often churned by other developers.
- Maintains version control within for various product releases and patch releases.
- Reduces the technical debt within the code.
- Allows earlier detection and prevention of defects
- Reduces manual testing effort
- Provides a facility of falling back to previous versions in case of any problem to the current build

CI also has the following risks and challenges:

- CI tools maintenance and their administration have associated costs to it.
- CI guidelines need to be well established before starting it.
- Test automation is a rare skill in the market and existing testers may have to be trained on that.
- Full fledge test coverage is required to see the benefits to automation.
- Teams sometimes depend too much on the unit testing and ignore automation and acceptance testing.

So in summary, CI is advantageous to a project team, if the required tools are put in place for automating the build process.

Lecture 3: Concept of Software Quality, Software Quality Attributes, Software Quality Metrics and Indicators,

Software quality:

1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets customer or user needs or expectations.

Software Quality Assurance

1. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
2. A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

DEFINITION OF QUALITY, QA, SQA

Quality:

The term 'quality' is often used in a vague, blurred way. If someone talks about 'working on quality', they may simply mean activities designed to improve the organisation and its services. Quality is essentially about learning what you are doing well and doing it better. It also means finding out what you may need to change to make sure you meet the needs of your service users. Quality is about:

- Knowing what you want to do and how you want to do it
- Learning from what you do
- using what you learn to develop your organization and its services
- seeking to achieve continuous improvement
- Satisfying your stakeholders

-

Those different people and groups with an interest in your organization. Quality assurance is the process of verifying or determining whether products or services meet or exceed customer expectations. Quality assurance is a process driven
approach with specific steps to help define and attain goals. This process considers
design, development, production, and service. The most popular tool used to determine quality assurance is the Shewhart

Lecture 4: The SEI Capability Maturity Model CMM),

Lecture 5: SQA Activities, Formal SQA Approaches: Proof of correctness, Statistical quality assurance, Clean room process.

SQA Activities:

FO SQA is the process of evaluating the quality of a product and enforcing adherence to software product standards and procedures. It is an umbrella activity that ensures conformance to standards and procedures throughout the SDLC of a software product. There are a large number of tasks involved in SQA activities.

This include-

i. Formulating a quality management plan
ii. Applying software engineering techniques
iii. Conducting formal technical reviews
iv. Applying a multi-tiered testing strategy
iv. Enforcing process adherence
vi. Controlling change
vii. Measuring impact of change
viii. Performing SQA audits
ix. Keeping records and report

Formulating a Quality Management Plan

One of the tasks of SQA is the formulation of a quality management plan. The quality management plan dentifies the quality aspects of the software product to be developed. It helps in planning checkpoints for work products and the development process. It also tracks changes made to the development process based on the results of the checks. The quality management plan is tracked as a live plan throughout the SDLC

Applying Software Engineering

Application of software engineering techniques helps the software designer to achieve high quality specification. The designer gathers information using techniques such as interviews and FAST. Using the information gathered, the designer prepares project estimation with the help of techniques such as WBS, SLOC estimation, or FP estimation

Conducting Formal Technical Reviews

Formal technical review (FTR) in conducted to assess the quality and design of the prototype. It is a meeting with the technical staff to discuss the quality requirements of a software product and its design quality. FTRs help in detecting errors at an early phase of development. This prevents errors from percolating down to the latter phases and resulting in rework

Applying a Multi-tiered Testing Strategy

Software testing is a critical task of SQA activity, which aims at error detection.Unit testing is the first level of testing. The subsequence levels of testing are integration testing and system level testing. There are various testing strategies followed by organization. At times, developers perform unit testing and integration testing with independence testing support. There are also occasions where testers perform functional testing and system level testing with developer support. Sometimes beta testing with selected clients is also conducted to test the product before it is finally released

Enforcing Process Adherence

This task of SQA emphasizes the need for process adherence during product development. In addition, the development process should also adhere to procedures defined for product development. Therefore, this is a combination of two tasks, product evaluation and process monitoring

Product Evaluation

Product evaluation ensures that the standards laid down for a project are followed. During product evaluation, the compliance of the software product to the existing standards is verified. Initially, SQA activities are conducted to monitor the standards and procedures of the project. Product evaluation ensures that the software product reflects the requirements identified in the project management plan.

Process Monitoring

Process monitoring ensures that appropriate steps to follow the product development procedures are carried out. SQA monitors processes by comparing the actual steps carried out with the steps in the documented procedures. Product evaluation and process monitoring ensure that the development and control processes described in the project management plan are correctly carried out. These tasks ensure that the project-re1ated procedures and standards are followed. They also ensure that products and processes conform to standards and procedures. Audits ensure that product evaluation and process monitoring are performed

Controlling Change

This task combines human procedures and automated tools to provide a mechanism for change control. The change control mechanism ensures software quality by formalizing requests for change, evaluating the nature of change, and controlling the impact of change. Change control mechanism is implemented during the development and maintenance stages

Measuring Impact of Change

Change is inevitable in the SDLC. However, the change needs to be measured and monitored. Changes in the product or process are measured using software quality metrics. Software qua1ity metrics helps in estimating the cost and resource requirements of a project. To control software quality; it is essential to measure quality and then compare it with established standards. Software qua1ity metrics are used to evaluate the effectiveness of techniques and tools, the productivity of development activities and the qua1ity of products. Metrics enables mangers and developers to monitor the activities and proposed changes throughout the SDLC and initiate corrective actions.

Performing SQA Audits
SQA audits scrutinize the software development process by comparing it to established processes. This ensures that proper control is maintained over the documents required during SDLC. Audits also ensure that the status of an activity performed by the developer is reflected in the status report of the developer.

Keeping Records and Reporting
Keeping records and reporting ensure the collection and circulation of information relevant to SQA. The results of reviews, audits, change control, testing, and other SQA activities are reported and compiled for future reference
Formal Technical Reviews
One of the most common, yet important, software quality assurance activity performed is FTR. This activity is performed to check errors in logic, function, or implementation for any representation of the software. Using FTR, you can verify whether or not the software product adheres to the defined standards. FTR is also conducted to check for consistency in the software product and audit the manageability of the software product. It includes activities such as walkthrough and inspection. FTR focuses on specific parts of the software product, such as the requirements components, detailed design module, and the source code listing for a module. FTR also concentrates on the entire product. The participants in FTR are the developer, the project leader, and all the reviewers. At the end of the review meeting, the issues recorded are formalized into review issues list. The minutes of the meeting are summarized into a review summary report as displayed in table 4

RMAL APPROACHES TO SQA

Proof of Correctness

- Treat a program as a mathematical object.
- Developed with a language with a rigorous syntax.
- Are attempts at developing a rigorous approach to specification of software requirements?
- With both can attempt to develop a mathematical proof that a program conforms exactly to its specification.
- In the code, can at selected statements formulate assertions on the set of correct values for program variables.
- Can then show that the statements between these assertions do the correct transformation of the values in the assertions.

Statistical Quality Assurance

In statistical quality assurance:

1. Information about software defects is collected and categorized.
2. An attempt is made to trace each defect to its underlying cause (e.g., not conforming to the specification, design error, violation of standards, poor communication with customer ...)
3. Using the 'Pareto principle' (80% of defects can be traced to 20% of all possible causes), isolate the 20% of causes (the "vital few").
4. Once the "vital few" causes have been identified, correct the problems that have caused the defects.

The Clean room Process

- Use statistical quality control and formal program verification.
- Attempt to prevent defects rather than find defects.
- In projects attempted so far with this method (size between 1000 and 50,000 LOC), 90% of
- all defects were found before the first execution tests were conducted.
- Has not been widely applied in industry.
- Requires significant change in both management and technical approaches to software development.

CLEANROOM METHODOLOGY

The objective of the Clean room methodology is to achieve or approach zero defects with certified reliability. As described by Hausler (1994), the Clean room methodology provides a complete discipline within which software personnel can plan, specify, design, verify, code, test and certify software. In a Clean room development, correctness verification replaces unit testing and debugging. After coding is complete, the software immediately enters system test with no debugging. All test errors are accounted for from the first execution of the program with no private testing allowed. As opposed to many development processes, the role of system testing is not to test in quality; the role of system testing is to certify the quality of the software with respect to the systems specification. This process is built upon an incremental development approach. Increment N+1 elaborates on the top down design of increment N . The Clean room process is built upon function theory where programs are treated as rules for mathematical functions subject to stepwise refinement and verification. Clean room specifications and designs are built upon box structure specifications and design. Box structure specifications begin with a black-box specification in which the expected behavior of the system is specified in terms of the system stimuli, responses and transition rules. Black boxes are then translated into state-boxes which define encapsulated state data required to satisfy black box behavior. Clear box designs are finally developed which define the procedural design of services on state data to satisfy black box behavior. Team reviews are performed to verify the correctness of every condition in the specification. During the specification stage an expected usage profile is also developed, which assigns probabilities or frequency of expected use of the system components. During system correctness testing, the system is randomly tested based on the expected usage of the system. In this process, software typically enters system test with near zero defects. The Clean room process places greater emphasis on design and verification rather than testing. In this process errors are detected early in the life cycle, closer to the point of insertion of the error.

CONCLUSION

Quality Assurance is possible only if quality products are turned out. In turn it is helped by auditing and reporting. Needless to say mere auditing and reporting will not produce quality products. Periodical and many a time continuous auditing and reporting systems are basic tools. The management has access to some data anyway. But will it enable the management to feel the standard and quality of the software product that is under process? Audit and report supply the management with what we may call as post-inspection data so that the management can feel confident that the product quality is up to the goals it is set to meet. Quality is a key measure of project success. It is what a customer remembers in the long run. High-quality products result in customer satisfaction, while poor quality results in customer dissatisfaction. Software quality factors cannot be measured because of their vague definitions. It is necessary to find measurements, or metrics, which can be used to quantify them as non-functional requirements.