Unit-1

Unit-1 Introduction to Software concepts
Definition and characteristics of Software
Overview of SDLC

Definition of Software

Computer software, or just software, is a collection of computer programs and related data that provides the instructions for telling a computer what to do and how to do it. Software refers to one or more computer programs and data held in the storage of the computer for some reasons. In other words, software is a set of *programs, procedures, algorithms* and its *documentation* concerned with the operation of a data processing system.

Following functionality, there are 5 other software attributes that characterize the usefulness of the software in a given environment.

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Each of the following characteristics can only be measured (and are assumed to exist) when the functionality of a given system is present. In this way, for example, a system can not possess *usability* characteristics if the system does not function correctly (the two just don't go together).

Functionality

Functionality is the essential purpose of any product or service. For certain items this is relatively easy to define, for example a ship's anchor has the function of holding a ship at a given location. The more functions a product has, e.g. an ATM machine, then the more complicated it becomes to define it's functionality. For software a list of functions can be specified, i.e. a sales order processing systems should be able to record customer information so that it can be used to reference a sales order. A sales order system should also provide the following functions:

Record	Sales order product, price and quantity
Calculate	appropriate sales tax.
Calculate	date available to ship, based on inventory.
Calculate	purchase orders when stock falls below given thresh hold

Reliability

Once a software system is functioning, as specified, and delivered the reliability characteristic defines the capability of the system to maintain its service provision under defined conditions for defined periods of time. One aspect of this characteristic is *fault tolerance* that is the ability of a system to withstand component failure. For example if the network goes down for 20 seconds then comes back the system should be able to recover and continue functioning.

Usability

Usability only exists with regard to functionality and refers to the ease of use for a given function. For example a function of an ATM machine is to dispense cash as requested. Placing common amounts on the screen for selection, i.e. $20.00, $40.00, $100.00 etc, does not impact the function of the ATM but addresses the Usability of the function. The ability to learn how to use a system (learnability) is also a major sub characteristic of usability.

Efficiency

This characteristic is concerned with the system resources used when providing the required functionality. The amount of disk space, memory, network etc. provides a good indication of this characteristic. As with a number of these characteristics, there are overlaps. For example the usability of a system is influenced by the system's Performance, in that if a system takes 3 hours to respond the system would not be easy to use although the essential issue is a performance or efficiency characteristic.

Maintainability

The ability to identify and fix a fault within a software component is what the maintainability characteristic addresses. In other software quality models this characteristic is referenced as supportability. Maintainability is impacted by code readability or complexity as well as modularization. Anything that helps with identifying the cause of a fault and then fixing the fault is the concern of maintainability. Also the ability to verify (or test) a system, i.e. testability, is one of the sub characteristics of maintainability.

Portability

This characteristic refers to how well the software can adopt to changes in its environment or with its requirements. The sub characteristics of this characteristic include adaptability. Object oriented design and implementation practices can contribute to the extent to which this characteristic is present in a given system.

The full table of Characteristics and Sub characteristics for the ISO 9126-1 Quality Model is:-

Characteristics	Subcharacteristics	Definitions
Functionality	Suitability	This is the essential Functionality characteristic and refers to the appropriateness (to specification) of the functions of the software.
	Accurateness	This refers to the correctness of the functions, an ATM may provide a cash dispensing function but is the amount correct?
	Interoperability	A given software component or system does not typically function in isolation. This sub characteristic concerns the ability of a software component to interact with other components or systems.
	Compliance	Where appropriate certain industry (or government) laws and guidelines need to be complied with, i.e. SOX. This sub characteristic addresses the compliant capability of software.
	Security	This sub characteristic relates to unauthorized access to the software functions.
Reliability	Maturity	This sub characteristic concerns frequency of failure of the software.
	Fault tolerance	The ability of software to withstand (and recover) from component, or environmental, failure.
	Recoverability	Ability to bring back a failed system to full operation, including data and network connections.
Usability	Understandability	Determines the ease of which the systems functions can be understood, relates to user mental models in Human Computer Interaction methods.
	Learnability	Learning effort for different users, i.e. novice, expert, casual etc.
	Operability	Ability of the software to be easily operated by a given user in a given environment.

Efficiency	Time behavior	Characterizes response times for a given thru put, i.e. transaction rate.
	Resource behavior	Characterizes resources used, i.e. memory, cpu, disk and network usage.
Maintainability	Analyzability	Characterizes the ability to identify the root cause of a failure within the software.
	Changeability	Characterizes the amount of effort to change a system.
	Stability	Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.
	Testability	Characterizes the effort needed to verify (test) a system change.
Portability		
	Adaptability	Characterizes the ability of the system to change to new specifications or operating environments.
	Installability	Characterizes the effort required to install the software.
	Conformance	Similar to compliance for functionality, but this characteristic relates to portability. One example would be Open SQL conformance which relates to portability of database used.
	Irreplaceability	Characterizes the *plug and play* aspect of software components; that is how easy is it to exchange a given software component within a specified environment.

SDLC - Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

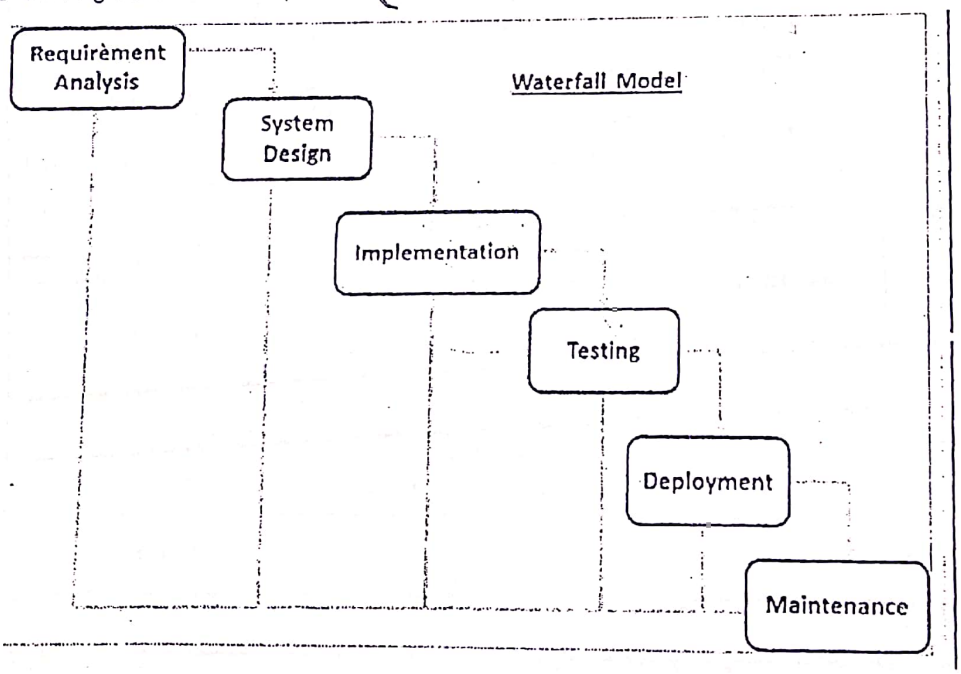
The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are –

- Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- System Design – the requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- Implementation – with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- Integration and Testing – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- Deployment of system – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- Maintenance – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are –

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.

- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows –

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows –

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.

- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Unit-2

Introduction to Testing

Contents

What is testing & Importance of testing
Testing goals and characteristics
Testing during planning stage
Testing during Design stage,
Testing During Coding Stage

What is testing?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities:

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc. It is not possible to test the software at any time during its cycle.

The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC:

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

- It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested.
- The following aspects are to be considered for stopping the testing process:
  - Testing Deadlines
  - Completion of test case execution
  - Completion of functional and code coverage to a certain point
  - Bug rate falls below a certain level and no high-priority bugs are identified

Importance of Testing

1:-Improves Quality: Software programming and development stages are complex. If there are bugs and glitches at any level, it could lead to irreparable financial losses. Companies behind the product can also lose their image and market standing. Through quality testing one can prevent these mishaps from happening.

2. Validates and Verifies: If a program carries a validation, then it guarantees that it is able to meet the predefined goals. One can hire independent agencies for testing software or they can perform the task in-house. This would add an extra quality mark to the product for the user.

3. Estimates Reliability: This presents a report about the software and how dependable it is in the market for various uses. Functionality is checked as well as software is analyzed on the basis of the end user review.

4. Usability and Operations: In order to test this, it is released to only a selected group of people and then a view point is acquired from each user.

5. Prevents defect from migrating: When errors are detected early, it prevents the errors from causing any bigger defect in the subsequent development level. This leads to huge savings.

TESTING GOALS

A goal is a projected state of affairs that a person or system plans or intends to achieve. A goal has to be accomplishable and measurable. It is good if all goals are interrelated. In testing we can describe goals as intended outputs of the software testing process. Software testing has following goals:

1. Verification and Validation

It would not be right to say that testing is done only to find faults. Faults will be found by everybody using the software. Testing is a quality control measure used to verify that a product works as desired . Software testing provides a status report of the actual product in comparison to product requirements (written and implicit). Testing process has to verify and validate whether the software fulfills conditions laid down for its release/use . Testing should reveal as many errors as possible in the software under test, check whether it meets its requirements and also bring it to an acceptable level of quality.

2. Priority Coverage

Exhaustive testing is impossible. We should perform tests efficiently and effectively, within budgetary and scheduling imitations. Therefore testing needs to assign effort reasonably and prioritize thoroughly. Generally every feature should be tested at least with one valid input case. We can also test input permutations, invalid input, and non-functional requirements depending upon the operational profile of software. Highly present and frequent use scenarios should have more coverage than infrequently encountered and insignificant scenarios. A study by on 25 million lines of code also revealed that 70-80% of problems were due to 10-15% of modules , 90% of all defects

were in modules containing 13% of the code, 95% of serious defects were from just 2.5% of the code.

3 Balanced

Testing process must balance the written requirements, real-world technical limitations, and user expectations. The testing process and its results must be repeatable and independent of the tester, i.e., consistent and unbiased [4]. Apart from the process being employed in development there will be a lot unwritten or implicit requirements. While testing, the software testing team should keep all such requirements in mind. They must also realize that we are part of development team, not the users of the software. Testers personal views are but one of many considerations. Bias in a tester invariably leads to a bias in coverage. The end user's viewpoint is obviously vital to the success of the software, but it is not all that matters as all needs cannot be fulfilled because of technical, budgetary or scheduling limitations. Every defect/shortcoming has to be prioritized with respect to their time and technical constraints.

4 Traceable

Documenting both the successes and failures helps in easing the process of testing. What was tested, and how it was tested, are needed as part of an ongoing testing process. Such things serve as a means to eliminate duplicate testing effort [10]. Test plans should be clear enough to be re-read and comprehended. We should agree on the common established documentation methods to avoid the chaos and to make documentation more useful in error prevention.

5 Deterministic

Problem detection should not be random in testing. We should know what we are doing, what are we targeting, what will be the possible outcome. Coverage criteria should expose all defects of a decided nature and priority. Also, afterward surfacing errors should be categorized as to which section in the coverage it would have occurred, and can thus present a definite cost in detecting such defects in future testing. Having clean insight into the process allows us to better estimate costs and to better direct the overall development.

Characteristics of software testing

1. High Probability of detecting Errors: - To detect maximum errors, the tester should understand the software thoroughly and try to find the possible ways in which the software can fail. In a program to divide two numbers, for example, the possible way in which the program can fail is when 2 and zero are given as inputs and two is to be divided by zero. In this case, a set of tests should be developed that can demonstrate an error in the division operator.
2. No Redundancy: - Resources and testing time are limited in software development process. Thus, it is not beneficial to develop several tests, which have the same intended purpose. Every test should have a distinct purpose.
3. Choose the most appropriate Test: - There can be different tests that have the same intent but due to certain limitations, such as time and resource constraint, only few of them are used. In such as case, the tests that have the highest probability of finding errors should be considered.
4. Moderate: - A test is considered good if it is neither too simple nor too complex. Many tests can be combined to form one test case. However, this can increase the complexity and leave many errors undetected. Hence, all tests should be performed separately.

TESTING PRINCIPLES

A principle is an accepted rule or method for application in action that has to be, or can be desirably followed. Testing Principles offer general guidelines common for all testing which assists us in performing testing effectively and efficiently. Principles for software testing are:

1 Test a program to try to make it fail

Testing is the process of executing a program with the intent of finding errors [9]. Our objective should be to demonstrate that a program has errors, and then only true value of testing can be accomplished. We should expose failures (as many as possible) to make testing process more effective.

2 Start testing early

If you want to find errors, start as early as possible. This helps in fixing enormous errors in early stages of development, reduces the rework of finding the errors in the initial stages. Fixing errors at early phases cost less as compared to later phases. For example, if a problem in the requirements is found after releasing the product, then it would cost 10– 100 times more to correct than if it had already been found by the requirements review.

Figure 1 depicts the increase in cost of fixing bugs detected/fixed in later phases.

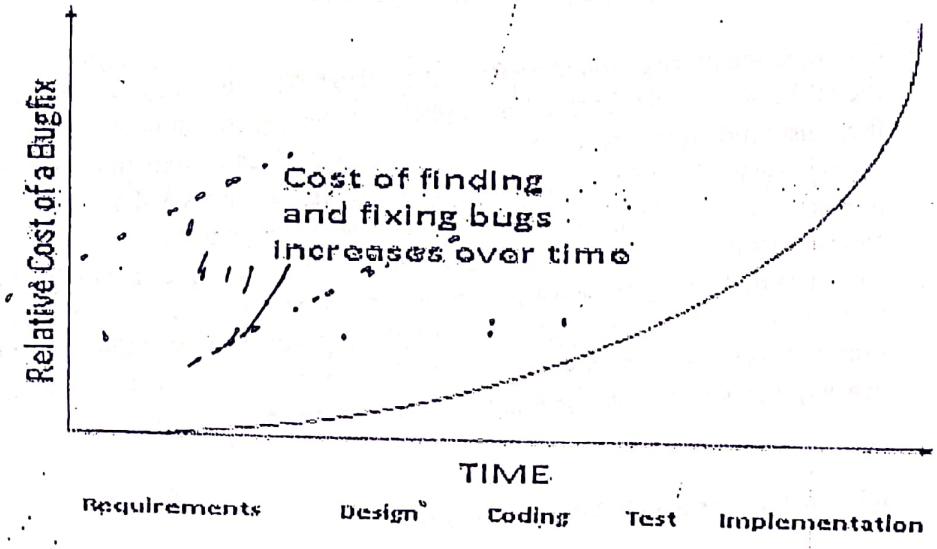


Figure 1. Cost of fixing bugs in different phases.

3 Testing is context dependent

Testing is done differently in different contexts. Testing should be appropriate and different for different points of time. For example, a safety-critical software is tested differently from an e-commerce site. Even a system developed using the waterfall approach is tested significantly differently than those systems developed using agile development approach. Even the objectives of testing differ at different point in software development cycle. For example, the objective of unit and integration testing is to ensure that code implemented the design properly. In system testing the objective is to ensure the system does what customer wants it to do [15]. Type of testing approach that will be used depends on a number of factors, including the type of system, regulatory standards, user requirements, level and type of risk, test objective, documentation available, knowledge of the testers, time and budget, development life cycle.

4 Define Test Plan

Test Plan usually describes test scope, test objectives, test strategy, test environment, deliverables of the test, risks and mitigation, schedule, levels of testing to be applied, methods, techniques and tools to be used. Test plan should efficiently meet the needs

of an organization and clients as well. The testing is conducted in view of a specific purpose (test objective) which should be stated in measurable terms, for example test effectiveness, coverage criteria. Although the prime objective of testing is to find errors, a good testing strategy also assesses other quality characteristics such as portability, maintainability and usability.

5 Design Effective Test cases

Complete and precise requirements are crucial for effective testing. User Requirements should be well known before test case design. Testing should be performed against those user requirements. The test case scenarios shall be written and scripted before testing begins. If you do not understand the user requirements and architecture of the product you are testing, then you will not be able to design test cases which will reveal more errors in short amount of time. A test case must consist of a description of the input data to the program and a precise description to the correct output of the program for that set of input data. A necessary part of test documentation is the specification of expected results, even if providing such results is impractical [9]. These must be specified in a way that is measurable so that testing results are unambiguous.

6 Test for valid as well as invalid conditions

In addition to valid inputs, we should also test system for invalid and unexpected inputs/conditions. Many errors are discovered when a program under test is used in some new and unexpected way and invalid input conditions seem to have higher error detection yield than do test cases for valid input conditions [9]. Choose test inputs that possibly will uncover maximum faults by triggering failures.

7 Review Test cases regularly

Repeating same test cases over and over again eventually will no longer find any new errors. Therefore the test cases need to be regularly reviewed and revised, and new and different need to be written to exercise different parts of the software or system to potentially find more defects. We should target and test susceptible areas. Exploratory Testing can prove very useful. Exploratory testing is any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.

8 Testing must be done by different persons at different levels

Different purposes are addressed at the different levels of testing. Factors which decide who will perform testing include the size and context of the system, the risks, the development methodology used, the skill and experience of the developers. Testing of individual program components is usually the responsibility of the component developer (except sometimes for critical systems); Tests at this level are derived from the developer's experience. Testing at system/sub-system level should be performed by the

independent persons/team. Tests at this level are based on a system specification [6]. Development staff shall be available to assist testers. Acceptance Testing is usually performed by end user or customer. Release Testing is performed by Quality Manager.

Figure 2 shows persons involved at different levels of software testing.

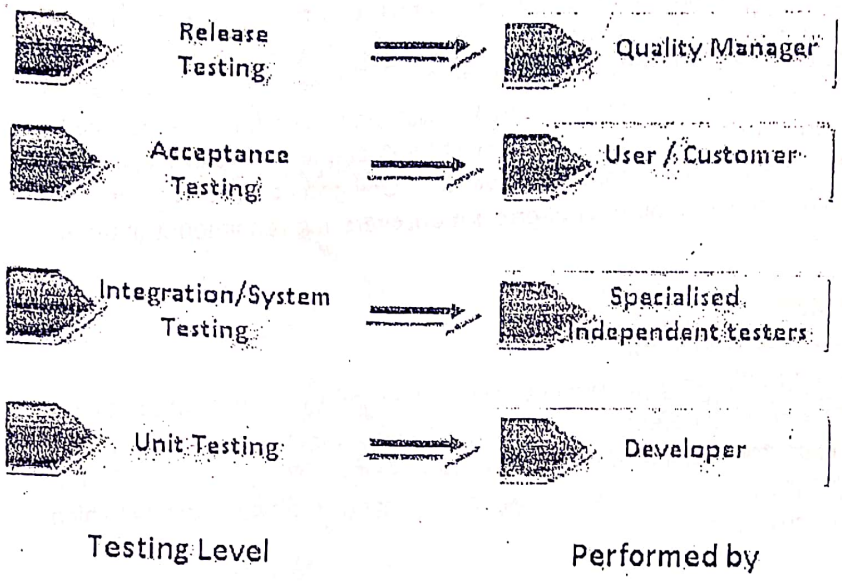


Figure 2: Software Testing Levels.

9 Test a program innovatively

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. It is impossible to test a program sufficiently to guarantee the absence of all errors [9]. Instead of exhaustive testing, we use risks and priorities to focus testing efforts more on suspected components as compared to less suspected and infrequently encountered components.

10 Use both Static and Dynamic testing

Static testing is good at depth; it reveals developers understanding of the problem domain and data structure. Dynamic testing is good at breadth; it tries many values, including extremes that humans might miss. To eliminate as many errors as possible, both static and dynamic testing should be used [12].

11 Defect clustering

Errors tend to come in clusters. The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that

section [9], so additional testing efforts should be more focused on more error-prone sections until it is subjected to more rigorous testing.

12 Test Evaluation

We should have some criterion to decide whether a test is successful or not. If limited test cases are executed, the test oracle (human or mechanical agent which decides whether program behaved correctly on a given test [1]) can be tester himself/herself who inspects and decides the conditions that makes test run successful. When test cases are quite high in number, automated oracles must be implemented to determine the success or failure of tests without manual intervention. One good criterion for test case evaluation is test effectiveness (number of errors it uncovers in given amount of time).

13 Error Absence Myth

System that does not fulfill user requirements will not be usable even if it does not have any errors. Finding and fixing defects does not help if the system built does not fulfill the users' needs and expectations. In addition to positive software testing (which verify that system does what it should do), we should also perform negative software testing (which verify that system does not do what it should not do).

14 End of Testing

Software testing is an ongoing process, which is potentially endless but has to be stopped somewhere. Realistically, testing is a trade-off between budget, time, and quality. The effort spent on testing should be correlated with the consequences of possible program errors. The possible factors for stopping testing are:

1. The risk in the software is under acceptable limit.

2. Coverage of code/functionality/requirements reaches a specified point.

3. Budgetary/scheduling limitations.

TESTING LIMITATIONS

Limitation is a principle that limits the extent of something. Testing also has some limitations that should be taken into account to set realistic expectations about its benefits. In spite of being most dominant verification technique, software testing too has following limitations:

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

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

3. Testing cannot establish that a product functions properly under all conditions but can only establish International Journal of Computer Applications (0975 – 8887) Volume 6– No.9, September 2010 40 that it does not function properly under specific conditions [14].

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

Testing during planning stage

In the Planning Phase, the team defines the solution in detail what to build, how to build it, who will build it, and when it will be built. During this phase the team works through the design process to create the solution architecture and design, writes the functional specification, and prepares work plans, cost estimates, and schedules for the various deliverables.

The Planning Phase culminates in the Project Plans Approved Milestone, indicating that the project team, customer, and key project stakeholders agree on the details of the plans. Plans prepared by team members for areas such as communications, test, and security, are rolled up into a master plan that the program manager coordinates. The team's goal during this phase is to document the solution to a degree that the team can produce and deploy the solution in a timely and cost-effective manner. These documents are considered living documents, meaning they will be updated continuously throughout the Planning Phase.

Major Planning Phase Tasks and Owners

Major Tasks	Owners
Developing the solution design and architecture The team begins the design process with the solution design and architecture and culminates it with a design document that becomes part of the functional specification.	Development
Validating the technology	Development
Creating the functional specification The team creates a functional specification that describes the solution requirements, the architecture, and the detailed design for all the features.	Program Management
Developing the project plans The team develops a collection of plans that address how the six MSF team roles will perform their tasks. These plans are consolidated into the master project plan. The master project plan is considered a roll-up of all of the plans and, thus, also includes strategic items such as the approach, dependencies, and assumptions for the solution.	Program Management
Creating the project schedules The team creates the master project schedule, which consists of milestonedriven schedules developed by each of the individual team roles when planning their respective activities.	Program Management
Setting up the development and test environments The team creates development and testing environments that are independent of the production environment to develop and test the solution.	Development and Test
Closing the Planning Phase The team completes the approval process for the Project Plans Approved Milestone and documents the results of completing the tasks performed in this phase.	Project team

Table 3.2 Role Cluster Focuses and Responsibilities in Planning Phase

Role Cluster	Focus and Responsibility
Product Management	Conceptual design; business requirements analysis; communications plan
Program Management	Conceptual and logical design; functional specification; master project plan and master project schedule; budget
Development	Technology evaluation; logical and physical design; development plan/schedule; development estimates
User Experience	Usage scenarios/use cases, user requirements, and localization/accessibility requirements; user documentation/training; plan/schedule for usability testing
Test	Design evaluation; testing requirements; test plan/schedule
Release Management	Design evaluation; operations requirements; pilot and deployment plan/schedule

Unit-3

Software Testing Life Cycle

Topics to be covered

1. *Principle of verification and validation*
2. *Techniques of verification*
   a. *Reviews*
   b. *Inspection*
   c. *Walkthrough*

3. *V-testing Model*
    a. *Software development V & V*
    b. *Software Acquisition V & V*
    c. *Software Supply V & V.*

PRINCIPLES OF SOFTWARE VERIFICATION AND VALIDATION

Verification can mean the:

- Act of reviewing, inspecting, testing, checking, auditing, or otherwise establishing and documenting whether items, processes, services or documents confirm conventional to specified requirements.

- Process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of the phase .

Validation can mean the:

- Validation is, according to its ANSI/IEEE definition, 'the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements'.

Validation is, therefore, 'end-to-end' verification.

In simple words Verification & validation

- Software *verification* asks the question, "Are we building the product right?" that is, does the software conform to its specification.
- Software *validation* asks the question, "Are we building the right product?"; that is, is the software doing what the user really requires.

Software Verification Techniques:-

Software verification is a broader and more complex discipline of software engineering whose goal is to assure that software fully satisfies all the expected requirements.

A: Software reviews: - It is of two type

i. Software Management Review
ii. Software Audit Review

- Software management reviews are conducted by management representatives to evaluate the status of work done and to make decisions regarding downstream activities.

Definition of Software management reviews by the IEEE as:

- A systematic evaluation of a software acquisition, supply, development, operation, or maintenance process performed by or on behalf of management ... [and conducted] to monitor progress, determine the status of plans and schedules, confirm requirements and their system allocation, or evaluate the effectiveness of management approaches used to achieve fitness for purpose. Management reviews support decisions about corrective actions, changes in the allocation of resources, or changes to the scope of the project.
- Management reviews are carried out by, or on behalf of, the management personnel having direct responsibility for the system. Management reviews identify consistency with and deviations from plans, or adequacies and inadequacies of management procedures. This examination may require more than one meeting. The examination need not address all aspects of the product."

- A Software management review is a management study into a project's status and allocation of resources. It is different from both a software engineering peer review, which evaluates the technical quality of software products, and a software audit, which is an externally conducted audit into a project's compliance to specifications, contractual agreements, and other criteria.

A management review can be an informal process, but generally requires a formal structure and rules of conduct, such as those advocated in the IEEE standard, which are:

1. Evaluate entry?
2. Management preparation?
3. Plan the structure of the review 4. Overview of review procedures?
5. [Individual] Preparation?
6. [Group] Examination?
7. Rework/follow-up?
8. [Exit evaluation]?

Software audit reviews are conducted by personnel external to the software project, to evaluate compliance with specifications, standards, contractual agreements, or other criteria.

A software audit review, or software audit, is a type of software review in which one or more auditors who are not members of the software development organization conduct "An independent examination of a software product, software process, or set of software processes to assess compliance with specifications, standards, contractual agreements, or other criteria".

"Software product" mostly, but not exclusively, refers to some kind of technical document. IEEE Std. 1028 offers a list of 32 "examples of software products subject to audit", including documentary products such as various sorts of plan, contracts, specifications, designs, procedures, standards, and reports, but also non-documentary products such as data, test data, and deliverable media.

Software audits are distinct from software peer reviews and software management reviews in that they are conducted by personnel external to, and independent of, the software development organization, and are concerned with compliance of products or processes, rather than with their technical content, technical quality, or managerial implications.

"The purpose of a software audit is to provide an independent evaluation of conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures". The following roles are recommended:

- The *Initiator* (who might be a manager in the audited organization, a customer or user representative of the audited organization, or a third party), decides upon the need for an audit, establishes its purpose and scope, specifies the evaluation criteria, identifies the audit personnel, decides what follow-up actions will be required, and distributes the audit report.
- The *Lead Auditor* (who must be someone "free from bias and influence that could reduce his ability to make independent, objective evaluations") is responsible for administrative tasks such as preparing the audit plan and assembling and managing the audit team, and for ensuring that the audit meets its objectives.
- The *Recorder* documents anomalies, action items, decisions, and recommendations made by the audit team.
- The *Auditors* (who must be, like the Lead Auditor, free from bias) examine products defined in the audit plan, document their observations, and recommend corrective actions. (There may be only a single auditor.)
- The *Audited Organization* provides a liaison to the auditors, and provides all information requested by the auditors. When the audit is completed, the audited organization should implement corrective actions and recommendations.

B. Software inspection

Inspection in software engineering refers to peer review of any work product by trained individuals who look for defects using a well-defined process. An inspection

might also be referred to as a Fagan inspection after Michael Fagan, the creator of a very popular software inspection process.

An inspection is one of the most common sorts of review practices found in software projects. The goal of the inspection is for all of the inspectors to reach consensus on a work product and approve it for use in the project. Commonly inspected work products include software requirements specifications and test plans. In an inspection, a work product is selected for review and a team is gathered for an inspection meeting to review the work product. A moderator is chosen to moderate the meeting. Each inspector prepares for the meeting by reading the work product and noting each defect. The goal of the inspection is to identify defects. In an inspection, a defect is any part of the work product that will keep an inspector from approving it. For example, if the team is inspecting a software requirements specification, each defect will be text in the document which an inspector disagrees with.

The Inspection Process

The inspection process was developed by Michael Fagan in the mid-1970s and it has later been extended and modified.

The process should have entry criteria that determine if the inspection process is ready to begin. This prevents unfinished work products from entering the inspection process. The entry criteria might be a checklist including items such as "The document has been spellchecked".

The stages in the inspections process are: Planning, Overview meeting, Preparation, Inspection meeting, Rework and Follow-up. The Preparation, Inspection meeting and Rework stages might be iterated.

- Planning: The inspection is planned by the moderator.
- Overview meeting: The author describes the background of the work product. Preparation: Each inspector examines the work product to identify possible defects.
- Inspection meeting: During this meeting the reader reads through the work product, part by part and the inspectors point out the defects for every part.
- Rework: The author makes changes to the work product according to the action plans
- Follow-up: The changes by the author are checked to make sure everything is correct.

The process is ended by the moderator when it satisfies some predefined exit criteria.

Inspection roles

During an inspection the following roles are used.

- Author: The person who created the work product being inspected.

- Moderator: This is the leader of the inspection. The moderator plans the inspection and coordinates it.
- Reader: The person reading through the documents, one item at a time. The other inspectors then point out defects.
- Recorder/Scribe: The person that documents the defects that are found during the inspection.
- Inspector: The person that examines the work product to identify possible defects.

Inspection types

Code review

A code review can be done as a special kind of inspection in which the team examines a sample of code and fixes any defects in it. In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved (for example, it could be made more readable or its performance could be improved). In addition to helping teams find and fix bugs, code reviews are useful for both cross-training programmers on the code being reviewed and for helping junior developers learn new programming techniques.

Peer Reviews

Peer reviews are considered an industry best-practice for detecting software defects early and learning about software artifacts. Peer Reviews are composed of software walkthroughs and software inspections and are integral to software product engineering activities. A collection of coordinated knowledge, skills, and behaviors facilitates the best possible practice of Peer Reviews. The elements of Peer Reviews include the structured review process, standard of excellence product checklists, defined roles of participants, and the forms and reports.

Software inspections are the most rigorous form of Peer Reviews and fully utilize these elements in detecting defects. Software walkthroughs draw selectively upon the elements in assisting the producer to obtain the deepest understanding of an artifact and reaching a consensus among participants. Measured results reveal that Peer Reviews produce an attractive return on investment obtained through

accelerated learning and early defect detection. For best results, Peer Reviews are rolled out within an organization through a defined program of preparing a policy and procedure, training practitioners and managers, defining measurements and populating a database structure, and sustaining the roll out infrastructure.

C. Software walkthrough

a walkthrough or walk-through is a form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems"

"Software product" normally refers to some kind of technical document. As indicated by the IEEE definition, this might be a software design document or program source code, but use cases, business process definitions, test case specifications, and a variety of other technical documentation may also be walked through.

A walkthrough differs from software technical reviews in its openness of structure and its objective of familiarization. It differs from software inspection in its ability to suggest direct alterations to the product reviewed its lack of a direct focus on training and process improvement, and its omission of process and product measurement.

A walkthrough is normally organized and directed by the author of the technical document. Any combination of interested or technically qualified personnel (from within or outside the project) may be included as seems appropriate.

Process

A walkthrough may be quite informal, or may follow the process detailed in IEEE 1028 and outlined in the article on software reviews.

IEEE 1028 recommends three specialist roles in a walkthrough:

- The Author, who presents the software product in step-by-step manner at the walkthrough meeting, and is probably responsible for completing most action items;
- The Walkthrough Leader, who conducts the walkthrough, handles administrative tasks, and ensures orderly conduct (and who is often the Author); and

- The Recorder, who notes all anomalies (potential defects), decisions, and action items identified during the walkthrough meetings.

Difference between Verification & Validation

Verification	Validation
Are you building it right?	Are you building the right thing?
Ensure that the software system meets all the functionality.	Ensure that functionalities meet the intended behavior.
Verification takes place first and includes the checking for documentation, code etc.	Validation occurs after verification and mainly involves the checking of the overall product.
Done by developers.	Done by Testers.
Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not.	Have dynamic activities as it includes executing the software against the requirements.
It is an objective process and no subjective decision should be needed to verify the Software.	It is a subjective process and involves subjective decisions on how well the Software works.

Unit-4

Software Testing Process

Topic to be covered

Testing Process

a. *Plan*
b. *Develop*
c. *Execute*
d. *manage*

Unit-5

Software testing Strategies

Topics to be covered

- Conventional software Architecture
- Strategic Issues
- Test strategies for conventional software
  - A) Unit Testing
  - B) Integration Testing
    1. Top-down integration
    2. Bottom-Up Integration