SOFTWARE TESTING
UNIT 1- QUALITY

Dr. Koyel Datta Gupta

OUTLINE

- SOFTWARE QUALITY
- FIVE VIEWS OF SOFTWARE QUALITY
- McCall’s QUALITY FACTORS AND CRITERIA
- ISO 9126 QUALITY MODEL
- SOFTWARE QUALITY MANAGEMENT
- QUALITY MANAGEMENT ACTIVITIES
- QUALITY MANAGEMENT METHODOLOGIES
- QUALITY COST

SOFTWARE QUALITY

- Quality means that a product satisfies the demands of its specifications
- It also means achieving a high level of customer satisfaction with the product
- In software systems this is difficult
  - customer quality requirements (e.g. efficiency or reliability) often conflict with developer quality requirements (e.g. maintainability or reusability)
  - software specifications are often incomplete, inconsistent, or ambiguous

SOFTWARE QUALITY

- **Quality of conformance**
  - degree to which design specifications are followed in manufacturing the product
- **Quality control**
  - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications
- **Quality assurance**
  - auditing and reporting procedures used to provide management with data needed to make proactive decisions

A software **quality analyst** is responsible for applying the principles and practices of software **quality assurance** throughout the software development life cycle. Though often referred to as "**quality assurance**", software testing is considered to be only one part of the larger process of reducing errors.

FIVE VIEWS OF SOFTWARE QUALITY

- Transcendental view
- User view
- Manufacturing view
- Product view
- Value-based view

FIVE VIEWS OF SOFTWARE QUALITY

- Transcendental view
  - Quality is something that can be recognized through experience, but not defined in some tractable form.
  - A good quality object stands out, and it is easily recognized.
- User view
  - Quality concerns the extent to which a product meets user needs and expectations.
  - Is a product fit for use?
  - This view is highly personalized.
    - A product is of good quality if it satisfies a large number of users.
    - It is useful to identify the product attributes which the users consider to be important.
  - This view may encompass many subject elements, such as *usability*, *reliability*, and *efficiency*.

FIVE VIEWS OF SOFTWARE QUALITY

- Manufacturing view
  - This view has its genesis in the manufacturing industry – auto and electronics.
  - Key idea: Does a product satisfy the requirements?
    - Any deviation from the requirements is seen as reducing the quality of the product.
  - The concept of process plays a key role.
  - Products are manufactured “right the first time” so that the cost is reduced
    - Development cost
    - Maintenance cost
  - Conformance to requirements leads to uniformity in products.
  - Some argue that such uniformity does not guarantee quality.
  - Product quality can be incrementally improved by improving the process.
    - The CMM and ISO 9001 models are based on the manufacturing view.

FIVE VIEWS OF SOFTWARE QUALITY

- Product view
  - Hypothesis: If a product is manufactured with good internal properties, then it will have good external properties.
  - One can explore the causal relationship between *internal properties* and *external qualities*.
  - Example: *Modularity* enables *testability*.

- Value-based view
  - This represents the merger of two concepts: *excellence* and *worth*.
  - Quality is a measure of excellence, and value is a measure of worth.
  - Central idea
    - How much a customer is willing to pay for a certain level of quality.
    - Quality is meaningless if a product does not make economic sense.
    - The value-based view makes a trade-off between cost and quality.

FIVE VIEWS OF SOFTWARE QUALITY

- Measuring quality
  - Reasons for developing a quantitative view of quality
  - Measurement of user’s view
  - Measurement of manufacturing view
- Reasons for developing a quantitative view of quality
  - Measurement allows us to establish baselines for qualities.
  - Measurement is key to process improvement.
  - The needs for improvements can be investigated after performing measurements.

FIVE VIEWS OF SOFTWARE QUALITY

- Measurement of user’s view
  - Functionalities can be easily measured.
    - What functionality test cases have passed?
    - Measure of delivered functionality = ratio of # of passed test cases to the total number of test cases designed to verify the functionalities.
  - Apply Gilb’s technique.
    - The quality concept is broken down into component parts until each can be stated in terms of directly measurable qualities.
    - Example: *Usability* can be broken down into
      - *Learnability*
      - *Understandability*
      - *Operability*

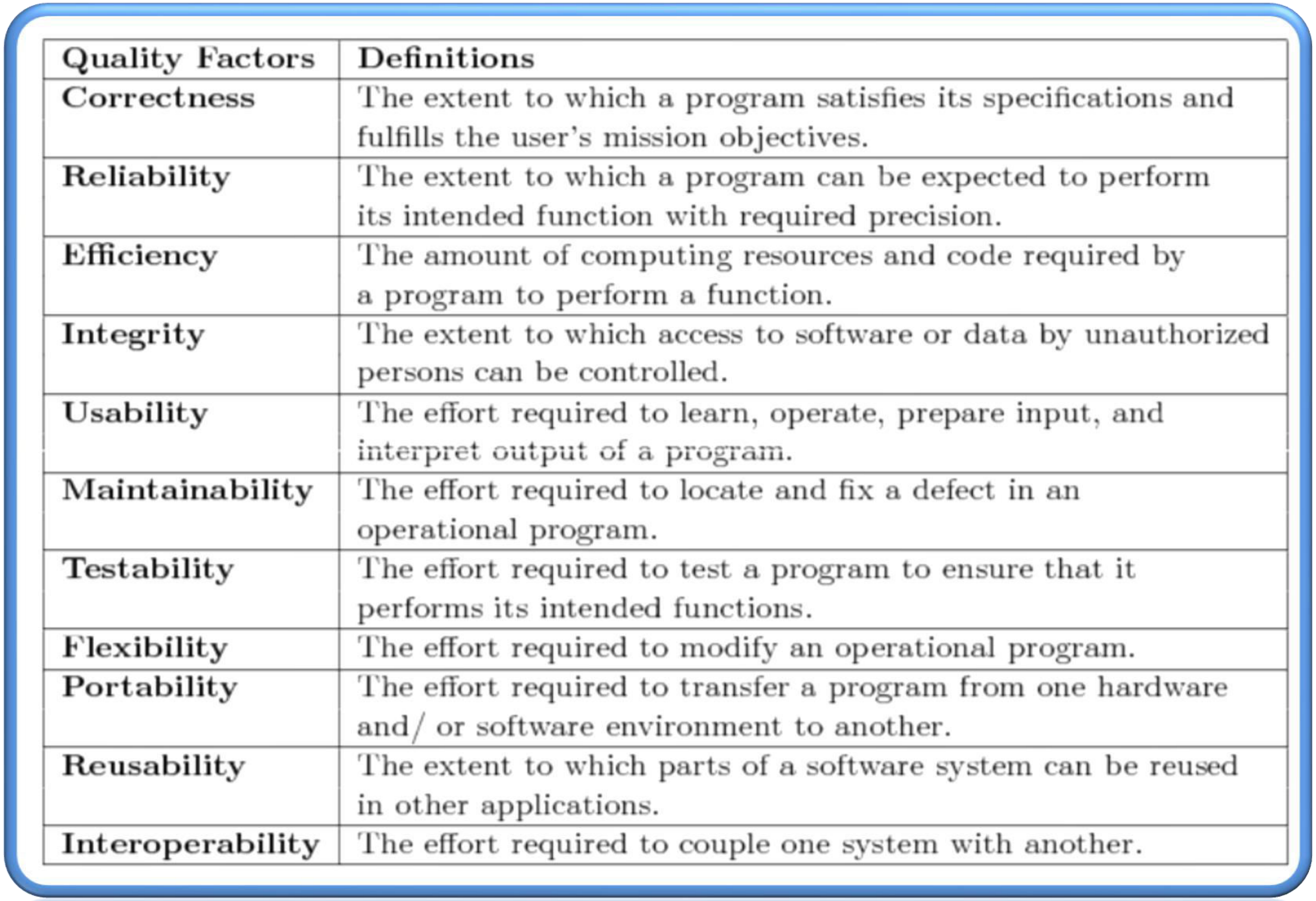
FIVE VIEWS OF SOFTWARE QUALITY

- Measurement of manufacturer's view
  - Manufacturers are interested in *defect count and rework cost.*
  - *Defect count:* The total number of defects detected during development and operation.
    - It is a measure of the quality of the work produced.
    - One can analyze the defects as follows.
      - For each defect, identify the development phase in which it was introduced.
      - Categorize the defects, say, based on modules.
      - Normalize defect count by product size.
        - » Defect density: Number of defects per 1000 lines of code.
      - Separate the defects found during operation from those found during development.
  - *Rework cost:* How much does it cost to fix the known defects?
    - Development rework cost: This is the rework cost incurred *before* a product is released. This is a measure of *development efficiency*.
    - Operation rework cost: This is the rework cost incurred *when* a product is in operation. This is a measure of the *delivered quality*.

McCall’s QUALITY FACTORS AND CRITERIA

- Quality Factors
  - McCall, Richards, and Walters studied the concept of software quality in terms of two key concepts as follows:
    - *quality factors*, and
    - *quality criteria*.
  - A quality factor represents the behavioral characteristic of a system.
    - Examples: correctness, reliability, efficiency, testability, portability, …
  - A quality criterion is an attribute of a quality factor that is related to software development.
    - Example:
      - Modularity is an attribute of the architecture of a software system.
      - A highly modular software allows designers to put cohesive components in one module, thereby increasing the maintainability of the system.
  - McCall et al. identified 11 quality factors

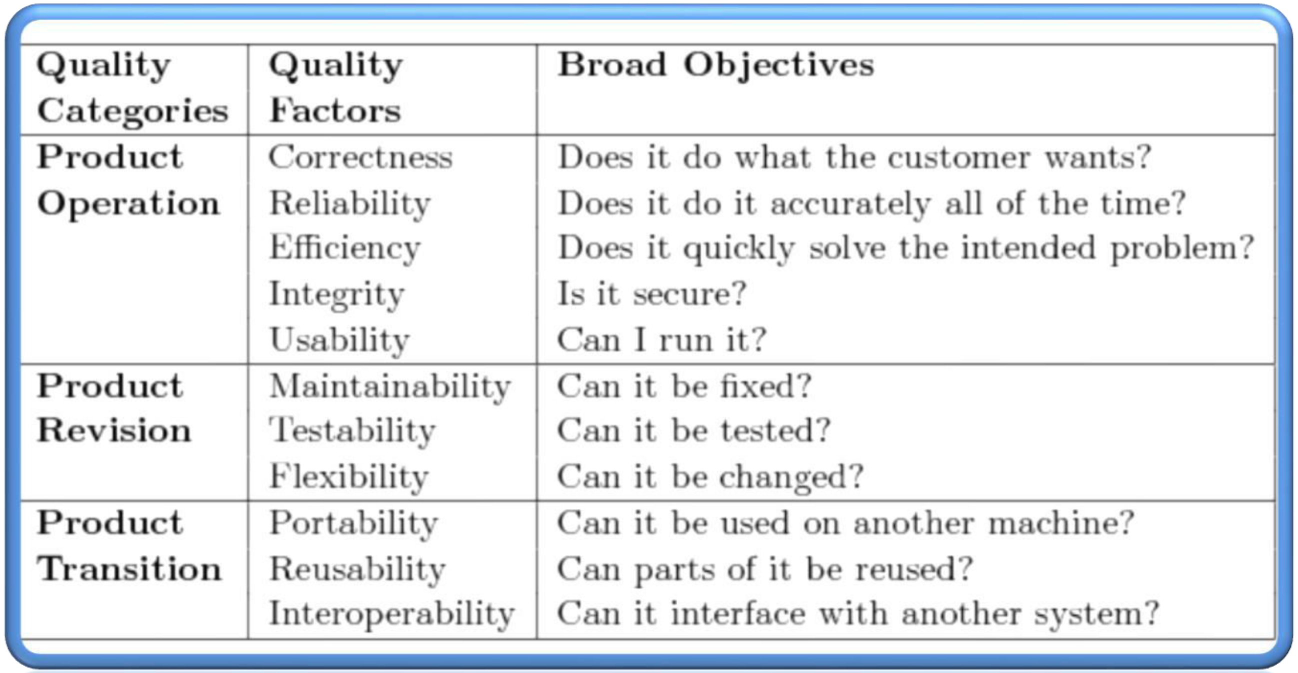
McCall's QUALITY FACTORS AND CRITERIA



Quality Factors	Definitions
Correctness	The extent to which a program satisfies its specifications and fulfills the user's mission objectives.
Reliability	The extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform a function.
Integrity	The extent to which access to software or data by unauthorized persons can be controlled.
Usability	The effort required to learn, operate, prepare input, and interpret output of a program.
Maintainability	The effort required to locate and fix a defect in an operational program.
Testability	The effort required to test a program to ensure that it performs its intended functions.
Flexibility	The effort required to modify an operational program.
Portability	The effort required to transfer a program from one hardware and/ or software environment to another.
Reusability	The extent to which parts of a software system can be reused in other applications.
Interoperability	The effort required to couple one system with another.

McCall's QUALITY FACTORS AND CRITERIA

- The 11 quality factors defined in have been grouped into three broad categories
  - Product operation
  - Product revision
  - Product transition



Quality Categories	Quality Factors	Broad Objectives
Product Operation	Correctness	Does it do what the customer wants?
	Reliability	Does it do it accurately all of the time?
	Efficiency	Does it quickly solve the intended problem?
	Integrity	Is it secure?
	Usability	Can I run it?
Product Revision	Maintainability	Can it be fixed?
	Testability	Can it be tested?
	Flexibility	Can it be changed?
Product Transition	Portability	Can it be used on another machine?
	Reusability	Can parts of it be reused?
	Interoperability	Can it interface with another system?

ISO 9126 QUALITY MODEL

Characteristics	Sub-characteristics	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 subcharacteristic addresses the compliant capability of software.
	Security	This subcharacteristic relates to unauthorized access to the software functions.

ISO 9126 QUALITY MODEL

Characteristics	Sub-characteristics	Definitions
Reliability	Maturity	This subcharacteristic 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.

ISO 9126 QUALITY MODEL

Characteristics	Sub-characteristics	Definitions
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 behaviour	Characterizes response times for a given thru put, i.e. transaction rate.
	Resource behaviour	Characterizes resources used, i.e. memory, cpu, disk and network usage.

ISO 9126 QUALITY MODEL

Characteristics	**Sub-characteristics**	**Definitions**
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.

SOFTWARE QUALITY MANAGEMENT

- QMS is responsible for ensuring that the required level of quality is achieved in a software product
- It involves defining appropriate quality standards and procedures and ensuring that these are followed
- QMS should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility

QUALITY MANAGEMENT ACTIVITIES

- Quality assurance
  - Establish organisational procedures and standards for quality
- Quality planning
  - Select applicable procedures and standards for a particular project and modify these as required
- Quality control
  - Ensure that procedures and standards are followed by the software development team

Quality management should be separate from project management to ensure independence

QUALITY MANAGEMENT METHODOLOGIES

- Standard Waterfall SDL based on the standards laid down by the IEEE
- CMMI (Capability Maturity Model Integration) based on the guideline of the SEI
- V-model
- Incremental model
- RUP (Rational Unified Process)
- Agile Method of Software Development
- Test-based development method
- Rapid Action Development model

QUALITY COST

- Includes all costs incurred in the pursuit of quality or in performing quality-related activities
- QC is analyzed to
  - provide a baseline for the current cost of quality
  - identify opportunities for reducing the cost of quality
  - provide a normalized basis of comparison

TYPES OF QUALITY COSTS

- **Prevention costs**
  - Quality planning, formal technical reviews, test equipment, training
- **Appraisal costs**
  - Inspections, equipment calibration and maintenance, testing
- **Failure costs** – subdivided into internal failure costs and external failure costs
  - ➢ **Internal failure costs**
    - Incurred when an error is detected in a product prior to shipment
    - Include rework, repair, and failure mode analysis
  - ➢ **External failure costs**
    - Involves defects found after the product has been shipped
    - Include complaint resolution, product return and replacement, help line support, and warranty work

SOFTWARE TESTING
UNIT 1- TESTING

Dr. Koyel Datta Gupta

OUTLINE

- SOFTWARE TESTING & ITS OBJECTIVES
- PRINCIPLES OF SOFTWARE TESTING
- TESTER ROLE IN SOFTWARE DEVELOPMENT
- FAULTS, ERRORS, AND FAILURES
- LIMITATIONS OF TESTING
- CHALLENGES IN SOFTWARE TESTING
- TESTING AND DEBUGGING
- VERIFICATION & VALIDATION
- TEST LEVELS

SOFTWARE TESTING

- Testing is the process of executing a program with the intention of finding errors." – Myers
- Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. This activity results in the actual, expected and difference between their results.
- 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".

SOFTWARE TESTING -OBJECTIVES

- Executing a program with the intent of finding an *error*.
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- To check if the system is “ Fit for purpose”.
- To check if the system does what it is expected to do.

SOFTWARE TESTING -OBJECTIVES

- A good test case is one that has a probability of finding an as yet undiscovered error.
- A successful test is one that uncovers a yet undiscovered error.
- A good test is not redundant.
- A good test should be “best of breed”.
- A good test should neither be too simple nor too complex.

PRINCIPLES OF SOFTWARE TESTING

1) Testing shows presence of defects

2) Exhaustive testing is impossible

3) Early testing

4) Defect clustering

PRINCIPLES OF SOFTWARE TESTING

5) Pesticide paradox

6) Testing is context depending

7) Absence – of – errors fallacy

TESTER ROLE
IN
SOFTWARE DEVELOPMENT

- Find bugs as early as possible and make sure they get fixed.
- To understand the application well.
- Study the functionality in detail to find where the bugs are likely to occur.
- Study the code to ensure that each and every line of code is tested.
- Create test cases in such a way that testing is done to uncover the hidden bugs and also ensure that the software is usable and reliable

FAULTS, ERRORS, AND FAILURES

- **Fault** : It is a condition that causes the software to fail to perform its required function.

  Error : Refers to difference between Actual Output and Expected output.

  Failure : It is the inability of a system or component to perform required function according to its specification.

FAULTS, ERRORS, AND FAILURES

IEEE Definitions

Failure: External behaviour is incorrect

- **Fault:** Discrepancy in code that causes a failure.
- **Error:** Human mistake that caused fault

- **Error** is terminology of Developer.
- **Bug** is terminology of Tester

LIMITATIONS OF TESTING

1) Testing can be used to show the presence of errors, but never to show their absence.
2) Software testing does not help in finding root causes which resulted in injection of defects in the first place.
3) Exhaustive (total) testing is impossible in present scenario.
4) Time and budget constraints normally require very careful planning of the testing effort.
5) Compromise between thoroughness and budget.
6) Test results are used to make business decisions for release dates.
7) Time will run out before the testing of all test cases.
8) Testing every true and false condition is impossible.
9) Testing every condition of a decision node is impossible.
10) Testing every path is impossible.
11) Testing every valid input is impossible.
12) Testing every invalid input is impossible.

CHALLENGES IN SOFTWARE TESTING

1. Complete testing is impossible
   1. *There is no simple answer for this.*
   2. *Therefore testers live and breathe tradeoffs.*
2. Testers misallocate resources because they fall for the company’s process myths
3. Test groups operate under multiple missions, often conflicting, rarely articulated
4. Test groups often lack skilled programmers, and a vision of appropriate projects that would keep programming testers challenged
5. Regression testing
6. Unavailability of the best Tools

TESTING AND DEBUGGING

- **Debugging** is the activity which is carried out by the development team (or developer), after getting the test report from the testing team about defect(s).
- The developer then tries to find the cause of the defect, in this quest he may need to go through lines of code and find which part of code in causing that defect.
- After finding out the bug, the developer tries to modify that portion of code and then rechecks if the defect has been finally removed.
- After fixing the bug, developers send the software back to testers.

TESTING AND DEBUGGING

Testing	Debugging
1. Testing always starts with known conditions, uses predefined methods, and has predictable outcomes too.	1. Debugging starts from possibly un-known initial conditions and its end cannot be predicted, apart from statistically.
2. Testing can and should definitely be planned, designed, and scheduled.	2. The procedures for, and period of, debugging cannot be so constrained.
3. It proves a programmers failure.	3. It is the programmer's vindication.
4. It is a demonstration of error or apparent correctness.	4. It is always treated as a deductive process.
5. Testing as executed should strive to be predictable, dull, constrained, rigid, and inhuman.	5. Debugging demands intuitive leaps, conjectures, experimentation, and some freedom also.
6. Much of the testing can be done without design knowledge.	6. Debugging is impossible without detailed design knowledge.
7. It can often be done by an outsider.	7. It must be done by an insider.
8. Much of test execution and design can be automated.	8. Automated debugging is still a dream for programmers.
9. Testing purpose is to find bug.	9. Debugging purpose is to find cause of bug.

VERIFICATION & VALIDATION

Verification - typically involves reviews and meeting to evaluate documents, plans, code, requirements, and specifications. This can be done with checklists, issues lists, walkthroughs, and inspection meeting.

Validation - typically involves actual testing and takes place after verifications are completed.

Validation and Verification process continue in a cycle till the software becomes defects free.

VERIFICATION & VALIDATION

Verification	Validation
1. Are you building it right?	1. Are you building the right thing?
2. Ensure that the software system meets all the functionality.	2. Ensure that functionalities meet the intended behaviour.
3. Verification takes place first and includes the checking for documentation, code etc.	3. Validation occurs after verification and mainly involves the checking of the overall product.
4. Done by developers.	4. Done by Testers.
5. Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not.	5. Have dynamic activities as it includes executing the software against the requirements.
6. It is an objective process and no subjective decision should be needed to verify the Software.	6. It is a subjective process and involves subjective decisions on how well the Software works.

TEST LEVELS

- Unit testing
- Integration testing
- System testing
- Acceptance testing