Internal Assesment Test – I, September 2020									
Sub:	Software Testing							Code:	18MCA351
Date:	16-09-2020	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MCA

	Answer Any 5 **QUESTION**s	Marks	OBE	
			CO	RBT
1a	List out quality attributes of software and explain each of them	05	CO1	L2
1 b	Differentiate between verification and validation	05	CO1	L1
2 a	Discuss how levels of testing are associated with levels of software development. Draw a supporting diagram.	06	CO2	L1
2b	Explain the difference between error, fault, failure and incident.	04	CO2	L2
3	What is Test Oracle ? Discuss the defect life-cycle and draw an appropriate diagram.	2+8	CO1	L2
4	Compare specification-based testing and code-based testing	10	CO2	L3
5	List down the different fault types and give two examples of each.	10	CO2	L3
6	Compare specification, program development and testing phases with help of Venn Diagram	10	CO2	L2
7	Explain the Test Case Generation Strategies	10	CO2	L4
8	Discuss Test-Debug cycle with the help of a diagram	10	CO1	L1

1 a. SOFTWARE QUALITY

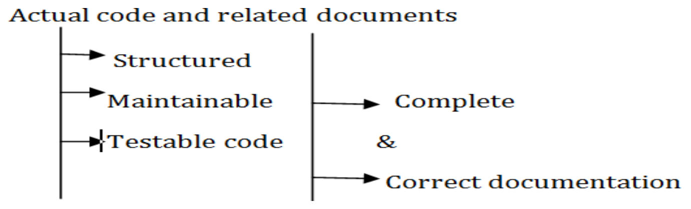
- Software quality is a multidimensional quantity and is measurable.

Quality Attributes

- These can be divided to static and dynamic quality attributes.

Static quality attributes

- It refers to the actual code and related documents.



Example: A poorly documented piece of code will be harder to understand and hence difficult to modify. A poorly structured code might be harder to modify and difficult to test.

Dynamic quality Attributes:

- Reliability
- Correctness

- Completeness
- Consistency
- Usability
- performance

Reliability:

- It refers to the probability of failure free operation.

Correctness:

- Refers to the correct operation and is always with reference to some artefact.
- For a Tester, correctness is w.r.t to the requirements
- For a user correctness is w.r.t the user manual

Completeness:

- Refers to the availability of all the features listed in the requirements or in the user manual.
- An incomplete software is one that does not fuly implement all features required.

Consistency:

- Refers to adherence to a common set of conventions and assumptions.
- Ex: All buttons in the user interface might follow a common-color coding convention.

Usability:

- Refer to ease with which an application can be used. This is an area in itself and there exist techniques for usability testing.
- Psychology plays an important role in the design of techniques for usability testing.
- Usability testing is a testing done by its potential users.
- The development organization invites a selected set of potential users and asks them to test the product.
- Users in turn test for ease of use, functionality as expected, performance, safety and security.
- Users thus serve as an important source of tests that developers or testers within the organization might not have conceived.
- Usability testing is sometimes referred to as user-centric testing.

Performance:

- Refers to the time the application takes to perform a requested task. Performance is considered as a non-functional requirement.

Reliability:

- (Software reliability is the probability of failure free operation of software over a given time interval & under given conditions.)
- Software reliability can vary from one operational profile to another. An implication is that one might say “this program is lousy” while another might sing praises for the same program.
- Software reliability is the probability of failure free operation of software in its intended environments.
- The term environment refers to the software and hardware elements needed to

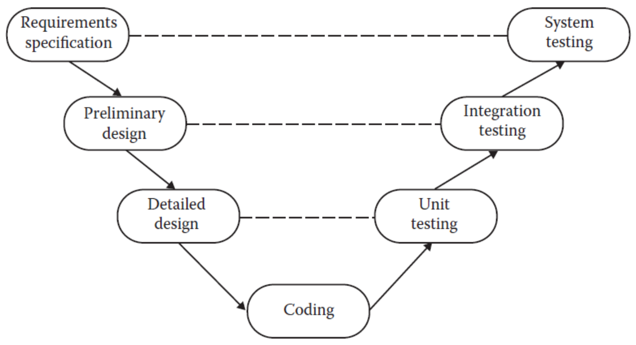
execute the application. These elements include the operating system(OS)hardware requirements and any other applications needed for communication.

1.b. Differences between Verification and Validation

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. Verification means **Are we building the product right?**
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing. Validation means **Are we building the right product?**
The difference between Verification and Validation is as follow:

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

2 a. **Discuss how levels of testing are associated with levels of software development. Draw a supporting diagram**.



Levels of testing echo the levels of abstraction found in the waterfall model of the software development life cycle. Although this model has its drawbacks, it is useful for testing as a means of identifying distinct levels of testing and for clarifying the objectives that pertain to each level. A diagrammatic variation of the waterfall model, known as the V-Model in ISTQB parlance, is given in Figure; this variation emphasizes the correspondence between testing and design levels. Notice that, especially in terms of specification-based testing, the three levels of definition (specification, preliminary design, and detailed design) correspond directly to three levels of testing— system, integration, and unit testing. A practical relationship exists between levels of testing versus specification-based and code based testing. Most practitioners agree that code-based testing is most appropriate at the unit level, whereas specification-based testing is most appropriate at the system level. This is generally true; however, it is also a likely consequence of the base information produced during the requirements specification, preliminary design, and detailed design phases. The constructs defined for code-based testing make the most sense at the unit level, and similar constructs are only now becoming available for the integration and system levels of testing.

***2b.* Explain the difference between error, fault, failure and incident.**

Error—People make errors. A good synonym is *mistake*. When people make mistakes while coding, we call these mistakes *bugs*. Errors tend to propagate; a requirements error may be magnified during design and amplified still more during coding.
Fault—A fault is the result of an error. It is more precise to say that a fault is the representation of an error, where representation is the mode of expression, such as narrative text, Unified Modeling Language diagrams, hierarchy charts, and source code. *Defect* is a good synonym for fault, as is *bug*. Faults can be elusive. An error of omission results in a fault in which something is missing that should be present in the representation. This suggests a useful refinement; we might speak of faults of commission and faults of omission. A fault of commission occurs when we enter something into a representation that is incorrect. Faults of omission occur when we fail to enter correct information. Of these two types, faults of omission are more difficult to detect and resolve.
Failure—A failure occurs when the code corresponding to a fault executes. Two subtleties arise here: one is that failures only occur in an executable representation, which is usually taken to be source code, or more precisely, loaded object code; the second subtlety is that this definition relates failures only to faults of commission. How can we deal with failures that correspond to faults of omission? We can push this still further: what about faults that never happen to execute, or perhaps do not execute for a long time? Reviews prevent many failures by finding faults; in fact, well-done reviews can find faults of omission.
Incident—When a failure occurs, it may or may not be readily apparent to the user (or customer or tester). An incident is the symptom associated with a failure that alerts the user to the occurrence of a failure.

3. Oracles can also be programs designed to check the behavior of other programs.
For example:

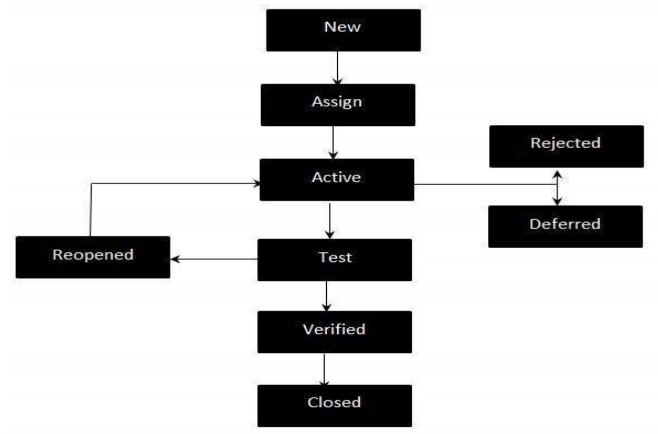
one might use a matrix multiplication program to check if a matrix inversion program has produced the correct output.

In this case, the matrix inversion program inverts a given matrix A and generates B as the output matrix.

Construction of automated oracles, such as the one to check a matrix multiplication program or a sort program, requires the determination of input output relationship.

In general, the construction of automated oracles is a complex undertaking.

Discuss the defect life-cycle and draw an appropriate diagram



Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.

Defect Life Cycle States:

- **New -** Potential defect that is raised and yet to be validated.
- **Assigned -** Assigned against a development team to address it but not yet resolved.
- **Active -** The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
- **Test -** The Defect is fixed and ready for testing.
- **Verified -** The Defect that is retested and the test has been verified by QA.
- **Closed -** The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
- **Reopened -** When the defect is NOT fixed, QA reopens/reactivates the defect.
- **Deferred -** When a defect cannot be addressed in that particular cycle it is deferred to future release.

Rejected - A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

4. Compare specification-based testing and code-based testing.

The Differences Between Black Box Testing (or specification-based testing) and White Box Testing (or code-based testing) are listed below.

Criteria	**Black Box Testing**	**White Box Testing**
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing: Acceptance Testing , System Testing	Mainly applicable to lower levels of testing: Unit Testing, Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

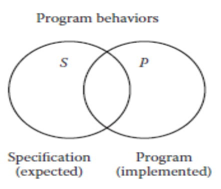
5. List down the different fault types and give two examples of each.

Our definitions of error and fault hinge on the distinction between process and product: process refers to how we do something, and product is the end result of a process. The point at which testing and Software Quality Assurance (SQA) meet is that SQA typically tries to improve the product by improving the process. In that sense, testing is clearly more product oriented. SQA is more concerned with reducing errors endemic in the development process, whereas testing is more concerned with discovering faults in a product. Both disciplines benefit from a clearer definition of types of faults. Faults can be classified in several ways: the development phase in which the corresponding error occurred, the consequences of corresponding failures, difficulty to resolve, risk
of no resolution, and so on. My favorite is based on anomaly (fault) occurrence: one time only, intermittent, recurring, or repeatable. For a comprehensive treatment of types of faults, see the IEEE Standard Classification for Software Anomalies (IEEE, 1993). (A software anomaly is defined in that document as
“a departure from the expected,” which is pretty close to our definition.) The IEEE standard defines a detailed anomaly resolution process built around four phases (another life cycle): recognition, investigation, action, and disposition.

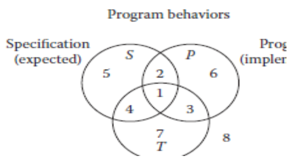
Fault Types:

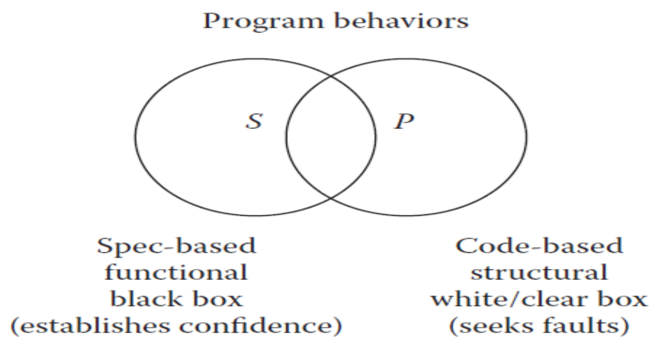
Input/Output Faults
ICorrect input not accepted
Incorrect input accepted
Output Wrong format
Wrong result
Cosmetic
Logic Faults
Missing case(s)
Duplicate case(s)
Extreme condition neglected
Wrong operator (e.g., < instead of ≤)
Computation Faults
Incorrect algorithm
Missing computation
Incorrect operand
Incorrect operation
Interface Faults
Incorrect interrupt handling
I/O timing
Call to wrong procedure
Call to nonexistent procedure
Parameter mismatch (type, number)
Incompatible types
Superfluous inclusion
Data Faults
Incorrect initialization
Incorrect storage/access
Wrong flag/index value
Incorrect packing/unpacking
Wrong variable used

6 Compare specification, program development and testing phases with help of Venn Diagram



Specified and implemented program behaviors.





- *Functional testing uses only the specifications to identify test cases.*
- *Structural testing uses the program source code as the basis of test case identification.*
- *Neither approach alone is sufficient.*
- *Consider program behaviors: if all specified have not been implemented, structural test cases will never be able to recognize this.*
- *Program implements behaviors that have not been specified, this will never be revealed by functional test cases.*
- *Both the approaches are needed to provide the confidence. Here, Problems can be recognized and resolved. The Venn diagram provides final insight.*

7. Test case Generation

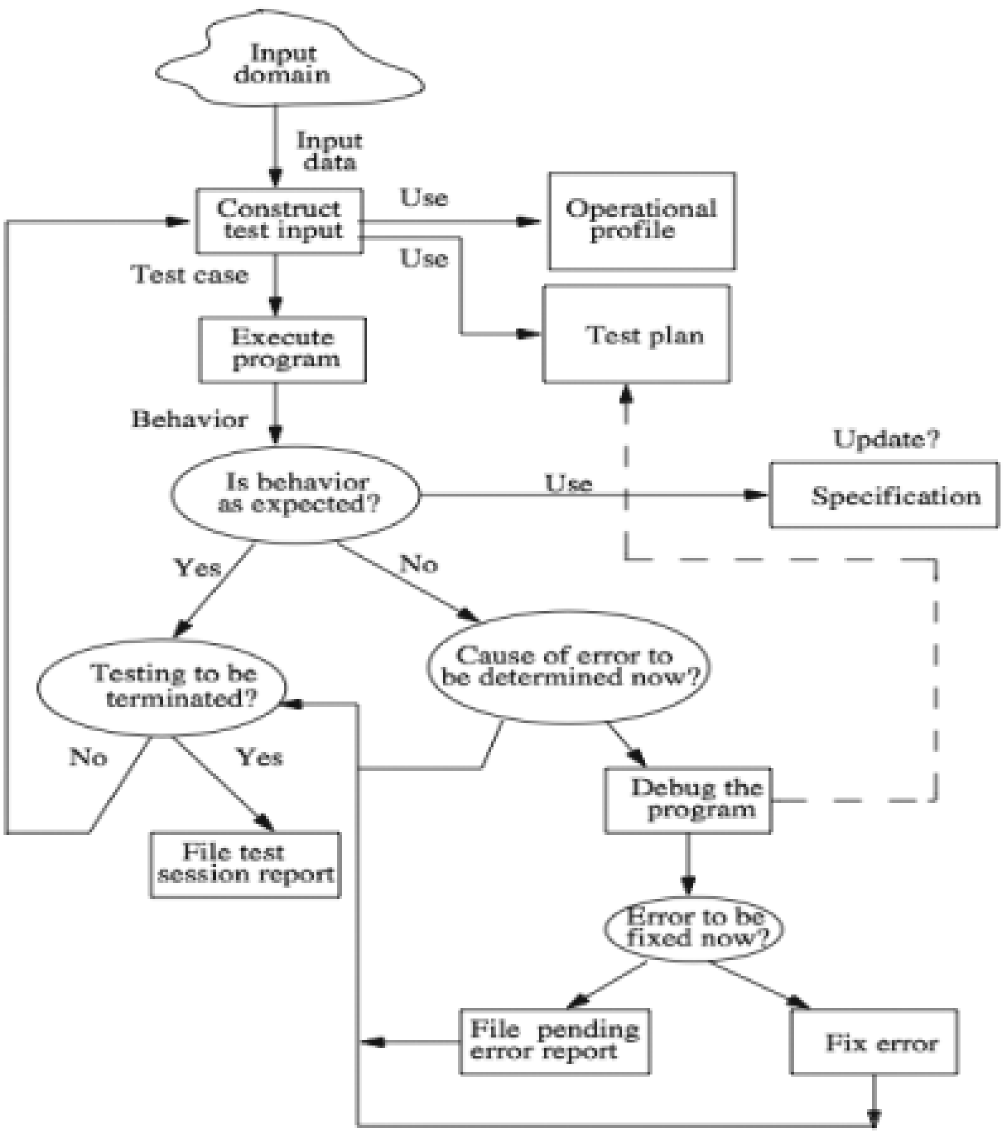
*A **TEST CASE** is a documented set of **preconditions** (prerequisites), **procedures** (inputs / actions) and **postconditions** (expected results) which a tester uses to determine whether a system under test satisfies requirements or works correctly. A **test case** can have one or multiple **test scripts** and a collection of test cases is called a **test suite.***

Test Suite ID	*TS001*
Test Case ID	*TC001*
Test Case Summary	*To verify that clicking the Generate Coin button generates coins.*
Related Requirement	*RS001*
Prerequisites	*1. User is authorized.* *2. Coin balance is available.*
Test Script / Procedure	*1. Select the coin denomination in the Denomination field.* *2. Enter the number of coins in the Quantity field.* *3. Click Generate Coin.*
Test Data	*1. Denominations: 0.05, 0.10, 0.25, 0.50, 1, 2, 5* *2. Quantities: 0, 1, 5, 10, 20*
Expected Result	*1. Coin of the specified denomination should be produced if the specified Quantity is valid (1, 5)* *2. A message 'Please enter a valid quantity between 1 and 10' should be displayed if the specified quantity is invalid.*

Actual Result	1. *If the specified quantity is valid, the result is as expected.* 2. *If the specified quantity is invalid, nothing happens; the expected message is not displayed*
Status	*Fail*
Remarks	*This is a sample test case.*
Created By	*John Doe*
Date of Creation	*01/14/2020*
Executed By	*Jane Roe*
Date of Execution	*02/16/2020*
Test Environment	• *OS: Windows Y* • *Browser: Chrome N*

8. Testing Debugging Cycle

1. ***P**reparing Test Plan*
2. ***C**onstructing Test Data*
3. ***E**xecuting Program*
4. ***S**pecifying program behavior*
5. ***A**ccessing Correctness of the Program*
6. ***C**onstruction of Oracles*



Input domain
Input data
Construct test input
Use
Operational profile
Use
Test case
Test plan
Execute program
Behavior
Update?
Is behavior as expected?
Use
Specification
Yes
No
Testing to be terminated?
Cause of error to be determined now?
No
Yes
Debug the program
File test session report
Error to be fixed now?
File pending error report
Fix error