11718

3 Hours / 100 Marks   Seat No.

Instructions :   (1)   All Questions are *compulsory*.

(2)   Answer each next main Question on a new page.

(3)   Illustrate your answers with neat sketches wherever necessary.

(4)   Figures to the right indicate full marks.

(5)   Assume suitable data, if necessary.

(6)   Mobile Phone, Pager and any other Electronic Communication devices are not permissible in Examination Hall.

                                                                                                     Marks

1.   (a)   Attempt any THREE of the following :   **12**

(i)   What is Software testing ? State objectives of Software testing.

(ii)   What is Black Box testing ? List any four techniques of Black Box testing.

(iii) Explain the Regression testing. State when the Regression testing shall be done.

(iv)   Write any four test cases to test login form.

(b)   Attempt any ONE of the following :   **6**

(i)   Describe use of load testing and stress testing to test online result display facility of MSBTE website.

(ii)   What are the points considered while estimating impact of a defect ? Also explain techniques to find defect.

2. **Attempt any FOUR of the following :** 16

   (i) What is White Box testing ? Explain any one technique of static White Box testing.

   (ii) How to select a testing tool ? Explain in detail.

   (iii) Enlist components of usability testing. Which features of Software are tested in usability testing ?

   (iv) Define Metrics and Measurements. Explain need of Software measurement.

   (v) Explain Defect Management Process.

   (vi) State any four objectives of user documentation testing. How these are useful in planning user documentation test ?

3. **Attempt any FOUR of the following :** 16

   (i) With the help of suitable example explain decision table. Why decision table is important ?

   (ii) Explain GUI testing with suitable example.

   (iii) Explain the 'Test Infrastructure' components with diagram.

   (iv) Give comparison between Alpha testing and Beta testing. (any four points)

   (v) Write down any four limitations of manual testing.

4. (a) **Attempt any THREE of the following :** 12

   (i) What is test case ? Which parameters are to be considered while documenting test cases ?

   (ii) What is automated testing ? Write down advantages of using automated testing tools in software testing.

   (iii) Explain defect life cycle to identify status of defect with proper labelled diagram.

   (iv) Explain test deliverables in details.

(b) **Attempt any ONE of the following :** **6**

(i) Describe V-model with labelled diagram. State its any two advantages and disadvantages. Also write where it is applicable.

(ii) With the help of example explain Boundary Value Analysis.

5. **Attempt any TWO of the following :** **16**

(i) Prepare any eight test cases for admission form of college admission.

(ii) With the suitable example, explain how 'Basis Path Testing' is used to derive the code complexity for the testing.

(iii) What is Integration testing ? List types of Integration testing and explain any four types in brief.

6. **Attempt any FOUR of the following :** **16**

(i) Which parameters are considered while writing good defect report ? Also write contents of defect template.

(ii) Define the terms Error : Defect, Fault and Bug in relation with Software testing.

(iii) Give the defect classification and its meaning.

(iv) What is test planning and test management ?

(v) With the help of diagram describe client-server testing.

_____

WINTER– 17 EXAMINATION

Subject Name: Software Testing   **Model Answer**   **Subject Code:** | 17624 |

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1.	(a)	**Attempt any THREE of the following:**	12 Marks
	(i)	**What is Software testing? State objectives of Software testing.**	4M
	Ans:	**Software testing:** Software testing: software testing is defined as performing Verification and Validation of the Software Product for its correctness and accuracy of working. Software Testing is the process of executing a program with the intent of finding errors. A successful test is one that uncovers an as-yet-undiscovered error. Testing can show the presence of bugs but never their absence. Testing is a support function that helps developers look good by finding their mistakes before anyone else does. Execution of a work product with intent to find a defect. **Objectives of software testing are as follows**: 1. Finding defects which may get created by the programmer while developing the software. 2. Gaining confidence in and providing information about the level of quality. 3. To prevent defects. 4. To make sure that the end result meets the business and user requirements. 5. To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications. 6. To gain the confidence of the customers by providing them a quality product.	(Definition :1 mark, any three objectives :3 marks)

(ii)	**What is Black Box testing? List any four techniques of Black Box testing.**	**4M**
Ans:	Black Box testing involves looking at the specifications and does not require examining the code of the program. It is done from customer's point of view. The testers know the input and expected output. They will check whether with given input they are getting expected output or not. **Different techniques of Black Box test are:** 1. Requirement base testing 2. Positive negative testing 3. Boundary value analysis 4. Decision tables 5. Equivalence partitioning 6. State based testing 7. Compatibility testing 8. User documentation testing 9. Domain testing	**(Black box testing:2 marks,Listing any four techniques:2 marks)**
(iii)	**Explain the Regression testing. State when the Regression testing shall be done.**	**4M**
Ans:	Regression testing a black box testing technique that consists of re-executing those tests that are impacted by the code changes. These tests should be executed as often as possible throughout the software development life cycle. It is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers. A normal regression testing is performed to verify if the build has not broken any other parts of the application by the recent code changes for defect fixing or for enhancement. It finds other related bugs. It tests to check the effect on other parts of the program. Regression testing produces Quality software**.** Validate the parts of software where changes occur. It validates parts of software which may be affected by some changes but otherwise unrelated. It ensures proper functioning of the software, as it was before changes occurred. It enhances quality of software, as it reduces the high risk bugs. Regression testing is useful when there are updates expected in product. It is helpful to check overall functionality in changing situations.	**(Regression Testing: 2 marks, two Reason why regression testing is done:2 marks)**

	(iv)	**Write any four test cases to test login form.**					**4M**
	Ans:	{**Note: any other relevant test cases shall also be considered**} Test cases for login page are as given below:					(Any valid **Four test cases for Login form: 1 mark each)**

Test cases for login page are as given below:

Step	Test step	Test data	Expected output	Actual output	Status
1	Navigate to login page of website	---	---	---	Pass
2	Provide valid username	Username abc@yahoo.co.in	Shall accept the username	Accepted user name	Pass
3	Provide password	Password co6g1234	Shall accept the password	Accepted the password	Pass
4	Click on submit	Press submit button	User should be able to login successfully	User successfully logged in	Pass
5	Go to the home page	Click on home button	Should display home page	Home page of the user displayed	Pass
6	Write the status	Type in the status in the area provided and press post	Should post the message typed in and the status of user should change	Status changed successfully	Pass

	(b)	**Attempt any ONE of the following:**	**6 Marks**
	(i)	**Describe use of load testing and stress testing to online result display facility of MSBTE website.**	**6M**
	Ans:	**Stress Testing:** In stress testing of MSBTE online result display, the resources used will be less than the requirement. For e.g. Provide less RAM for the server, or decrease the bandwidth of the internet connection, or provide less hits for page. If the system has limited resources available, the response of the online result system may deteriorate due to non-availability of the resources. It tries to break the page, site or connection under test by overwhelming its resources in order to find the circumstances under which it will crash. It is also a type of load testing. It is designed to determine the behavior of the software under abnormal situations. In stress testing test cases are designed to execute the system in such a way that abnormal conditions. **Load Testing:** When a MSBTE online result display facility is tested with a load that causes it to allocate its resources in maximum amounts. The idea is to create an environment more demanding than the application would experience under normal workloads. Eg. Apply more number of hits on the result section, try displaying multiple results in multiple browsers, etc. Load is varied from minimum to the maximum level the system can sustain without running out of resources. Load is being increased transactions may suffer excessive delays. Load testing involves simulating real-life user load for the target application. It helps to determine how application behaves when multiple students hits it simultaneously to check the results. Load	(Use of **load testing with example :3 marks, use of stress testing of MSBTE online result display: 3 marks)**

		testing can be done under controlled lab conditions to compare the capabilities of different systems or to accurately measure the capabilities of a single system.	
(ii)		**What are the points considered while estimating impact of a defect? Also explain techniques to find defect.**	**6M**
Ans:		Estimate Expected Impact of a Defect, Techniques for Finding Defects, Reporting a Defect. Once the critical risks are identified, the financial impact of each risk should be estimated. This can be done by assessing the impact, in dollars, if the risk does become a problem combined with the probability that the risk will become a problem. The product of these two numbers is the expected impact of the risk. The expected impact of a risk (E) is calculated as $E = P * I$, where: P= probability of the risk becoming a problem and I= Impact in dollars if the risk becomes a problem. Once the expected impact of each risk is identified, the risks should be prioritized by the expected impact and the degree to which the expected impact can be reduced. While guess work will constitute a major role in producing these numbers, precision is not important. What will be important is to identify the risk, and determine the risk's order of magnitude. Large, complex systems will have many critical risks. Whatever can be done to reduce the probability of each individual critical risk becoming a problem to a very small number should be done. Doing this increases the probability of a successful project by increasing the probability that none of the critical risks will become a problem. **Example:** • An organization with a project of 2,500 function points and was about medium at defect discovery and removal would have 1,650 defects remaining after all defect removal and discovery activities. • The calculation is 2,500 x 1.2 = 3,000 potential defects. • The organization would be able to remove about 45% of the defects or 1,350 defects. • The total potential defects (3,000) less the removed defects (1,350) equals the remaining defects of 1,650. **Estimate Expected Impact of a Defect :** i. There is a strong relationship between the number of test cases and the number of function points. ii. There is a strong relationship between the number of defects and the number of test cases and number of function points. iii. The number of acceptance test cases can be estimated by multiplying the number of function points by 1.2. iv. Acceptance test cases should be independent of technology and implementation techniques. v. If a software project was 100 function points the estimated number of test cases would be 120. vi. To estimate the number of potential defects is more involved.	**(Explanation of Impact of a defect : 4 marks, Techniques : 2 marks)**

Techniques to find defects:
a) Quick Attacks:
b) Equivalence and Boundary Conditions
c) Common Failure Modes
d) State-Transition Diagrams
e) Use Cases
f) Code-Based Coverage Models
g) Regression and High-Volume Test Techniques

{Note: Following explanation is optional**}**

a) Quick Attacks:
- The quick-attacks technique allows you to perform a cursory analysis of a system in a very compressed timeframe.
- Even without a specification, you know a little bit about the software, so the time spent is also time invested in developing expertise.
.

b) Equivalence and Boundary Conditions:
- Boundaries and equivalence classes give us a technique to reduce an infinite test set into something manageable.
- They also provide a mechanism for us to show that the requirements are "covered".

c) Common Failure Modes:
- The heart of this method is to figure out what failures are common for the platform, the project, or the team; then try that test again on this build.
- If your team is new, or you haven't previously tracked bugs, you can still write down defects that "feel" recurring as they occur—and start checking for them.
- The more your team stretches itself (using a new database, new programming language, new team members, etc.), riskier the project will be—and, at the same time, the less valuable this technique will be.

d) State-Transition Diagrams:
- Mapping out the application provides a list of immediate, powerful test ideas.
- Model can be improved by collaborating with the whole team to find "hidden" states—transitions that might be known only by the original programmer or specification author.
- Once you have the map, you can have other people draw their own diagrams, and then compare theirs to yours.
- The differences in those maps can indicate gaps in the requirements, defects in the software, or at least different expectations among team members.
- The map you draw doesn't actually reflect how the software will operate; in other words, "the map is not the territory."
- Drawing a diagram won't find these differences,
- Like just about every other technique on this list, a state-transition diagram can be helpful, but it's not sufficient by itself to test an entire application.

e) Use Cases:
Use cases and scenarios focus on software in its role to enable a human being to do something. Use cases and scenarios tend to resonate with business customers, and if done as part of the requirement process, they sort of magically generate test cases

from the requirements.

f) Code-Based Coverage Models:

Imagine that you have a black-box recorder that writes down every single line of code as it executes. Programmers prefer code coverage. It allows them to attach a number— an actual, hard, real number, such as 75%—to the performance of their unit tests, and they can challenge themselves to improve the score.

- Customer-level coverage tools are expensive, programmer-level tools that tend to assume the team is doing automated unit testing and has a continuous-integration server and a fair bit of discipline.
- After installing the tool, most people tend to focus on statement coverage—the least powerful of the measures.

g) Regression and High-Volume Test Techniques:

- People spend a lot of money on regression testing, taking the old test ideas described above and rerunning them over and over.
- This is generally done with either expensive users or very expensive programmers spending a lot of time writing and later maintaining those automated tests.

 ii. Weaknesses

- Building a record/playback/capture rig for a GUI can be extremely expensive, and it might be difficult to tell whether the application hasn't broken, but has changed in a minor way.

2.		Attempt any FOUR of the following:	16Marks
	(i)	**What is White Box testing? Explain any one technique of static White Box testing.**	**4M**
	Ans:	White Box Testing: Classification of Whit e Box. 1. This is also known as glass box, clear box, and open box testing. 2. In white box testing, test cases are created by looking at the code to detect any Potential failure scenarios. 3. The suitable input data for testing various APIs and the special code paths that Need to be tested by analyzing the source code for the application block. 4. Therefore, the test plans need to be updated before starting white box testing and only after a stable build of the code is available. 5. White box testing assumes that the tester can take a look at the code for the application block and create test cases that look for any potential failure scenarios. 6. During white box testing, analyze the code of the application block and prepare test cases for testing the functionality to ensure that the class is behaving in accordance with the specifications and testing for robustness. 7. A failure of a white box test may result in a change that requires all black box testing to be repeated and white box testing paths to be reviewed and possibly changed. **i. Static Testing- Inspections, Structured Walkthroughs, Technical Review:** **1. Inspection** i. Inspections are the most formal type of reviews. ii. They are highly structured and require training for each participant. iii. Inspections are different from peer reviews and walkthroughs in that the person who presents the code, the presenter or reader, isn't the original programmer. iv. These forces someone else to learn and understand the material being presented,	**(Explanation of white box testing: 2 marks, Any one technique : 2 marks)**

potentially giving a different slant and interpretation at the inspection meeting.

v. The other participants are called inspectors.

vi. Each is tasked with reviewing the code from a different perspective, such as a
   User, a tester, or a product support person.

vii. This helps bring different views of the product under review and very often
   Identifies different bugs.

viii. One inspector is even tasked with reviewing the code backward—that is, from
   the end to the beginning—to make sure that the material is covered evenly
   And completely.

2. Walkthrough:

i. Walkthroughs are the next step up in formality from peer reviews.

ii. In a walkthrough, the programmer who wrote the code formally presents
(Walks through) it to a small group of five or so other programmers and testers.

iii. The reviewers should receive copies of the software in advance of the review so
they can examine it and write comments and questions that they want to ask at the
review.

iv. Having at least one senior programmer as a reviewer is very important.

 3. Technical Review:

i. Formal Review:

- A formal review is the process under which static white-box testing is performed.
- A formal review can range from a simple meeting between two programmers to a
  detailed, rigorous inspection of the code.

There are four essential elements to a formal review

1. Identify Problems:
2. Follow Rules:
3. Prepare: -
4. Write a Report: -

ii. Peer Reviews:

- The easiest way to get team members together and doing their first formal reviews
  of the software is through peer reviews, the least formal method.
- Sometimes called buddy reviews, this method is really more of a discussion.
- Peer reviews are often held with just the programmer who wrote the code and one or
  two other programmers or testers acting as reviewers.
- Small group simply reviews the code together and looks for problems and
  oversights.
- To assure that the review is highly effective all the participants need to make sure
  that the four key elements of a formal review are in place: Look for problems, follow
  rules, prepare for the review, and write a report.
- As peer reviews are informal, these elements are often scaled back. Still, just getting
  together to discuss the code can find bugs.

**ii. Structural Testing-Code Functional Testing, Code Coverage Testing, Code
Complexity Testing.**

1. Data Flow (Code Functional Testing)

i. Data flow coverage involves tracking a piece of data completely through the
software.

ii. At the unit test level this would just be through an individual module or function.

iii. The same tracking could be done through several integrated modules or even
through the entire software product—although it would be more time consuming to do

so.

iv. During data flow, the check is made for the proper declaration of variables declared and the loops used are declared and used properly.
For example
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("enter the number ");
6. scanf("%d",&n);
7. for(i =1 ;i <=n;i++)
8. fact = fact * i;
9. printf ("the factorial of a number is "%d", fact);
10. }

2. Data Coverage (Code Coverage Testing)
i. The logical approach is to divide the code just as you did in black-box testing into its data and its states (or program flow).
ii. By looking at the software from the same perspective, you can more easily map the white-box information you gain to the black-box cases you've already written.
iii. Consider the data first. Data includes all the variables, constants, arrays, data structures, keyboard and mouse input, files and screen input and output, and
I/O to other devices such as modems, networks, and so on.

For example
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("enter the number ");
6. scanf("%d",&n);
7. for(i =1 ;i <=n;i++)
8. fact = fact * i;
9. printf ("the factorial of a number is %d", fact);
10. }
The declaration of data is complete with the assignment statement and the variable declaration statements. All the variable declared are properly utilized.

3. Program Statements and Line Coverage (Code Complexity Testing)
i. The most straightforward form of code coverage is called statement coverage or line coverage.
ii. If you're monitoring statement coverage while you test your software, your goal is to make sure that you execute every statement in the program at least once.
iii. With line coverage the tester tests the code line by line giving the relevant output.
For example
1. #include<stdio.h>
2. void main()
3. {

		4. int i , fact= 1, n; 5. printf("enter the number "); 6. scanf("%d", &n); 7. for(i =1 ;i <=n; i++) 8. fact = fact * i; 9. printf ("the factorial of a number is %d", fact); 10. } **4. Branch Coverage (Code Complexity Testing)** i. Attempting to cover all the paths in the software is called path testing. ii. The simplest form of path testing is called branch coverage testing. iii. To check all the possibilities of the boundary and the sub boundary conditions and it's branching on those values. iv. Test coverage criteria requires enough test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry to a program or subroutine is invoked at least once. v. Every branch (decision) taken each way, true and false. vi. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application. **5. Condition Coverage (Code Complexity Testing)** i. Just when you thought you had it all figured out, there's yet another Complication to path testing. ii. Condition coverage testing takes the extra conditions on the branch statements into account.	
	(ii)	**How to select testing tool? Explain in detail.**	4M
	Ans:	Selecting a Testing Tool. i. While introducing the tool in the organization it must match a need within the organization, and solve that need in a way that is both effective and efficient. ii. The tool should help in building the strengths of the organization and should also address its weaknesses. iii. The organization needs to be ready for the changes that will come along with the new tool. iv. If the current testing practices are not good enough and the organization is not mature, then it is always recommended to improve testing practices first rather than to try to find tools to support poor practices. Certainly, we can sometimes improve our own processes in parallel with introducing a tool to support those practices and we can always pick up some good ideas for improvement from the ways that the tools work. v. Do not depend on the tool for everything, but it should provide support to your organization as expected. **The following factors are important during tool selection:** i. Assessment of the organization's maturity (e.g. readiness for change); ii. Identification of the areas within the organization where tool support will help to improve testing processes; iii. Evaluation of tools against clear requirements and objective criteria; iv. Proof-of-concept to see whether the product works as desired and meets the requirements and objectives defined for it; v. Evaluation of the vendor (training, support and other commercial aspects) or open-source network of support; vi. Identifying and planning internal implementation (including coaching and	**(Tool selection parameters: 2 marks, factors : 2 marks)**

		mentoring for those new to the use of the tool).	
	(iii)	**Enlist components of usability testing. Which features of Software are tested in usability testing?**	**4M**
	Ans:	**Usability Testing:** Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users. Usability Testing is a **black box testing** technique. Usability testing also reveals whether users feel comfortable with your application or Web site according to different parameters – the flow, navigation and layout, speed and content – especially in comparison to prior or similar applications. **Usability testing includes the following five components:** 1. **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design? 2. **Efficiency:** How fast can experienced users accomplish tasks? 3. **Memorability:** When users return to the design after a period of not using it, does the user remember enough to use it effectively the next time, or does the user have to start over again learning everything? 4. **Errors:** How many errors do users make, how severe are these errors and how easily can they recover from the errors? 5. **Satisfaction:** How much does the user like using the system? **Usability Testing tests the following features of the software:** How easy it is to use the software. How easy it is to learn the software. How convenient is the software to end user. **Usability Testing Process: (OPTIONAL) continued..** 	**(components: 2 marks, features: 2 marks)**
	(iv)	**Define Metrics and Measurements. Explain need of Software measurement.**	**4M**
	Ans.:	A **Metric** is a measurement of the degree that any attribute belongs to a system, product or process. For example the number of errors per person hours would be a metric. Thus, software measurement gives rise to software metrics. A **measurement** is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. For example the number of errors in a system is a measurement.	**(Definition: 1 mark and Needs: 3 marks)**

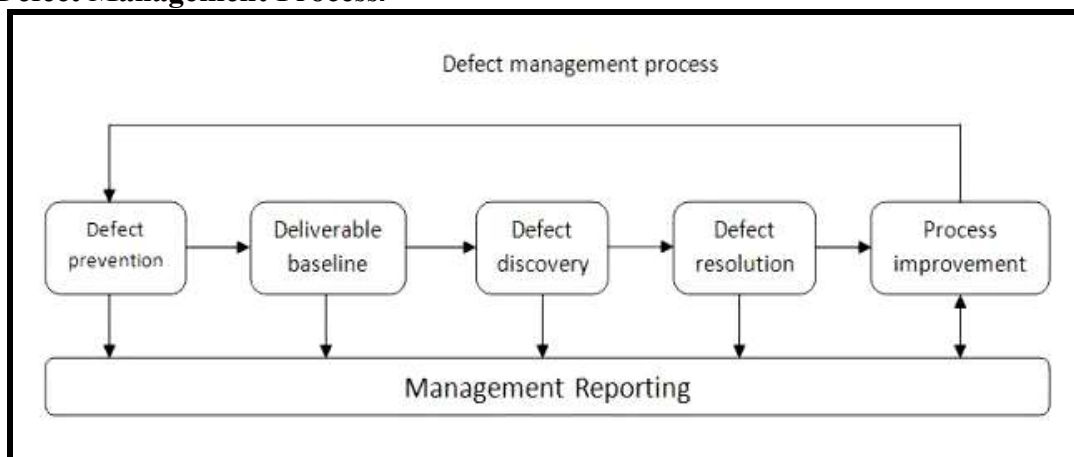**Software measurement is required to:**

- Establish the quality of the current product or process.
- To predict future qualities of the product or process.
- To improve the quality of a product or process.
- To determine the state of the project in relation to budget and schedule.
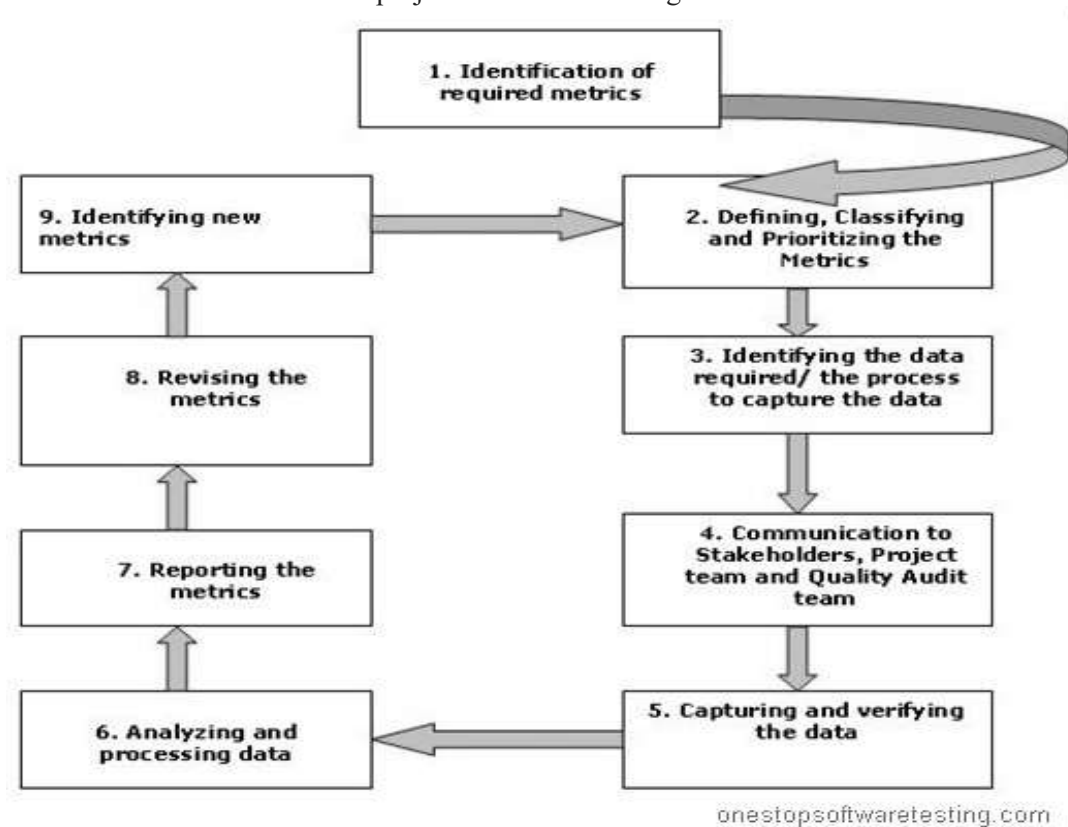
			4M
(v)		**Explain Defect Management Process.**	
Ans.:		**Defect Management Process:**	**(Explanation with diagram:4 marks)**

Figure 1: Defect management Process

		Defect Management process must include the appraisal of a defect finding process, software development process and the actions initiated to close the defects and strengthen the product/process associated with development, so that defects are not repeated again and again. It typically includes correction, corrective action and preventive action. It includes, Defect Naming Defect Resolution Once the developer have acknowledged a valid defect , the resolution process starts: • To report on the status of individual defects • To provide tactical information and metrics to help project management, redesign of error prone modules. • To provide strategic information and metrics to senior management, defect trends, problem systems. Process to be improved to either prevents defects or minimizing their impact. • To provide insight into the likelihood that target dates and cost estimates will be i. **Defect Prevention**-- Implementation of techniques, methodology and standard processes to reduce the risk of defects. ii. **Deliverable Baseline**-- Establishment of milestones where deliverables will be considered complete and ready for further development work. When a deliverable is base lined, any further changes are controlled. Errors in a deliverable are not considered defects until after the deliverable is base lined. iii. **Defect Discovery**-- Identification and reporting of defects for development team acknowledgment. A defect is only termed discovered when it has been documented and acknowledged as a valid defect by the development team Member responsible for the component(s) in error. iv. **Defect Resolution**-- Work by the development team to prioritize, schedule and fix a defect, and document the resolution. This also includes notification back to the tester to ensure that the resolution is verified.	
(vi)		**State any four objectives of user documentation testing. How these are useful in planning user documentation test?**	**4M**
Ans.:		User Documentation covers all the manuals, user guides, installation guides, setup guides, read me files, software release notes, and online help that are provided along with the software to help the end user to understand the software system. User Documentation Testing should have two objectives:- • To check if what is stated in the document is available in the software • To check if what is there in the product is explained correctly in the document • This testing is plays a vital role as the users will refer this document when they start using the software at their location. a badly written document can put off a user and bias them against the product even the product offers rich functionality. Defects found in the user documentation need to be tracked to closure like any regular software defect. Because these documents are the first interactions the users have with the product. A good User Documentation aids in reducing customer	**(Objectives:2 marks, usefulness: 2 marks)**

support calls. The effort and money spend on this effort would form a valuable investment in the long run for the organization. A good observation is that projects that have all the documents in place have a high level of maturity as compared to the un-documented project. This is a trend in today's testing process. Yet there are a few companies that pay little or no attention to documentation and are only attentive towards the software development process. Documentation for an organization saves time, cost and makes testing easy and systematic. It is equally important for the client's acceptance because documentation defines a software product's effectiveness. If the documentation is poor, deficient, or defective, it may affect the quality of software or application. QA practices should be documented such that they are repeatable, and are not dependent on any individuals. During manual software testing, documentation will include specifications, test designs, test plan, prevalent business rules, reports, configurations details, changes in code, test cases, bug reports, user manuals, etc. As a part of documentation, there needs to be a system for easily finding and obtaining documents and determining what documentation will have a particular piece of information. Once the details are documented, they should be placed at a common databank where easy search and timely availability of the records is feasible. These documents come handy in times of any dispute or comparing the requirement specification with the delivered product.

Few essential software testing documents that need to be used and maintained on a day to day basis:

- Test design document
- Test case specification
- Test Strategy
- Test summary reports
- Document of Weekly Status Report
- User Documents
- Document of User Acceptance Report
- Report of Risk Assessment
- Test Log document
- Test plan document
- Bug reports document
- Test data document
- Test analysis

There should be standard templates for all the kinds of documentation starting from Test strategy, test Plan, Test cases, and Test data to Bug report. It is imperative for the testers to synchronize the quality process with documentation standards and other process in an organization. Documentation is also very effective when automated testing or software performance testing is planned to be executed.

3.		Attempt any FOUR of the following:	16 Marks
	(i)	With the help of suitable example explain decision table. Why decision table is important?	4M
	Ans:	**Decision Tables.**	(Explanation with Diagram: 3 marks , Importance: 1 mark)

Decision Table

Conditions	TC1	TC2	TC3	TC4
Request login	0	1	1	1
Valid user name entered	X	0	1	1
Valid password entered	X	X	0	1
Actions				
Offer recovery credentials	0	1	1	0
Activate entrybox user name	0	1	1	0
Activate entrybox password	0	0	1	0
Enter priviliged area	0	0	0	1

i. Decision table testing is black box test design technique to determine the test scenarios for complex business logic.

ii. Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.

iii. Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.

iv. It helps the developers to do a better job can also lead to better relationships with them.

v. Testing combinations can be a challenge, as the number of combinations can often be huge.

vi. Testing all combinations may be impractical if not impossible.

vii. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important.

viii. If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

Importance of Decision Table:
Essentially it is a structured exercise to formulate requirements when dealing with complex business rules. **Decision tables** are used to model complicated logic. They can make it easy to see that all possible combinations of conditions have been considered and when conditions are missed, it is easy to see.

(ii)	**Explain GUI testing with suitable example.**	**4M**
Ans:	{**Note: Another tools may be considered**} **GUI Testing:** i. GUI testing is a testing technique in which the application's user interface is tested whether the application performs as expected with respect to user interface behavior. ii. GUI Testing includes the application behavior towards keyboard and mouse movements and how different GUI objects such as toolbars, buttons, menu bars, dialog boxes, edit fields, lists, behaviour to the user input. GUI Testing Guidelines i. Check Screen Validations ii. Verify All Navigations iii. Check usability Conditions iv. Verify Data Integrity v. Verify the object states vi. Verify the date Field and Numeric Field Formats **GUI Automation Tools** Following are some of the open source GUI automation tools in the market:	(Explanation of GUI Testing: 4 marks)

Product	Licensed Under	URL
AutoHotkey	GPL	http://www.autohotkey.com/
Selenium	Apache	http://docs.seleniumhq.org/
Sikuli	MIT	http://sikuli.org
Robot Framework	Apache	www.robotframework.org
Water	BSD	http://www.watir.com/
Dojo Toolkit	BSD	http://dojotoolkit.org/

Following are some of the Commercial GUI automation tools in the market.

Product	Vendor	URL
AutoIT	AutoIT	http://www.autoitscript.com/site/autoit/
EggPlant	TestPlant	www.testplant.com
QTP	Hp	http://www8.hp.com/us/en/software-solutions/
Rational Functional Tester	IBM	http://www-03.ibm.com/software/products/us/en/functional
Infragistics	Infragistics	www.infragistics.com
iMacros	iOpus	http://www.iopus.com/iMacros/
CodedUI	Microsoft	http://www.microsoft.com/visualstudio/

	(iii)	Explain the 'Test infrastructure' components with diagram.	4M
	Ans:	Testing requires a robust infrastructure to be planned upfront. This infrastructure is made up of three essential elements.	**(Listing of component : 2 marks, Explanation: 2 marks)**

1. A test case database (TCDB) (additional): A test case database captures all the relevant information about the test cases in an organization. Some of the entities and the attributes are given in following table

Sr. No.	Test Case	Purpose	Attributes
1	Test case	Records all static information about tests.	• Test case Id • Test case name (File name) • Test case owner • Associated files for test case.
2	Test case product cross reference	Provide mapping between the tests and the corresponding product features, enables identification of test cases for given feature.	• Test case Id • Module Id

3	Test case run history	Gives the history of when the test case was run and what was result , provided inputs on selection of test for regression runs	• Test case Id • Run date • Time taken • Run status(Success/ Failure)	
4	Test case-defect cross-reference	Gives details of test cases introduced to test certain specific defects detected in the product, provides inputs on the selection of test for regression runs.	• Test case Id • Defect reference	

Test Infrastructure

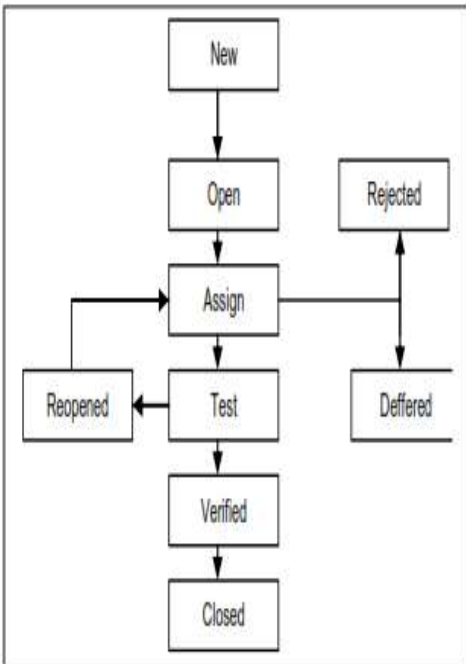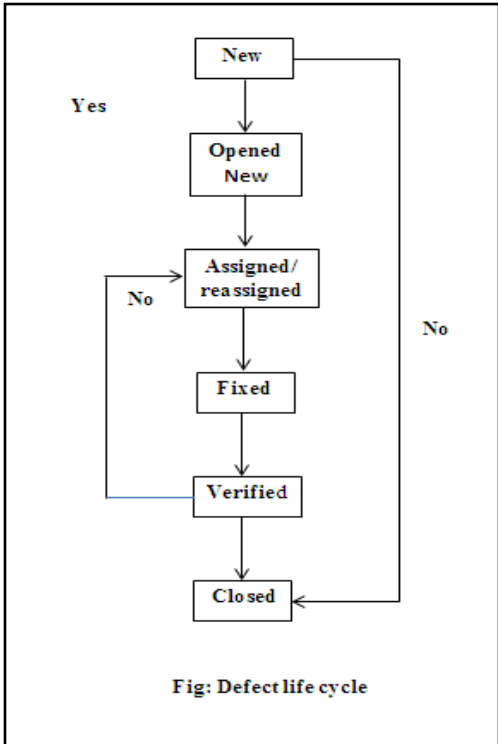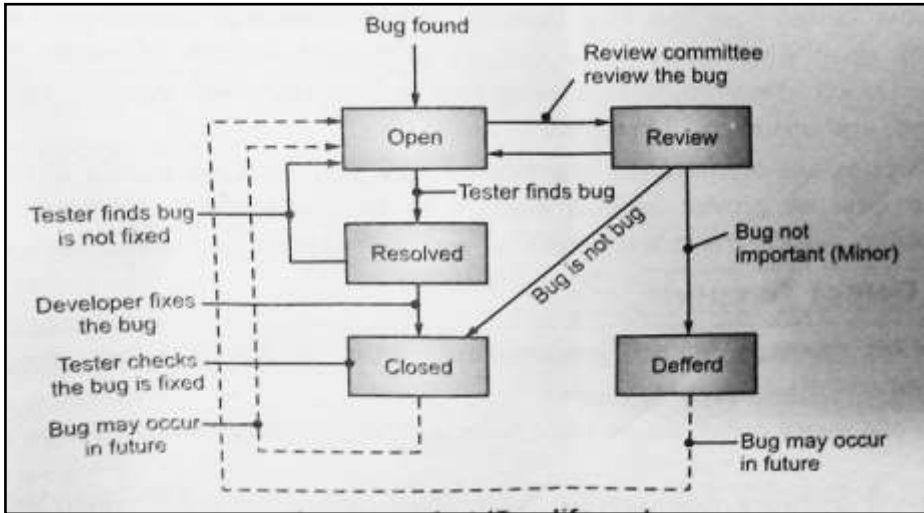| (iv)	Give comparison between Alpha testing and Beta Testing (any four points)	**4M**
Ans:	1. Alpha Testing is conducted by a team of highly skilled testers at development site whereas Beta Testing is always conducted in Real Time environment by customers or end users at their own site. 2. Alpha testing requires lab environment or testing environment, whereas Beta testing doesn't require any lab environment or testing environment. 3. Since Alpha Testing is done onsite therefore developers as well as business analyst are involved with the testing team whereas in Beta Testing developers and business analysts are not at all involved. 4. Beta testers can be naive or proficient end users of software product but alpha testers are always high skilled professional testers. 5. Alpha Testing involves both black box testing as well as white box testing. Beta Testing is always a black box testing or functional testing. 6. Alpha Testing is done before the launch of software product into the market whereas Beta Testing is done at the time of software product marketing. 7. Alpha Testing is conducted in the presence of developers and in the absence of end users whereas for Beta Testing this is exactly reversed. 8. Alpha testing is to ensure the quality of the product before moving to Beta testing Beta testing also concentrates on quality of the product, but gathers users input on the product and ensures that the product is ready for real time users. 9. Reliability and security testing are not performed in-depth Alpha Testing Reliability, Security, Robustness are checked during Beta Testing	**(Comparison of Alpha testing and Beta testing: 1 mark for each point minimum 4 points)**
(v)	Write down any four limitations of manual testing.	**4M**
Ans:	**Manual Testing:** Testing Computer Software manually without using any Automation Tools **Limitations of Manual Testing**: • Manual Testing requires more time or more resources, sometimes both • Performance testing is impractical in manual testing. • Less Accuracy • Executing same tests again and again time taking process as well as Tedious. • GUI Objects Size difference and Color combinations etc.. Are not easy to find in Manual Testing. • Not Suitable for Large scale projects and time bounded projects. • Batch Testing is not possible, for each and every test execution Human user interaction is mandatory. • Manual Test Case scope is very limited, if it is Automated test, scope is unlimited. • Comparing large amount of data is impractical • Checking relevance of search of operation is difficult • Processing change requests during software maintenance takes more time.	**(Any 4 limitations :1 mark each)**

4.	(a)	Attempt any THREE of the following:	12 Marks
	(i)	What is test case? Which parameters are to be considered while documenting test cases?	4M
	Ans:	**Test Case Specification:** Test case is a well-documented procedure designed to test the functionality of the feature in the system. For designing the test case, it needs to provide set of inputs and its corresponding expected outputs. **Parameters:** 1. Test case ID: is the identification number given to each test case. 2. Purpose: defines why the case is being designed. 3. Precondition: for running in the system can be defined, if required, in the test case. 4. Input: should not hypothetical. Actual inputs must be provided, instead of general inputs. Using the test plan as the basis, the testing team designs *test case specification*, which then becomes the basis for preparing individual *test cases*. Hence, a test case specification should clearly identify, 1. **The purpose of the test**: This lists what features or part the test is intended for. 2. **Items being tested**, along with their version/release numbers as appropriate. 3. **Environment that needs to be set up for running the test cases:** This includes the hardware environment setup, supporting software environment setup, setup of the product under test. 4. **Input data to be used for the test case:** The choice of input data will be dependent on the test case itself and the technique followed in the test case. 5. **Steps to be followed to execute the test:** If automated testing is used, then, these steps ate translated to the scripting language of the tool. 6. **The expected results that are considered to be "correct result".** 7. **A step to compare the actual result produced with the expected result:** This step should do an "intelligent" comparison of the expected and actual results to highlight any discrepancies. 8. **Any relationship between this test and other test:** These can be in the form of dependencies among the tests or the possibilities of reuse across the tests.	**(Explain test case: 2 marks, Parameters: 2 marks)**

(ii)	**What is automated testing? Write down advantages of using automated testing tools in software testing.**	**4M**
Ans:	i. An automated testing tool is able to playback pre-recorded and predefined actions, Compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. ii. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. iii. Because of this, savvy managers have found that automated software testing is an essential component of successful development projects. **Benefits of automation testing:** **1. Speed:** Think about how long it would take you to manually try a few thousand test cases for the windows Calculator. You might average a test case every five seconds or so. Automation might be able to run 10, 100 even 1000 times that fast. **2. Efficiency**: While you are busy running test cases, you can't be doing anything else. If you have a test tool that reduces the time it takes for you to run your tests, you have more time for test planning and thinking up new tests. **3. Accuracy and Precision**: After trying a few hundred cases, your attention may reduce and you will start to make mistakes .A test tool will perform the same test and check the result perfectly, each and every time. **4. Resource Reduction**: Sometimes it can be physically impossible to perform a certain test case. The number of people or the amount of equipment required to create the test condition could be prohibitive. A test tool can used to simulate the real world and greatly reduce the physical resources necessary to perform the testing. **5. Simulation and Emulation**: Test tools are used to replace hardware or software that would normally interface to your product. This "face" device or application can then be used to drive or respond to your software in ways that you choose-and ways that might otherwise be difficult to achieve. **6. Relentlessness**: Test tool and automation never tire or give up. It will continuously test the software. **OR** **Benefits of Automation Testing are:** **1**. **Save Time /Speed**: Due to advanced computing facilities, automation test tools prevail in speed of processing the tests. Automation saves time as software can execute test cases faster than human. **2**. **Reduces the tester's involvement in executing tests**: It relieves the testers to do some other work. **3. Repeatability/Consistency**: The same tests can be re-run in exactly the same manner eliminating the risk of human errors such as testers forgetting their exact actions, intentionally omitting steps from the test scripts, missing out steps from the test script, all of which can result in either defects not being identified or the reporting of invalid bugs (which can again, be time consuming for both developers and testers to reproduce) **4. Simulated Testing:** Automated tools can create many concurrent virtual users/data and effectively test the project in the test environment before releasing the product. **5. Test case design:** Automated tools can be used to design test cases also. Through automation, better coverage can be guaranteed than if done manually.	(Automated testing description: 1 mark, Any relevant three automated test tools benefits : 3 marks)

		6. Reusable: The automated tests can be reused on different versions of the software, even if the interface changes. **7. Avoids human mistakes:** Manually executing the test cases may incorporate errors. But this can be avoided in automation testing. **8. Internal Testing:** Testing may require testing for memory leakage or checking the coverage of testing. Automation can done this easily. **9. Cost Reduction:** If testing time increases cost of the software also increases. Due to testing tools time and therefore cost is reduced.	
(iii)		**Explain defect life cycle to identify status of defect with proper labeled diagram.**	**4M**
Ans:		**OR** Fig: Defect life cycle **OR** 	**(Diagram: 2 marks , Explanation: 2 marks minimum 6 states)**

The different states of bug life cycle are as shown in the above diagram:

- **New:** When the bug is posted for the first time, its state will be "NEW". This means that the bug is not yet approved.
- **Open:** After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as "OPEN".
- **Assign:** Once the lead changes the state as "OPEN", he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to "ASSIGN".
- **Test/Retest:** Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to "TEST". It specifies that the bug has been fixed and is released to testing team.// At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.
- **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.
- **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "REJECTED".
- **Verified:** Once the bug is fixed and the status is changed to "TEST", the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "VERIFIED".
- **Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "REOPENED". The bug traverses the life cycle once again.
- **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "CLOSED". This state means that the bug is fixed, tested and approved.
- **Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as „Fixed" and the bug is passed to testing team.
- **Pending retest:** After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.

Optional

- **Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to "duplicate".
- **Not a bug:** The state given as "Not a bug" if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the application.

| (iv)	Explain test deliverables in detail.		4M
Ans:	The deliverables include the following,		(any eight points :1/2 mark each)
	The test plan	Helpful for tester	
	Test case Specification	Details needed for testing	
	Test design specification documents	Helpful in designing test	
	Testing Strategy	Approach to follow testing	
	Testing Scripts/ procedures	Need to be followed	
	Test data	Data useful during testing	
	Test Incident report	Details of situation where testing performed	
	Test Traceability matrix	Metrix to follow testing	
	Test results /Reports	Entire report of testing	
	Install/Configuration guides	Provides guidelines before testing	
	Test logs produced	Useful for future testing	
	Defect Report/ Release report	After completion of test this report is generated/prepared	
(b)	**Attempt any ONE of the following::**		**6 Marks**
(i)	Describe V-model with labelled diagram. State its any two advantages and disadvantages. Also write where it is applicable.		6M
Ans:	V-model means verification and validation model. It is sequential path of execution of processes. Each phase must be completed before the next phase begins. Under V-model, the corresponding testing phase of the development phase is planned in parallel. So there is verification on one side of V & validation phase on the other side of V. **Verification Phase:** 1. **Overall Business Requirement:** In this first phase of the development cycle, the product requirements are understood from customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirements. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.		(Explanation: 1 mark and diagram: 1 mark , Advantages:2 marks , Disadvantages: 2 marks (any two)

2. **Software Requirement**: Once the product requirements are clearly known, the system can be designed. The system design comprises of understanding & detailing the complete hardware, software & communication set up for the product under development. System test plan is designed based on system design. Doing this at earlier stage leaves more time for actual test execution later.

3. **High level design:** High level specification are understood & designed in this phase.

Usually more than one technical approach is proposed & based on the technical & financial feasibility, the final decision is taken. System design is broken down further into modules taking up different functionality.

4. **Low level design:** In this phase the detailed integral design for all the system modules is specified. It is important that the design is compatible with the other modules in the system & other external system. Components tests can be designed at this stage based on the internal module design,

5. **Coding:** The actual coding of the system modules designed in the design phase is taken up in the coding phase. The base suitable programming language is decided base on requirements. Coding is done based on the coding guidelines & standards.

<u>Validation:</u>

▪ **Unit Testing:** Unit testing designed in coding are executed on the code during this validation phase. This helps to eliminate bugs at an early stage.

▪ **Components testing:** This is associated with module design helps to eliminate defects in individual modules.

▪ **Integration Testing:** It is associated with high level design phase & it is performed to test the coexistence & communication of the internal modules within the system

▪ **System Testing:** It is associated with system design phase. It checks the entire system functionality & the communication of the system under development with external systems. Most of the software & hardware compatibility issues can be uncovered using system test execution.

▪ **Acceptance Testing:** It is associated with overall & involves testing the product in user environment. These tests uncover the compatibility issues with the other systems available in the user environment. It also uncovers the non-functional issues such as load & performance defects in the actual user environment.
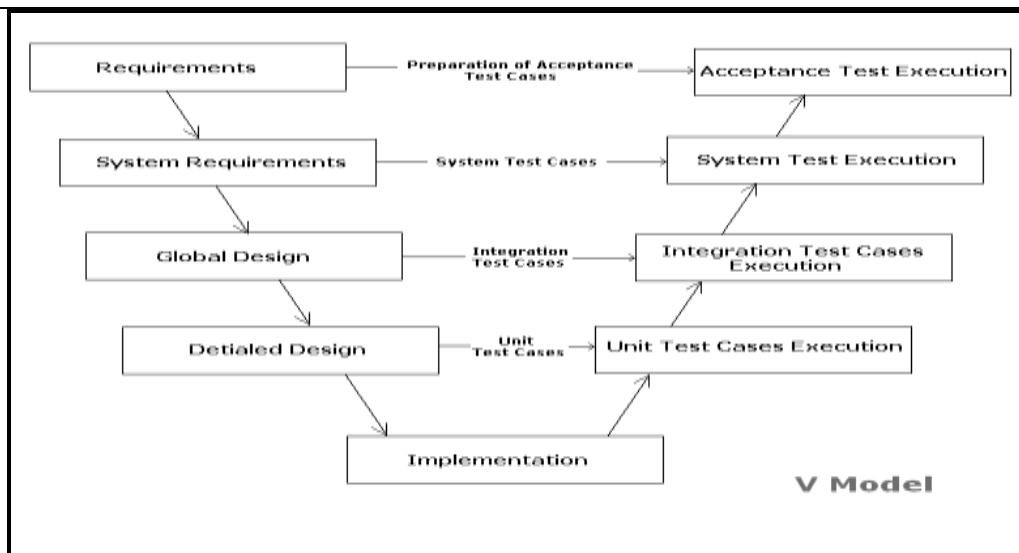
V Model

OR
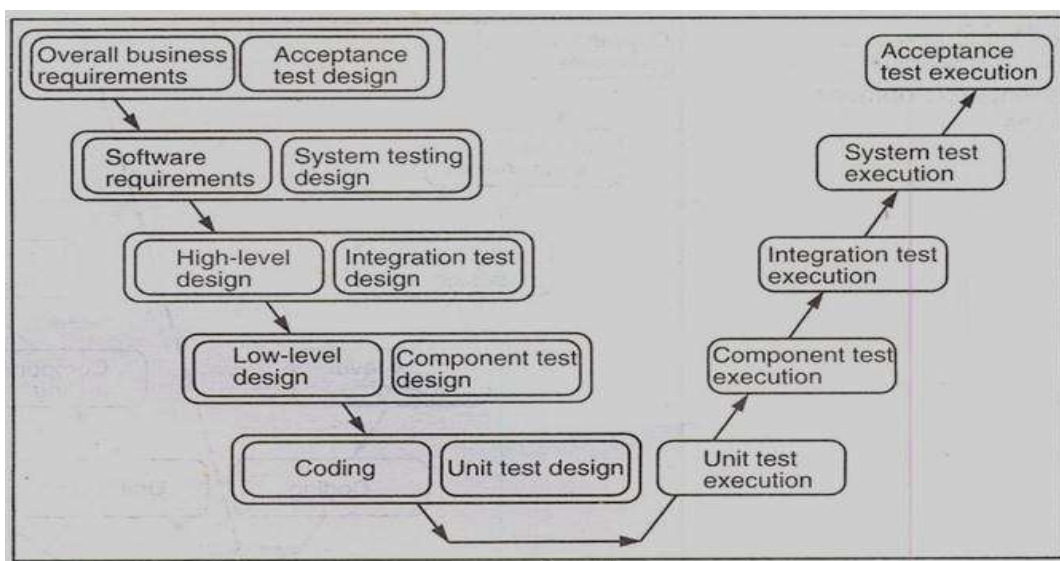
Advantages of V-model:	Disadvantages of V-model:
• Simple and easy to use. • Testing activities like planning, <u>test designing</u> happens well before coding. • Saves a lot of time. • Higher chance of success over the waterfall model. • Proactive defect tracking – that is defects are found at early stage. • Avoids the downward flow of the defects. • Works well for small projects where requirements are easily understood.	• Very rigid and least flexible. • Software is developed during the implementation phase, so no early prototypes of the software are produced. • If any changes happen in midway, then the test documents along with requirement documents has to be updated.

		When to use the V-model: • The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed. • The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.	
(ii)		**With the help of example explain Boundary Value Analysis.**	**6M**
Ans:		Most of the defects in software products hover around conditions and boundaries. By conditions, we mean situations wherein, based on the values of various variables, certain actions would have to be taken. By boundaries, we mean "limits" of values of the various variables. • This is one of the software testing technique in which the test cases are designed to include values at the boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing. • Boundary value analysis is another black box test design technique and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input. • Each boundary has a valid boundary value and an invalid boundary value. Test cases are designed based on the both valid and invalid boundary values. Typically, we choose one test case from each boundary. • Same examples of Boundary value analysis concept are: One test case for exact boundary values of input domains each means 1 and 100. One test case for just below boundary value of input domains each means 0 and 99. One test case for just above boundary values of input domains each means 2 and 101. **For Example:** A system can accept the numbers from 1 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values 0, 1,2, 9,10,11 can be tested. Another Example is in exam has a pass boundary at 40 percent, merit at 75 percent and Distinction at 85 percent. The Valid Boundary values for this scenario will be as follows: 49, 50 - for pass 74, 75 - for merit 84, 85 - for distinction Boundary values are validated against both the valid boundaries and invalid boundaries. The Invalid Boundary Cases for the above example can be given as follows 0 - for lower limit boundary value 101 - for upper limit boundary value • Boundary value analysis is a black box testing and is also applies to white box testing. Internal data structures like arrays, stacks and queues need to be checked for boundary or limit conditions; when there are linked lists used as internal structures, the behavior of the list at the beginning and end have to be tested thoroughly.	**(Explanation: 4 marks and 2 mark for Example)**

• Boundary value analysis help identify the test cases that are most likely to uncover defects.				

5.		Attempt any TWO of the following :	**16 Marks**
	(i)	**Prepare any eight test cases for admission form of college admission.**	**8M**
	Ans:	{** Note: Any other relevant test cases also can be considered**} Consider the college admission form having different fields such as Student's Name, Father's Name, Zip code ,Address, Phone, Caste, admission type ,S.S.C percentage, SC Board, Submit button, Reset button.	**(8 test cases:1 mark each)**

Test Case Id	Test case Objectives	Input Data	Expected Result	Actual Result	Status
TC1	Name field	Any name (abcd xyz)	It should accept the name	The name is accepted	Pass
TC2	Phone Field	Any number Having less than 10 digits(1234)	It should not Accept. Should give error message "Please enter valid phone number"	Error message "Please enter valid phone number"	Pass
TC3	Phone Field	Any Alphabets (abcde)	It should give error message as "Only Numbers"	Error message as "Only Numbers"	Pass
TC4	SSC Percentage Field	65	It should accept	It accepted	Pass
TC5	SSC Percentage Field	30	It should not accepted should give error message	Gives error message	Pass

	TC6	Address field	Any characters (A-51, Market road ,Mumbai)	It should accept	It accepted	Pass	
	TC7	Zip code	Any six digit number(4314 01)	It should accept the number	It accepted	Pass	
	TC8	Required fields should be filled in the form	Data in the mandatory fields.	If it is filled.	Do not display Error Message	Pass	

(ii)		**With the suitable example, explain how 'Basis path Testing' is used to derive the code complexity for the testing.**	**8M**
Ans:		**{** Note: any other relevant answer content shall also be considered**}** **{** Note: Basis/Basic path Testing shall be considered**}** **Basis path testing is the structural testing technique:** ▪ Path testing is based on control structure of the program for which flow graph is prepared ▪ Path testing requires complete knowledge of the programs structure. ▪ Path testing is closer to the developer and used by him to test his module. ▪ The effectiveness of path testing gets reduced with te increase in size of software under test. • Choose enough paths in a program such that maximum logic coverage is achieved. **Branch Coverage, code coverage, line coverage is also called as basis path testing.** Attempting to cover all the paths in the software is called basis path testing. It's the actual structural testing which is the part of static white box testing. So many times static white box testing is called basis path testing. The simplest form of path testing is called branch coverage testing. To check all the possibilities of the boundary and the sub boundary conditions and it's branching on those values. Test coverage criteria requires enough test cases such that each condition in a decision takes on all possible	**(Basis path testing:2 marks, description of how it is used to derive the code complexity for the testing: 6 marks)**

outcomes at least once, and each point of entry to a program or subroutine is invoked at least once.

Every branch (decision) taken each way, true and false. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

For example in the following code all the branches in the program are checked thoroughly. The decisions are evaluated and are tested whether are not these branches are taken.

1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("enter the number ");
6. scanf("%d", &n);
7. for(i =1 ;i <=n; i++)
8. fact = fact * i;
9. printf ("the factorial of a number is %d", fact);
10. }

Data Flow (Code Functional Testing): Data flow coverage involves tracking a piece of data completely through the software. At the unit test level this would just be through an individual module or function. The same tracking could be done through several integrated modules or even through the entire software product— although it would be more time consuming. During data flow, the check is made for the proper declaration of variables declared and the loops used are declared and used properly.

Line coverage or code coverage testing: The most straightforward form of code coverage is called statement coverage or line coverage. If you're monitoring statement coverage while you test your software, your goal is to make sure that you execute every statement in the program at least once. With line coverage the tester tests the code line by line giving the relevant output.

For example :

1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("enter the number ");
6. scanf("%d", &n);
7. for(i =1 ;i <=n; i++)
8. fact = fact * i;
9. printf ("the factorial of a number is %d", fact);
10.   }

	(iii)	What is integration testing? List types of integration testing and explain any four types in brief.	8M
	Ans:	Testing that occurs at the lowest level is called unit testing or module testing. As the units are tested and the low-level bugs are found and fixed, they are integrated and integration testing is performed against groups of modules. This process of incremental testing continues, putting together more and more pieces of the software until the entire product or at least a major portion of it is tested at once in a process called system testing. With this testing strategy, it's much easier to isolate bugs. When a problem is found at the unit level, the problem must be in that unit. If a bug is found when multiple units are integrated, it must be related to how the modules interact. Of course, there are exceptions to this, but by and large, testing and debugging is much more efficient than testing everything at once. **Types of Integration testing:** **1). Top down Testing:** In this approach testing is conducted from main module to sub module. If the sub module is not developed a temporary program called STUB is used for simulate the sub module. **Advantages:** • Advantageous if major flaws occur toward the top of the program. • Once the I/O functions are added, representation of test cases is easier. • Early skeletal Program allows demonstrations and boosts morale. **Disadvantages:** • Stub modules must be produced • Stub Modules are often more complicated than they first appear to be. • Before the I/O functions are added, representation of test cases in stubs can be difficult. • Test conditions may be impossible, or very difficult, to create. • Observation of test output is more difficult. • Allows one to think that design and testing can be overlapped. • Induces one to defer completion of the testing of certain modules. **2). Bottom up testing:** In this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulate the main module. **Advantages:** • Advantageous if major flaws occur toward the bottom of the program. • Test conditions are easier to create. • Observation of test results is easier. • Driver Modules must be produced. • The program as an entity does not exist until the last module is added. **3). Bi-Directional Integration.**	(Integration testing: 2 marks; List any four types: 2 marks, explanation with example of the steps of integration :4 marks)

- Bi-directional Integration is a kind of integration testing process that combines top-down and bottom-up testing.
- With an experience in delivering Bi-directional testing projects custom software development services provide the best quality of the deliverables right from the development of software process.
- Bi-directional Integration testing is a vertical incremental testing strategy that tests the bottom layers and top layers and tests the integrated system in the computer software development process.
- Using stubs, it tests the user interface in isolation as well as tests the very lowest level functions using drivers. Bi-directional Integration testing combines bottom-up and top-down testing
- Bottom-up testing is a process where lower level modules are integrated and then tested.
- This process is repeated until the component of the top of the hierarchy is analysed. It helps custom software development services find bugs easily without any problems.
- Top down testing is a process where the top integrated modules are tested and the procedure is continued till the end of the related module.
- Top down testing helps developers find the missing branch link easily.

4). Incremental Integration.
- After unit testing is completed, developer performs integration testing.
- It is the process of verifying the interfaces and interaction between modules.
- While integrating, there are lots of techniques used by developers and one of them is the incremental approach.
- In Incremental integration testing, the developers integrate the modules one by one using stubs or drivers to uncover the defects.
- This approach is known as incremental integration testing.
- To the contrary, big bang is one other integration testing technique, where all the modules are integrated in one shot.

Features:
- Each Module provides a definitive role to play in the project/product structure
- Each Module has clearly defined dependencies some of which can be known only at the runtime.
- The incremental integration testing's greater advantage is that the defects are found early in a smaller assembly when it is relatively easy to detect the root cause of the same.
- A disadvantage is that it can be time-consuming since stubs and drivers have to be developed for performing these tests

5). Non- Incremental Integration.
- The non-incremental approach is also known as ―Big-Bang‖ Testing.
- Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system.
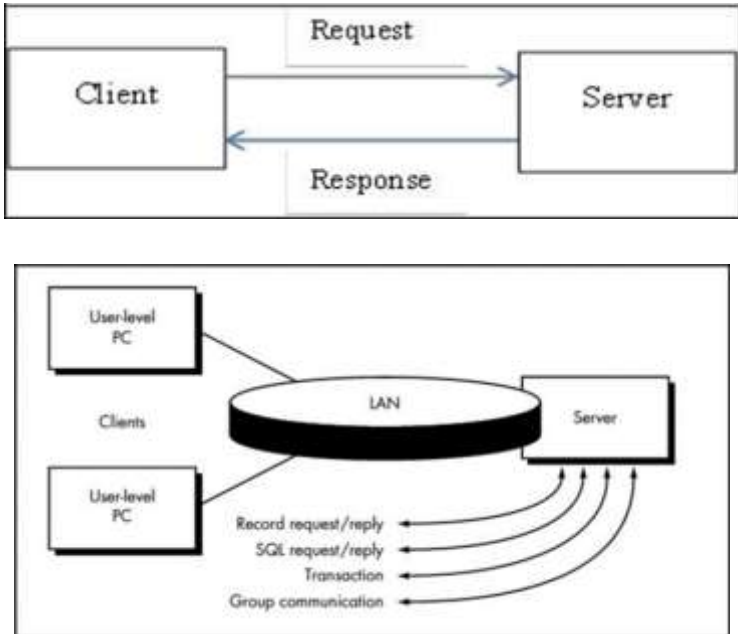- When this type of testing strategy is adopted, it is difficult to isolate any

		errors found, because attention is not paid to verifying the interfaces across individual units.	
6.		**Attempt any FOUR of the following:**	**16 Marks**
	(i)	**Which parameters are considered while writing good defect report? Also write contents of defect template.**	**4M**
	Ans:	A defect report documents an anomaly discovered during testing. It includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc. After uncovering a defect (bug), testers generate a formal defect report. The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it. **DEFECT REPORT TEMPLATE:** In most companies, a defect reporting tool is used and the elements of a report can vary. However, in general, a defect report can consist of the following elements.	**(Parameters: 2 marks, contents of defect template:2 marks)**

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name.
Product	Product name.
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.
Description	Detailed description of the defect. Describe as much as possible but without Repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.
Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.
Defect Priority	Priority of the Defect.
Reported By	The name of the person who reported the defect.
Assigned To	The name of the person that is assigned to analyze/fix the defect.
Status	The status of the defect.
Fixed Build Version	Build version of the product where the defect was fixed (e.g. 1.2.3.9)

(ii)	Define the term Error: Defect, Fault and Bug in relation with Software testing.	4M
Ans:	The various terms related to software failure with respect to the area of application are listed as Defect, Variance, Fault, Failure, Problem, Inconsistency, Error, Feature, Incident, Bug, and Anomaly. • **Fault:** An incorrect step, process, or data definition in a computer program. • **Error:** A human action that produces an incorrect result. • An **error** can be a **grammatical error** in one or more of the code lines, or a **logical error** in carrying out one or more of the client's requirements. • Not all software errors become **software faults.** in some cases, the software error can cause improper functioning of the software. In many other cases, erroneous code lines will not affect the functionality of the software as a whole. • **A failure** is said to occur whenever the external behavior of a system does not conform to that prescribed in the system specification. A software fault becomes a software failure only when it is ―activated‖ • The various terms related to software failure with respect to the area of application are listed as **Defect, Variance, Fault, Failure, Problem, Inconsistency, Error, Feature, Incident, Bug, and Anomaly** Problem, error, and bug are probably the most generic terms used. ▪ Anomaly, incident, and variance don't sound quite so negative and infer more unintended operation than an all-out failure. ▪ Fault, failure, and defect tend to imply a condition that's really severe, maybe even dangerous. It doesn't sound right to call an incorrectly colored icon a fault. These words also tend to imply blame: ―It's his fault that the software failed. ▪ As all the words sound the same they are distinguished based on the severity and the area in which the software failure has occurred. ▪ When we run a program the error that we get during execution is termed on the basis of runtime error, compile time error, computational error, and assignment error. ▪ The error can be removed by debugging, if not resolved leads to a problem and if the problem becomes large leads to software failure. ▪ A bug can be defined as the initiation of error or a problem due to which fault, failure, incident or an anomaly occurs. • The program to find the factorial of a number given below lists few errors problem and failure in a program. **For example** ▪ #include<stdio.h> ▪ void main() ▪ { ▪ int i , fact, n; ▪ printf(―enter the number ―); ▪ scanf(―%d‖,&n); ▪ for(i =1 ;i <=n;i++) ▪ fact = fact * i; ▪ printf (―the factorial of a number is ‖%d‖, fact); ▪ }	(Description of error, defect, fault & bug: 4 marks)

		As in line number 4 the fact is not initialized to 1, so it takes garbage value and gives a wrong output, this is an example of a bug. If fact is initialized to zero (fact = 0) than the output will be zero as anything multiplied by zero will give the output as zero. This is a bug which can be removed by initializing fact = 1 during initializing. As the fact is declared as integer, for the number till 7! will work perfectly. When the number entered is 8, the output is garbage value as the integer limit is from – 32767 to +32768, so in declaration change the initialization to long int fact.	
	(iii)	**Give the defect classification and its meaning.**	**4M**
	Ans:	**Defect Classification:**	*(Any 4 classification: 1 mark for each)*

Defect Classification:

Requirements and specification defect: Requirement related defects arise in a product when one fails to understand what is required by the customer. These defects may be due to customer gap, where the customer is unable to define his requirements, or producer gap, where developing team is not able to make a product as per requirements. Defects injected in early phases can persist and be very difficult to remove in later phases. Since any requirements documents are written using natural language representation, there are very often occurrences of ambiguous, contradictory, unclear, redundant and imprecise requirements. Specifications are also developed using natural language representations.

Design Defects: Design defects occur when system components, interactions between system components, interactions between the outside software/hardware, or users are incorrectly designed. This covers in the design of algorithms, control, logic/ data elements, module interface descriptions and external software/hardware/user interface descriptions. Design defects generally refer to the way of design creation or its usage while creating a product. The customer may or may not be in a position to understand these defects, if structures are not correct. They may be due to problems with design creation and implementation during software development life cycle.

Coding Defects: Coding defects may arise when designs are implemented wrongly. If there is absence of development/coding standards or if they are wrong, it may lead to coding defects. Coding defects are derived from errors in implementing the code. Coding defect classes are closely related to design defect classes especially if pseudo code has been used for detailed design. Some coding defects come from a failure to understand programming language constructs, and miscommunication with the designers. Others may have transcription or omission origins. At times it may be difficult to classify a defect as a design or as a coding detect.

Testing Defect: Testing defect are defects introduced in an application due to wrong testing, or defects in the test artifact leading to wrong testing. Defects which cannot be reproduced, or are not supported by requirement or are duplicate may represent a false call .In this defects includes

1. **Test-design defect:** test-design defect refers to defects in test artifacts. there can be defects in test plans, test scenarios, test cases and test data definition which can lead to defect in software.

2. **Test-environment defect:** this defect may arise when test environment is not set properly. Test environment may be comprised of hardware, software, simulator and people doing testing.

3. **Test-tool defects:** any defects introduced by a test tool may be very difficult to find and resolve, as one may have to find the defect using manual test as against automated tools.

		OR **Software Defects/ Bugs are normally classified as per:** • Severity / Impact • Probability / Visibility • Priority / Urgency • Related Dimension of Quality • Related Module/Component • Phase Detected • Phase Injected	
	(iv)	**What is test planning and test management?**	**4M**
	Ans:	**Test Planning**: Like any project, the testing also should be driven by a plan. The test plan acts as the anchor for the execution, tracking and reporting of the entire testing project. Activities of test plan: **1.** Scope Management: Deciding what features to be tested and not to be tested. **2.** Deciding Test approach /strategy: Which type of testing shall be done like configuration, integration, localization etc. **3.** Setting up criteria for testing: There must be clear entry and exit criteria for different phases of testing. The test strategies for the various features and combinations determined how these features and combinations would be tested. **4.** Identifying responsibilities, staffing and training needs **Test Management**: It concerned with both test resource and test environment management. It is the role of test management to ensure that new or modified service products meet business requirements for which they have been developed or enhanced	**(Test Planning: 2 marks , Test Management: 2 marks)**
	(v)	**With the help of diagram describe client-server testing.**	**4M**
	Ans:	{**Note: Any other relevant diagram and tests also can be considered**} 	**(Diagram :1 mark, Explanation:3 marks)**

In Client-server testing there are several clients communicating with the server.

1. Multiple users can access the system at a time and they can communicate with the server.
2. Configuration of client is known to the server with certainty.
3. Client and server are connected by real connection.
4. Testing approaches of client server system:

1. **Component Testing:** One need to define the approach and test plan for testing client and server individually. When server is tested there is need of a client simulator, whereas testing client a server simulator, and to test network both simulators are used at a time.
2. **Integration testing:** After successful testing of server, client and network, they are brought together to form system testing.
3. **Performance testing:** System performance is tested when number of clients are communicating with server at a time. Volume testing and stress testing may be used for testing, to test under maximum load as well as normal load expected. Various interactions may be used for stress testing.
4. **Concurrency Testing:** It is very important testing for client-server architecture. It may be possible that multiple users may be accessing same record at a time, and concurrency testing is required to understand the behavior of a system in this situation.
5. **Disaster Recovery/ Business continuity testing:** When the client server are communicating with each other , there exit a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them. The requirement specifications must describe the possible expectations in case of any failure.
6. **Testing for extended periods:** In case of client server applications generally server is never shutdown unless there is some agreed Service Level Agreement (SLA) where server may be shut down for maintenance. It may be expected that server is running 24X7 for extended period. One needs to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons like memory leakage.
7. **Compatibility Testing:** Client server may be put in different environments when the users are using them in production. Servers may be in different hardware, software, or operating system environment than the recommended. Other testing such as security testing and compliance testing may be involved if needed, as per testing and type of system.