WINTER– 16 EXAMINATION

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
	(i)	**List and describe any four skills of software tester.**	4M
	Ans:	**Skills of software tester are as follows :** **1. Analytical and logical thinking** **i).** The major objective of testing is to identify the hidden errors, not simply prove that the software works. **ii).** For a tester to be effective in his role, he must be able to analyze the given business situation and judge all the possible scenarios. **iii).** He should have the capacity to identify and tackle unfamiliar problems and should develop a strategy to validate it. **iv).** Creating situations and validating the application under test, before presenting it to customers, can be done effectively only by a person who has strong analytical skills. **2. The ability to envision business situations** **i).** A tester should be able to envisage real-time business situations through mental mapping, abstracting the idea inferred from the specifications. **ii).** The real-time business scenarios should crystallize in a tester's mind, and he should think about what the case is rather than what ought to be the case or what he believes the case is. **iii).** A tester should be able to anticipate complex problems, in addition to visualizing and articulating them. **iv).** He should be able to do a complete system simulation rapidly and accurately. **v).** In the present software development environments, it is hard to believe that	**(Any 4 skills of software tester: list and describing: 4marks, 1mark each)**

teams/individuals will get enough time to do a series of conventional brainstorming sessions to finalize the concept mapping.

vi). Therefore, it is vital that a tester develop his conceptualization skills through mental mapping.

3. A sense of intellectual curiosity and creativity

i). A tester should understand that being an intellectual and being intellectually curious are not the same.

ii). A tester should arguably be the latter one -- intellectually curious -- which is all about asking questions and not about having answers.

iii). He should believe in the pursuit of knowledge as a value in and of itself.

iv). He should love asking questions and should not consider it a blow to his ego if he is wrong about something.

v). It is intellectual curiosity that motivates and prompts a tester to identify interesting questions about the software being tested.

vi). Thus, a tester should develop the skill to see what everyone else hasn't seen, to think what no one else has thought of and to do what no one else has dared.

4. A "global" approach

i). Software systems have become extremely complex.

ii). Most of the time, the system designed involves multiple stakeholders, and dealing with such systems is not always easy.

iii). A tester should be able to deal effectively with business situations marked by complexity and the number of interactions with third-party systems.

iv). He should be able to identify how the system under test interacts with other constituents of the system.

v). He should also be able to isolate the minutest units of the application under test and do the validation, keeping in mind the behavior of the system as a whole.

5. Critical thought and rational enquiry

i). The quality of life of an individual and the quality of what he produces/delivers depends largely on the quality of his thought process.

ii). The thought process of a tester should be undistorted, impartial and without any prejudices.

iii). A tester should be able to take charge of the inherent structures and impose intellectual standards upon the software under test.

iv). He should be able to raise vital questions precisely and clearly, gather and assess relevant information, interpret it effectively in order to come to well-reasoned conclusions and solutions, and test those conclusions against the given criteria and standards.

6. The ability to apply basic and fundamental knowledge

i). Knowledge in the context of testing can be attributed as the fluid mix of experience, values, contextual information and expert insight.

ii). Those things provide a framework for evaluating the system under test. One can attain knowledge by so many means, but that knowledge is worthwhile only when it adds value to situations encountered.

iii). A smart tester should be able to apply the knowledge attained over years of experience with the domain, process, product, customers, mistakes and successes in his testing.

iv). He should be able to make use of fundamental communication, mathematical and software application skills.

v). He should also be able to effectively apply the skills he has attained to practical situations.

7. Continue to learn

i). Organizations and business environments change rapidly, which means the approaches and processes that work well today will be outdated tomorrow.

ii). Therefore, it is imperative that a tester place priority on noticing, adapting and learning from change that is happening around him.

iii). That doesn't mean a tester should continually undergo training or certification, rather he should be open to learning from everything in life that comes he across.

iv). If he has gained basic and fundamental knowledge, then the rest can be achieved through self-directed learning.

v). Learning should be a lifelong habit.

8. Respect for truth and intellectual integrity

i). A tester should be able to examine the piece of software under test and the resulting processes, with focus on the given specification, and understand the behavior of the software.

ii). Being human, a tester may have severe biases, prejudices and intolerances that prevent him from performing well.

iii). He should possess the intellectual integrity to correct those barriers in order to efficiently understand the nature of the software under test.

iv). He should also be willing to shrug off the set of practices and character traits that undermine his intellectual integrity.

9. Planning, time management skills

i). Planning is nothing but writing the story of the future.

ii). A tester needs to have a thorough plan and must develop a well-thought test strategy/approach.

iii). And that plan must be in place before work begins on any software testing assignment.

iv). It should describe the items and features to be tested, the test strategy and levels of testing pass/fail criteria, suspension/resumption criteria, schedule, etc.

v). The plan developed should be monitored continually, and validations should be done through organized system feedback.

vi). Sticking to the plan and monitoring the progress in order to ensure timely delivery is key to any software testing assignment's success.

10. Effective communication skills

i). A tester must be able to communicate his thoughts and ideas effectively, using a variety of tools and media.

ii). He needs to develop and use this skill throughout his career and should learn to communicate effectively to the stakeholders so as to avoid ambiguities and inconsistencies.

iii). For example, printed presentations should be concise and to the point and should follow logically.

iv). The language should be pragmatic rather than philosophical, and arguments should be supported by facts.

v). In the case of oral presentations, the voice, body language and appearance of the presenter are as important as the content and visual aids.

vi). A tester should develop his skills to overcome shyness and any fear of speaking. He should also have good listening skills.

(ii) **Explain decision table with suitable example.** **4M**

Ans: A decision table is a good way to deal with combinations of things (e.g. inputs). This technique is sometimes also referred to as a cause-effect' table. The first task is to identify a suitable function or subsystem which reacts according to a combination of inputs or events.
The system should not contain too many inputs otherwise the number of combinations will become unmanageable. It is better to deal with large numbers of conditions by dividing them into subsets and dealing with the subsets one at a time. Once you have identified the aspects that need to be combined, then you put them into a table listing all the combinations of True and False for each of the aspects.
Following decision table shows the sample example of a credit card black box testing

(Decision table: description: 2marks, example: 2 marks)

Condition	**Rule1**	**Rule2**	**Rule 3**	**Rule 4**
Pin Number	T	T	T	F
Payment Detail	T	F	F	T
Overdue details	F	T	T	F

OR

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.

Example of decision table :
In each column of two conditions mention "Yes" or "No", user will get here four combinations (two to the power of the number of things to be combined). Because of this, it's always good to take small sets of combinations at once. To keep track on combinations, give alternate "Yes" and "No" on the bottom row, put two "Yes" and then two "No" on the row above the bottom row, etc., so the top row will have all "Yes" and then all "No" (Apply the same principle to all such tables).

TABLE 1: Decision table – Input combination

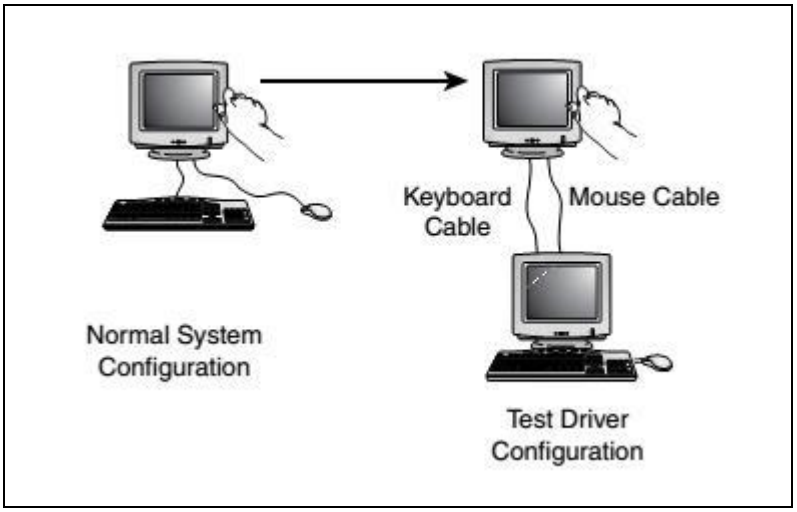
Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned	**Y**	**Y**	**N**	**N**
Terms of loan has been mentioned	**Y**	**N**	**Y**	**N**

(iii) **Describe how drivers and stubs can be used in unit testing with neat diagrams.** **4M**

Ans:

1. Drivers

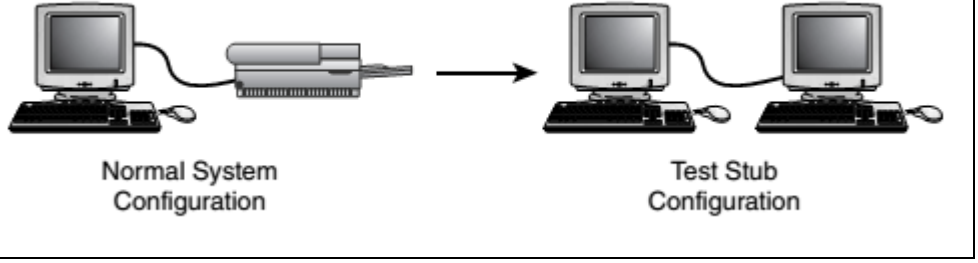
1. Drivers are tools used to control and operate the software being tested.
2. One of the simplest examples of a driver is a batch file, a simple list of programs or commands that are executed sequentially. In the days of MS-DOS, this was a popular means for testers to execute their test programs.
3. They'd create a batch file containing the names of their test programs, start the batch running, and go home.
4. With today's operating systems and programming languages, there are much more sophisticated methods for executing test programs.
5. For example, a complex Perl script can take the place of an old MS-DOS batch file, and the Windows Task Scheduler can execute various test programs at certain times throughout the day as shown in Figure below :



(stubs:2 marks and drivers: 2marks)

2. Stubs

1. Stubs, like drivers, are white-box testing techniques.
2. Stubs are essentially the opposite of drivers in that they don't control or operate the software being tested; they instead receive or respond to data that the software.
3. The Figure shows the general view of stub configuration.



(iv) **Describe two types of test reports.** **4M**

Ans: **(Any two types of test report with explanation: 4marks, 2marks each)**

Test reporting is a means of achieving communication through the testing cycle. There are 3 types of test reporting.

1. Test incident report:

A test incident report is communication that happens through the testing cycle as and when Defects are encountered .A test incident report is an entry made in the defect repository each defect has a unique id to identify incident .The high impact test incident are Highlighted in the test summary report.

2. Test cycle report:

A test cycle entails planning and running certain test in cycle, each cycle using a different build of the product .As the product progresses through the various cycles it is expected to stabilize.

Test cycle report gives

1. A summary of the activities carried out during that cycle.
2. Defects that are uncovered during that cycle based on severity and impact
3. Progress from the previous cycle to the current cycle in terms of defect fixed
4. Outstanding defects that not yet to be fixed in cycle
5. Any variation observed in effort or schedule

3. Test summary report:

The final step in a test cycle is to recommend the suitability of a product for release. A report that summarizes the result of a test cycle is the test summary report.

There are two types of test summary report:

1. Phase wise test summary ,which is produced at the end of every phase
2. Final test summary report.

A Summary report should present

1. Test Summary report Identifier
2. Description : Identify the test items being reported in this report with test id
3. Variances: Mention any deviation from test plans, test procedures, if any.
4. Summary of results: All the results are mentioned here with the resolved incidents and their solutions.
5. Comprehensive assessment and recommendation for release should include Fit for release assessment and recommendation of release.

6

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

(i) What are sanity testing and smoke testing. Describe.

Ans: **Smoke Testing** is a testing technique that is inspired from hardware testing, which checks for the smoke from the hardware components once the hardware's power is switched on. **(Sanity testing: 3 marks and Smoke testing: 3marks)**

i) In Software testing context, smoke testing refers to testing the basic functionality of the build.

ii) If the Test fails, build is declared as unstable and it is NOT tested anymore until the smoke test of the build passes.

Features of smoke testing are :

i) Identifying the business critical functionalities that a product must satisfy.

ii) Designing and executing the basic functionalities of the application.

iii) Ensuring that the smoke test passes each and every build in order to proceed with the testing.

iv) Smoke Tests enables uncovering obvious errors which saves time and effort of test team.

v) Smoke Tests can be manual or automated.

Sanity testing, a software testing technique performed by the test team for some basic tests. The aim of basic test is to be conducted whenever a new build is received for testing. The terminologies such as Smoke Test or Build Verification Test or Basic Acceptance Test or Sanity Test are interchangeably used; however, each one of them is used under a slightly different scenario.

vi) Sanity test is usually unscripted, helps to identify the dependent missing functionalities. It is used to determine if the section of the application is still working after a minor change.

vii) Sanity testing can be narrow and deep. Sanity test is a narrow regression test that focuses on one or a few areas of functionality.

(ii) Describe the requirement defects and coding defects in details. **6M**

Ans: **Requirements defects**: A valid requirement which is mandatory and supposed to code (or implement) is missed. The root cause of this defect is due to not capturing this requirement in the specification document. These types of observations are categorized as 'Missed requirements defects' since the requirements are missed from requirement specification document. **(Requirement defects : 3 marks and coding defects: 3 marks)**

These types of defects are usually caused by business analyst's oversight.

Example:

Tester observation while testing the Website: Tester found that 'Disclaimer' link is missing in

		the website. According to organization/webmaster guidelines it is mandatory to show 'Disclaimer' link in the website so tester expressed his/her concern that the 'Disclaimer' link is missing and to fix this developer expects the business analyst to document in requirement specification document. **Coding defects**: The requirements have been coded incorrectly due to which behavior of an implemented software function is not in accordance with the requirement specification documents. The other frequently occurred defects made by developers are due to • Missed to code for the requirements which listed in requirement specification document • Coding the requirements which are not specified in the requirement specification document These types of defects are usually caused by developer's oversight. **Example:** Requirement as per specification document: If user clicks on 'Home' link in a website then 'Home' page should be presented to the user. Tester observation while testing the 'Home' link: Tester found that 'About Me' page is displayed each time the 'Home' link is clicked which is a deviation in the behavior from the requirement specification document. This is an example of coding defect.	
2.		**Attempt any <u>FOUR</u> of the following :**	**16**
	a)	**What factors shall be considered while selecting resource requirements?**	**4M**
	Ans:	1. Machine configuration (RAM, processor, disk, and so on) needed to run the product under test. 2. Overheads required by the test automation tool, if any 3. Supporting tools such as compilers , test data generators configuration management tools, and so on 4. The different configurations of the supporting software (for example, OS) that must be present 5. Special requirements for running machine- intensive tests such as load tests and performance tests 6. Appropriate number of license of all the software **OR** **Factors to be considered while selecting the resource requirements are :** • **People:** How many people are required? How much experience they should posses? What kind of experience is needed? What should they be expertise in? Should they be full-time, part-time, contract, students? • **Equipment:** How many Computers are required? What configuration computers will be required? What kind of test hardware is needed? Any other devices like printers, tools etc. • **Office and lab space:** Where will they be located?	***(Any four Factors: 1 mark each)***

How big will they be?
How will they be arranged?

- **Software**: Word processors, databases, custom tools. What will be purchased, what needs to be written?
- **Outsource companies:** Will they be used? What criteria will be used for choosing them? How much will they cost?
- **Miscellaneous supplies:** Disks, phones, reference books, training material. What else might be necessary over the course of the project? The specific resource requirements are very project-, team-, and company-dependent, so the test plan effort will need to carefully evaluate what will be needed to test the software.

b) **What are the advantages and disadvantages of using automated tools for testing? List any two each.** **4M**

Ans: **(Two advantages: 2 marks, 2 disadvantages: 2 marks)**

Advantages of using automated tools for testing are as follows :

1. Automated Software Testing Saves Time and Money

i) Software tests have to be repeated often during development cycles to ensure quality. Every time source code is modified software tests should be repeated.
ii) For each release of the software it may be tested on all supported operating systems and hardware configurations.
iii) Manually repeating these tests is costly and time consuming. Once created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests.
iv) Automated software testing can reduce the time to run repetitive tests from days to hours.
v) A time savings that translates directly into cost savings.

2. Testing Improves Accuracy

i) Even the most conscientious tester will make mistakes during monotonous manual testing.
ii) Automated tests perform the same steps precisely every time they are executed and never forget to record detailed results.

3. Increase Test Coverage

i) Automated software testing can increase the depth and scope of tests to help improve software quality.
ii) Lengthy tests that are often avoided during manual testing can be run unattended.
iii) They can even be run on multiple computers with different configurations.
iv) Automated software testing can look inside an application and see memory contents, data tables, file contents, and internal program states to determine if the product is behaving as expected.
v) Automated software tests can easily execute thousands of different complex test cases during every test run providing coverage that is impossible with manual tests.
vi) Testers freed from repetitive manual tests have more time to create new automated software tests and deal with complex features.

4. Automation Does What Manual Testing Cannot

i) Even the largest software departments cannot perform a controlled web application test with thousands of users.

ii) Automated testing can simulate tens, hundreds or thousands of virtual users interacting with network or web software and applications.

Disadvantages of using automated tools for testing are as follows:

1. High investments required in package purchasing and training.
2. High package development investment costs.
3. High manpower requirements for test preparation.
4. Considerable testing areas left uncovered.

c) What is load testing and stress testing? Describe with respect to system testing. **4M**

Ans: **(load testing : 2 marks, stress testing : 2 marks)**

Stress testing is testing the software under less than ideal conditions. So subject your software to low memory, low disk space, slow cpus, and slow modems and so on. Look at your software and determine what external resources and dependencies it has. Stress testing is simply limiting them to bare minimum. With stress testing you starve the software.

For e.g. Word processor software running on your computer with all available memory and disk space, it works fine. But if the system runs low on resources you had a greater potential to expect a bug. Setting the values to zero or near zero will make the software execute different path as it attempt to handle the tight constraint. Ideally the software would run without crashing or losing data.

Load testing is testing the software under customer expected load. In order to perform load testing on the software you feed it all that it can handle. Operate the software with largest possible data files. If the software operates on peripherals such as printer, or communication ports, connect as many as you can. If you are testing an internet server that can handle thousands of simultaneous connections, do it. With most software it is important for it to run over long periods. Some software's should be able to run forever without being restarted. So Time acts as a important variable.

Stress testing and load testing can be best applied with the help of automation tools**.**

Stress testing and load testing are the types of performance testing.

The Microsoft stress utility program allows you to individually set the amounts of memory, disk space, files and other resources available to the software running on the machine.

Example: Open many number of browsers in the windows simultaneously.
Connect more than the specifies clients to the server.
Connect more than one printer to the system.

d) List what are the different guidelines to be followed while selecting dynamic test tools. **4M**

Ans: **(Guidelines: 4marks*)***

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 mentoring for those new to the use of the tool).

e) **Draw labeled diagram of defect management process. List any two characteristics of defect management process.** **4M**

Ans: **Defect Management Process diagram :**

(Defect management process Diagram :2 marks , any two characteristics of defect management process : 2 marks)

Characteristics of defect management process are :

1. The primary goal is to prevent defects. Where this is not possible or practical, the goals are to both find the defect as quickly as possible and minimize the impact of the defect.
2. The defect management process should be risk driven-- i.e., strategies, priorities, and resources should be based on the extent to which risk can be reduced.
3. Defect measurement should be integrated into the software development process and be used by the project team to improve the process. In other words, the project staff, by doing their job, should capture information on defects at the source. It should not be done after-the-fact by people unrelated to the project or system
4. As much as possible, the capture and analysis of the information should be automated.
5. Defect information should be used to improve the process. This, in fact, is the primary reason for gathering defect information.
6. Most defects are caused by imperfect or flawed processes. Thus to prevent defects, the process must be altered.

f) **Why boundary value analysis is required? Give example.** **4M**

Ans:

i) Boundary conditions are special because programming, by its nature, is susceptible to problems at its edges.
ii) The boundary conditions are defined as the initial and the final data ranges of the variables declared.
iii) If an operation is performed on a range of numbers, odds are the programmer got it right for the vast majority of the numbers in the middle, but maybe made a mistake at the edges.
iv) The edges are the minimum and the maximum values for that identifier.

```
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++)
```

(Need for BVA : 2 marks, example : 2 marks)

```
8. fact = fact * i;
9. printf ("the factorial of a number is □ "%d", fact);
10. }
```

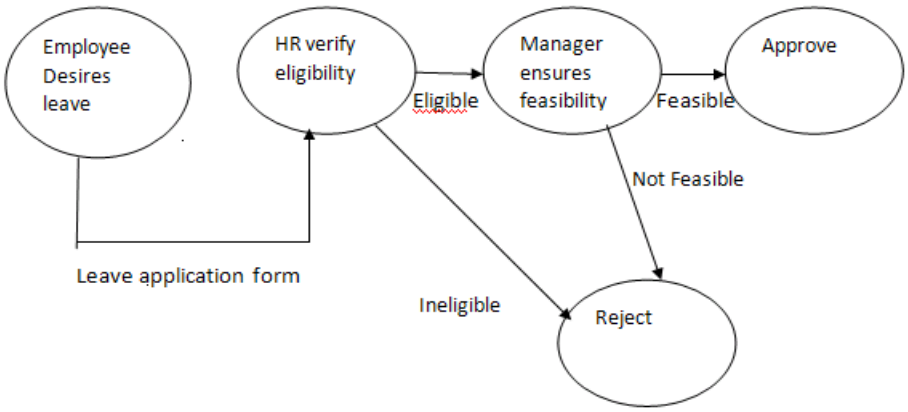
The boundary condition in the above example is for the integer variable.

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

a) **Describe graph based testing with appropriate diagram.** **4M**
{**Note: Any other relevant diagram also shall be considered**}

Ans: State or graph based testing is useful in situations where **(Diagram :2 marks, Explanation:2 marks)**

1. The product under test is a language processor wherein the syntax of the language automatically lends itself to a state machine or context free grammar represented by a railroad diagram.
2. Workflow modeling where, depending on the current state and appropriate combinations of input variables, specific workflows are carried out resulting in new output and new state Dataflow modeling, where the system is modeled as a set of dataflow, leading from one state to another.

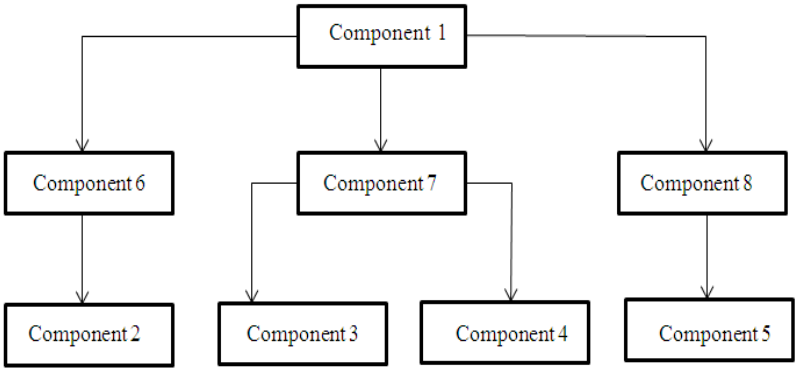


In the above case, each of the states (represented by circles) is an event or a decision point while the arrows or lines between the states represent data inputs. This can be applicable when

1. The application can be characterized by a set of states.
2. The data values (screen, mouse clicks) that cause the transition from one state to another is well understood.
3. The methods of processing within each state to process the input received is also well understood.

b) **Describe bidirectional/sandwich integration testing with neat diagram.** **4M**

Ans: Bi- directional integration is a combination of the top-down & bottom-up integration approaches used together to derive integration steps. **(Diagram : 2 marks, Explanation:2 marks)**



As shown in fig, assume that the software components become available in the order mentioned by the component numbers. The individual components 1, 2, 3, 4 and 5 are tested separately and bi-directional integration performed initially with the use of stubs and drivers. Drivers are used to provide upstream connectivity while stubs provide downstream connectivity .A driver is a function which redirects the requests to some other components and stubs simulate the behavior of missing components. After the functionality of these integrated components is tested, the drivers and stubs are discarded. Once components 6, 7 and 8 become available, the integration methodology then focus only on those components, as there are the components which need focus and are new. This approach is also called ***"Sandwich Integration".***

Advantages:

1. Sandwich approach is useful for very large projects having several subprojects.
2. Both Top-down and Bottom-up approach starts at a time as per development schedule.
2. 3. Units are tested and brought together to make a system Integration is done downwards.

Disadvantages:

1. It require very high cost for testing because one part has Top-down approach while another part has bottom-up approach.
2. It cannot be used for smaller system with huge interdependence between different modules. It makes sense when the individual subsystem is as good as complete system.

c) **Write four test cases to test sign-in form of gmail account.** **4M**

Ans: ***{**Note: Any other relevant test cases shall be considered**}***

Test Case Id	Description	Input Data	Expected Result	Actual Result	Status
TC1	Login(email id) Field of Sign-in form of Gmail	Enter "abc123" and click on "Next" button	It shall prompt to enter Password	It prompts to enter password	Pass
TC2	Password field of Sign-in form of Gmail	Enter "xyz" (valid id)and click on "Sign in" button	It shall open Gmail account	It opens Gmail account	Pass
TC3	Login(email id) Field of Sign-in form	Without entering login id, click on	It shall give message as "Please enter	It gives message as "Please enter	Pass

(Any 4 test cases: 1 mark each)

	of Gmail	"Next" button	your email"	your email"	
TC4	Password field of Sign-in form of Gmail	Without entering password, click on "Sign in" button	It shall give message as "Please enter your password"	It gives message as "Please enter your password"	Pass
TC5	Login(email id) Field of Sign-in form of Gmail	Enter "pqr" (invalid id)and click on "Next button"	It shall give message as "Sorry, Google doesn't recognize that email"	It give message as" Sorry, Google doesn't recognize that email"	Pass
TC6	Password field of Sign-in form of Gmail	Enter "abc123" (invalid password) at password field after entering valid login id.	It shall give message as "Wrong password. Try again"	It gives message "Wrong password. Try again"	Pass

d) **Compare alpha testing and beta testing.(Any four differences)** **4M**

Ans:

Alpha Testing	Beta Testing
1. Performed at Developer's site.	1. Performed at End user's site.
2. Performed in controlled Environment as developer is present.	2. Performed in uncontrolled Environment as developer is not present.
3. Less probability of finding of errors as it is driven by developer.	3. High probability of finding errors as end user can use it the way he wants.
4. It is done during implementation phase of software	4. It is done as pre-release of software.
5. It is not considered as live application	5. It is considered as live application.
6. Less time consuming as developer can make necessary changes in given time.	6. More time consuming. As user has to report bugs if any via appropriate channel

(Any4 points: 1 mark each)

e) **Explain three types of product metrics.** **4M**

Ans:

Product Metrics is classified as :

1. **Project Metrics**: A set of metrics that indicates how the project is planned and executed.
2. **Progress Metrics:** A set of metrics that tracks how the different activities of the project are progressing. It includes both development activities and testing activities. Progress Metrics is classified as a. Test defect metrics b. Development defect metrics
   a. **Test defect metrics:** help the testing team in analysis of product quality and testing
   b. **Development defect metrics:** help the development team in analysis of development

(Listing types : 1 mark ,Explanation : 3 marks)

activities.

3. **Productivity Metrics:** A set of metrics that takes into account various productivity numbers that can be collected and used for planning and tracking testing activities. These metrics help in planning and estimating of testing activities.

4. a. Attempt any <u>THREE</u> of the following. **12**

(i) Describe structural walk through under static testing. **4M**

Ans: One of the static testing methods is structural walkthrough.
In walkthroughs, a set of people look at the program code and raise questions for the author. The author explains the logic of the code and answers the questions. If the author is unable to answer some questions, he or she then takes those questions and finds their answers.

(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.

(Walk through explanation : 4 marks)

(ii) Describe any four limitations of manual testing. **4M**

Ans: **Limitations of Manual Testing are as given below :**

- Manual testing is slow and costly
- It is very labor intensive, it takes a long time to complete tests.
- Manual tests don't scale well. As the complexity of the software increases the complexity of the testing problem grows exponentially. This leads to an increase in total time devoted to testing as well as total cost of testing.
- Manual testing is not consistent or repeatable. Variations in how the tests are performed as inevitable, for various reasons. One tester may approach and perform a certain test differently from another, resulting in different results on the same test, because the tests are not being performed identically.
- Lack of training is the common problem, although not unique to manual software testing.
- GUI objects size difference and color combinations are not easy to find in manual testing.
- Not suitable for large scale projects and time bound projects.
- Batch testing is not possible, for each and every test execution Human user interaction is mandatory.
- Comparing large amount of data is impractical.
- Processing change requests during software maintenance takes more time.

OR

Limitations of manual testing:

- Time consuming process: manual testing consumes much time because most pf test operations or test case are repeated again and again.
- Limited support for regression testing: when one portion of software is changed then regression testing is performed to ensure that other portions are not infected by the recent

(Any 4 points: 1 mark each)

change .in this process entire testing is repeated which take more time.

- Error Prone Testing: as test process is repeated several times test engineers become bored which may result in missing some important test case and leads to release defective software.
- Impractical performance testing: In doing performance testing of any client server application resources like people and computers are required. client applications has to be installed on several machines and it is supposed to be tested by one person to see the performance of whole software which is very tedious and time consuming task
- Non consistent: Manual testing process is not consistent as testing methods and approaches applied by every person are not remaining same. This produces different result on same test so variation in test process in unavoidable. There is no standardization
- Limited scope: not suitable for the time bounded and large scale projects as scope of manual testing is very limited as compared to automated testing. As the size of software

iii) **Explain the defect template with its attribute.** **4M**

Ans: **DEFECT 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. **(Template with attributes :4 marks)**

i). Reporting a bug/defect properly is as important as finding a defect.

ii). If the defect found is not logged/reported correctly and clearly in bug tracking tools (like Bugzilla, Clear Quest etc.) then it won't be addressed properly by the developers, so it is very important to fill as much information as possible in the defect template so that it is very easy to understand the actual issue with the software.

Sample defect template
Abstract :
Platform :
Test case Name :
Release :
Build Level :
Client Machine IP/Hostname :
Client OS :
Server Machine IP/Hostname :
Server OS :
Defect Type :
Priority :
Severity :
Developer Contacted :
Test Contact Person :
Attachments :
Any Workaround :
Steps to Reproduce 1. 2. 3.
Expected Result:
Actual Result:

OR

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)

iv) Explain any two internal standards in test management. **4M**

Ans: **Internal standards are:** **(Any 2 standards explanation: 2 marks each)**

1. Naming and storage conventions for test artifacts.
2. Document standards
3. Test coding standards
4. Test reporting standards.

1. Naming and storage conventions for test artifacts: Every test artifacts(test specification, test case, test results and so on)have to be named appropriately and meaningfully. It enables
   a) Easy identification of the product functionality.
   b) Reverse mapping to identify the functionality corresponding to a given set of tests. e.g. modules shall be M01,M02. Files types can be .sh, .SQL.
2. Documentation standards:
   a) Appropriate header level comments at the beginning of a file that outlines the functions to be served by the test.
   b) Sufficient inline comments, spread throughout the file
   c) Up-to-Date change history information, reading all the changes made to the test file.

3. Test coding standards:
   a) Enforce right type of initialization
   b) Stipulate ways of naming variables.
   c) Encourage reusability of test artifacts
   d) Provide standard interfaces to external entities like operating system, hardware and so on.
4. Test reporting standard: All the stakeholders must get a consistent and timely view of the progress of tests. It provides guidelines on the level of details that should be present in the test report, their standard formats and contents.

b. **Attempt any ONE of the following:** **6**

(i) **Explain V model with diagram.** **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.

(Diagram : 2 marks, verification Explanation: 2 marks, validation Explanation: 2 marks)

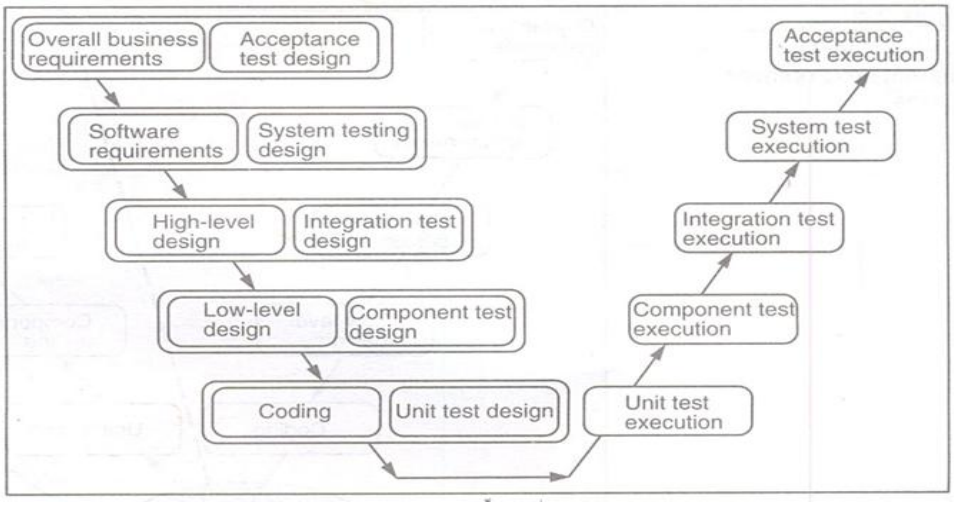
Verification Phase:

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.
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.

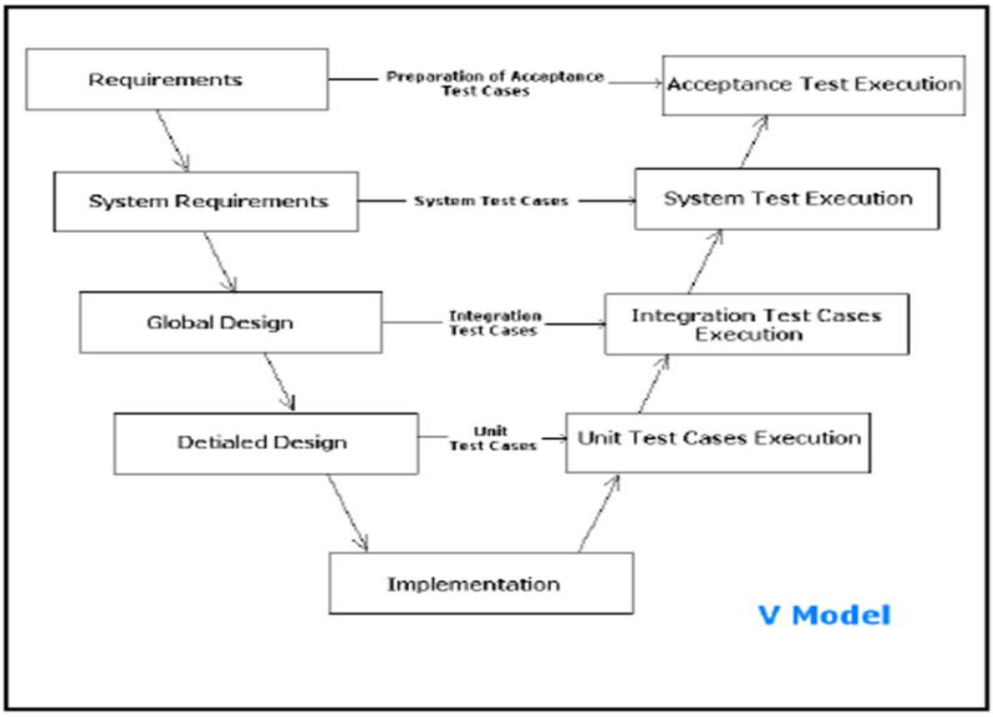
Validation:

1. **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.
2. **Components testing:** This is associated with module design helps to eliminate defects in individual modules.
3. **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

4. **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.
5. **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.



OR



Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence 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.

Disadvantages of V-model:

- 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.

(ii) **What is equivalence partitioning? Give example.** **6M**

{**Note: Any other relavant example also shall be considered**}

Ans:

(Explanation: 4marks, Example: 2 marks)

Equivalence partitioning is a software technique that involves identifying a small set of representative input values that produce as much different output condition as possible. It is use to reduce test cases. This reduces the number of permutation & combination of input, output values used for testing, thereby increasing the coverage and reducing the effort involved in testing.

The set of input values that generate one single expected output is called a partition. When the behavior of the software is the same for a set of values, then the set is termed as equivalence class or partition.

Example: 1.An insurance company that has the following premium rates based on the age group.

A life insurance company has base premium of $0.50 for all ages. Based on the age group, an additional monthly premium has to pay that is as listed in the table below. For example , a person aged 34 has to pay a premium=$0.50 +$ 1.65=$2.15

2.Based on the equivalence portioning technique, the equivalence partitions that are based on marks are given below:

Marks	**result**
Under 40	fail
40-100	pass

Equivalence partitioning:

Above	40 marks in subject	(valid input)
Between	40 and 100 marks	(valid input)
Below	40 marks	(invalid input)
Below	Negative marks	(Invalid Input)

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

a) **Write the entity, purpose and attributes of the following elements of test infrastructure management.** **8M**

i) A test case database(TCDB)

ii) A defect repository

Ans: **i). Test case database(TCDB)** **(Test case database(TCDB) : 4 marks, Defect repository: 4 marks)**

Entity	Purpose	Attributes
Test case	Records all the "static" information about the tests	• Test case ID • Test case name (filename) • Test case owner • Associated files for the test case
Test case- product cross-reference	Provides a mapping between the tests and the corresponding product features ; enables identification of tests for a given feature	• Test case ID • Modulate ID
Test case run history	Gives the history of when a test was run and what was the result; provides inputs on selection of tests for regression runs (see chapter 8)	• Test case ID • Run date • Time taken • Run status (success/failure)
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 tests for regression runs	• Test case ID • Defect reference# (points to a record in the defect repository)

ii). Defect Repository:

Entity	Purpose	Attributes
Defect details	Records all the " static" information about the tests	• Defect ID • Defect priority /severity • Defect description • Affected product(s) • Any relevant version information (for example, OS version) • Customers who encountered the problem (could be reported by the internal testing team also) • Date and time of defect occurrence
Defect test details	Provides details of test cases for a given defect. Cross-references the TCDB	• Defect ID • Test case ID
Fix details	Provides details of fixes for a given defect; cross-references the configuration management repository	• Defect ID • Fix details (file changed ,fix release information)
Communication	Captures all the details of the communication that transpired for this defect among the various stakeholders these could include communication between the testing team and development team, customer communication, and so on. Provides insights into effectiveness of communication	• • Test case ID • Defect reference # • Details of communication

b) **What is code coverage testing? Explain the following types of coverage:** **8M**

i) Path coverage

ii) Condition coverage

Ans:

Code Coverage Testing

(Code coverage testing with Example: 4 marks, Path coverage and condition testing: 2 marks each.)

(1) 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).

(2) 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.

(3) 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)        }
(11)
```

The declaration of data is complete with the assignment statement and the variable declaration statements. All the variable declared are properly utilized.

i) Path Coverage

1. The most straightforward form of code coverage is called statement coverage or line coverage.
2. 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.
3. 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)    }
```

ii) Condition Coverage

1. Attempting to cover all the paths in the software is called path testing.
2. The simplest form of path testing is called branch coverage testing.
3. To check all the possibilities of the boundary and the sub boundary conditions and it's branching on those values.
4. 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.
5. Every branch (decision) taken each way, true and false.
6. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.
7. Just when you thought you had it all figured out, there's yet another complication to path testing.
8. Condition coverage testing takes the extra conditions on the branch statements into account.

c) **List any four features of client server application. Explain any four testing approaches of client – server testing.** **8M**

Ans:

i) This type of testing usually done for 2 tier applications (usually developed for LAN) Here we will be having front-end and backend.
ii) The application launched on front-end will be having forms and reports which will be monitoring and manipulating data. E.g: applications developed in VB, VC++, Core Java, C, C++, D2K, Power Builder etc.
iii) The backend for these applications would be MS Access, SQL Server, Oracle, Sybase, MySQL, Quadbase.
iv) The tests performed on these types of applications would be– User interface testing Manual support testing– Functionality testing– Compatibility testing & configuration testing – Intersystem testing.

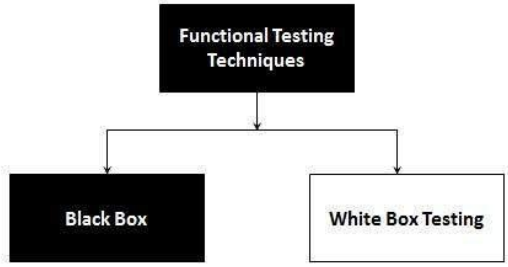
(Client Server Testing: 4 marks, Approaches : 4 marks, 1 mark each)

The approaches used for client server testing are

1. **User interface testing**: User interface testing, a testing technique used to identify the presence of defects is a product/software under test by using Graphical user interface [GUI].GUI Testing - Characteristics:
   i) GUI is a hierarchical, graphical front end to the application, contains graphical objects with a set of properties.
   ii) During execution, the values of the properties of each objects of a GUI define the GUI state.
   iii) It has capabilities to exercise GUI events like key press/mouse click.
   iv) Able to provide inputs to the GUI Objects.
   v) To check the GUI representations to see if they are consistent with the expected ones.
   vi) It strongly depends on the used technology.
2. **Manual testing**: Manual testing is a testing process that is carried out manually to find defects without the usage of tools or automation scripting. A test plan document is prepared that acts as a guide to the testing process to have the complete test coverage. Following are the testing techniques that are performed manually during the test life cycle are Acceptance Testing, White Box Testing, Black Box Testing, Unit Testing, System Testing, Integration Testing.

3. **Functional testing**: Functional Testing is a testing technique that is used to test the features/functionality of the system or Software, should cover all the scenarios including failure paths and boundary cases.

There are two major Functional Testing techniques as shown below:



4. **Compatibility testing**: Compatibility testing is a non-functional testing conducted on the application to evaluate the application's compatibility within different environments. It can be of two types - forward compatibility testing and backward compatibility testing.
   - Operating system Compatibility Testing - Linux , Mac OS, Windows
   - Database Compatibility Testing - Oracle SQL Server
   - Browser Compatibility Testing - IE , Chrome, Firefox
   - Other System Software - Web server, networking/ messaging tool, etc.

6. **Attempt any <u>FOUR</u> of the following:** **16**

a) **List the different techniques to detect defects. Describe any two of them.** **4M**

Ans: **Static Techniques**: Static techniques of quality control define checking the software product and related artifacts without executing them. It is also termed 'desk checking/verification /white box testing'. It may include reviews, walkthroughs, inspection, and audits Here; the work product is reviewed by the reviewer with the help of a checklist, standards, any other artifact, knowledge and experience, in order to locate the defect with respect to the established criteria. Static technique is so named because it involves no execution of code, product, documentation, etc. This technique helps in establishing 'conformance to requirements 'view.

Dynamic Testing: Dynamic testing is a validation technique which includes dummy or actual execution of work products to evaluate it with expected behavior. It includes black box testing methodology such as system testing and unit testing. The testing methods evaluate the product with respect to requirements defined, designs created and mark it as 'pass 'or 'fail'. This technique establishes 'fitness for use' view.

Operational techniques: Operational techniques typically include auditing work products and projects to understand whether the processes defined for development /testing are being followed correctly o not, and also whether they are effective or not. It also includes revisiting the defects before and after fixing and analysis. Operational technique may include smoke testing and sanity testing of a work product.

(List: 1 mark, Describing: 1.5 marks each)

OR

The various techniques to detect defects are

a) Quick attacks.
b) Equivalence and Boundary Conditions
c) Common Failure Modes
d) State-Transition Diagrams
e) Use Cases and Soap Opera Tests

a) Quick Attacks:

i. Strengths

- 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.
- The skill is relatively easy to learn, and once you've attained some mastery your quick-attack session will probably produce a few bugs.
- Finally, quick attacks are ***quick***.
- They can help you to make a rapid assessment. You may not know the requirements, but if your attacks yielded a lot of bugs, the programmers probably aren't thinking about exceptional conditions, and it's also likely that they made mistakes in the main functionality.
- If your attacks don't yield any defects, you may have some confidence in the general, happy-path functionality.

ii. Weaknesses

- Quick attacks are often criticized for finding "bugs that don't matter"— especially for internal applications.
- While easy mastery of this skill is strength, it creates the risk that quick attacks are "all there is" to testing; thus, anyone who takes a two-day course can do the work.

b) Equivalence and Boundary Conditions

i. Strengths

- 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".

ii. Weaknesses

- The "classes" in the table in Figure 1 are correct only in the mind of the person who chose them.
- We have no idea whether other, "hidden" classes exist—for example, if a numeric number that represents time is compared to another time as a set of characters, or a "string," it will work just fine for most numbers.

c) Common Failure Modes

i. Strengths

- 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.

ii. Weaknesses

- In addition to losing its potency over time, this technique also entirely fails to find "black swans"—defects that exist outside the team's recent experience.

- The more your team stretches itself (using a new database, new programming language, new team members, etc.), the riskier the project will be—and, at the same time, the less valuable this technique will be.

d) State-Transition Diagrams

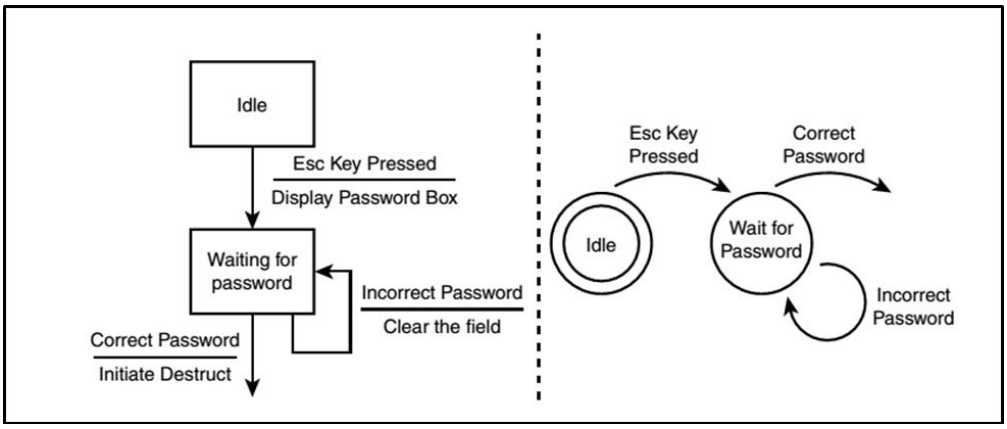


Figure 4: State Transition Map

i. Strengths

- 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.

ii. Weaknesses

- 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, and it might even give the team the illusion of certainty.
- 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 and Soap Opera Tests

Use cases and scenarios focus on software in its role to enable a human being to do something.

i. Strengths

- 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.
- They make sense and can provide a straightforward set of confirmatory tests. Soap opera tests offer more power, and they can combine many test types into one execution.

ii. Weaknesses

- Soap opera tests have the opposite problem; they're so complex that if something goes wrong, it may take a fair bit of troubleshooting to find exactly where the error came from!

f) Code-Based Coverage Models

Imagine that you have a black-box recorder that writes down every single line of code as it executes.

i. Strengths

- Programmers love 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.
- Meanwhile, looking at the code that ***isn't*** covered also can yield opportunities for improvement and bugs!

ii. Weaknesses

- 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.
- Even decision coverage doesn't deal with situations where the decision contains defects, or when there are other, hidden equivalence classes; say, in the third-party library that isn't measured in the same way as your compiled source code is.
- Having code-coverage numbers can be helpful, but using them as a form of process control can actually encourage wrong behaviours. In my experience, it's often best to leave these measures to the programmers, to measure optionally for personal improvement (and to find dead spots), not as a proxy for actual quality.

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.

i. Strengths

- For the right kind of problem, say an IT shop processing files through a database, this kind of technique can be extremely powerful.
- Likewise, if the software deliverable is a report written in SQL, you can hand the problem to other people in plain English, have them write their own SQL statements, and compare the results.
- Unlike state-transition diagrams, this method shines at finding the hidden state in devices. For a pacemaker or a missile-launch device, finding those issues can be pretty important.

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.
- For the most part, these techniques seem to have found a niche in IT/database work, at large companies like Microsoft and AT&T, which can have programming testers doing this work in addition to traditional testing, or finding large errors such as crashes without having to understand the details of the business logic.
- While some software projects seem ready-made for this approach, others...aren't.
- You could waste a fair bit of money and time trying to figure out where your project falls.

b) **Differentiate between quality assurance and quality control. (Any four points)** **4M**

Ans:

(Any four points :1 mark each)

Quality Assurance	Quality Control
Process oriented activities.	Product oriented activities.
QA is the process of managing for quality.	QC is used to verify the quality of the output
They measure the process, identify the deficiencies/weakness and suggest improvements.	They measure the product, identify the deficiencies/weakness and suggest improvements.
Relates to all products that will ever be created by a process	Relates to specific product
Activities of QA are Process Definition and Implementation, Audits and Training	Activities of QC are Reviews and Testing
Verification is an example of QA	Validation/Software Testing is an example of QC
Preventive activities.	It is a corrective process.
Quality assurance is a proactive process	Quality control is a reactive process.
QA is a managerial tool	QC is a corrective tool

OR

Quality Assurance:

i. A part of quality management focused on providing confidence that quality requirements will be fulfilled.
ii. All the planned and systematic activities implemented within the quality system that can be demonstrated to provide confidence that a product or service will fulfill requirements for quality
iii. Quality Assurance is fundamentally focused on planning and documenting those processes to assure quality including things such as quality plans and inspection and test plans.
iv. Quality Assurance is a system for evaluating performance, service, of the quality of a product against a system, standard or specified requirement for customers.
v. Quality Assurance is a complete system to assure the quality of products or services. It is not only a process, but a complete system including also control. It is a way of management.

Quality Control

i. A part of quality management focused on fulfilling quality requirements.
ii. The operational techniques and activities used to fulfill requirements for quality.
iii. Quality Control on the other hand is the physical verification that the product conforms to these planned arrangements by inspection, measurement etc.
iv. Quality Control is the process involved within the system to ensure job management, competence and performance during the manufacturing of the product or service to ensure it meets the quality plan as designed.
v. Quality Control just measures and determines the quality level of products or services.

c) **What are the different points to be noted in reporting defects?** **4M**

Ans: **(Any four points :1 mark each)**

It is essential that you report defects effectively so that time and effort is not unnecessarily wasted in trying to understand and reproduce the defect. Here are some guidelines:

i. **Be specific:**
- Specify the exact action: Do not say something like 'Select Button B'.
- Do you mean 'Click Button B' or 'Press ALT+B' or 'Focus on Button B and click ENTER'.
- In case of multiple paths, mention the exact path you followed: Do not say something like "If you do 'A and X' or 'B and Y' or 'C and Z', you get D." Understanding all the paths at once will be difficult. Instead, say "Do 'A and X' and you get D." You can, of course, mention elsewhere in the report that "D can also be got if you do 'B and Y' or 'C and Z'."
- Do not use vague pronouns: Do not say something like "In Application A, open X, Y, and Z, and then close it." What does the 'it' stand for? 'Z' or, 'Y', or 'X' or 'Application A'?"

ii. **Be detailed:**
- Provide more information (not less). In other words, do not be lazy.
- Developers may or may not use all the information you provide but they sure do not want to beg you for any information you have missed.

iii. **Be objective:**
- Do not make subjective statements like "This is a lousy application" or "You fixed it real bad."
- Stick to the facts and avoid the emotions.

iv. **Reproduce the defect:**
- Do not be impatient and file a defect report as soon as you uncover a defect. Replicate it at least once more to be sure.

v. **Review the report:**
- Do not hit 'Submit' as soon as you write the report.
- Review it at least once.
- Remove any typing errors.

d) **What are the things that test case specification shall identify?** **4M**

Ans:

(1 mark each, or any relevant point)

1. Test cases specify the inputs, predicted results and execution conditions. Each test case should aim to evaluate the operation of a key element or function of the system.
2. Failure of a test case, depending upon the severity of the failure, would be catalogued as part of the overall evaluation of the suitability of the system for its intended use.
3. Test cases can start with a specific 'form' that allows operator entry of data into the system. This needs to be mapped, if the architecture is based upon an n-tier solution, through the business logic and rules into the server systems with transactions being evaluated both in a 'nominal' mode where the transaction is a success and for those occasions when the transaction or 'thread' fails.
4. Test design may also require one or more test cases and one or more test cases may be executed by a test procedure.

e) **Describe any four testing approaches of web application.** **4M**

Ans: Web application testing, a software testing technique exclusively adopted to test the applications that are hosted on web in which the application interfaces and other functionalities are tested. **(Any four testing approaches:1 mark each)**

Web Application Testing - Techniques:

(1) Functionality Testing - The below are some of the checks that are performed but not limited to the below list:
- Verify there is no dead page or invalid redirects.
- First check all the validations on each field.
- Wrong inputs to perform negative testing.
- Verify the workflow of the system.
- Verify the data integrity.

(2) Usability testing - To verify how the application is easy to use with.
- Test the navigation and controls.
- Content checking.
- Check for user intuition.

(3) Interface testing - Performed to verify the interface and the dataflow from one system to other.

(4) Compatibility testing- Compatibility testing is performed based on the context of the application.
- Browser compatibility
- Operating system compatibility
- Compatible to various devices like notebook, mobile, etc.

(5) Performance testing - Performed to verify the server response time and throughput under various load conditions.
- **Load testing** - It is the simplest form of testing conducted to understand the behaviour of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc. are also monitored.
- **Stress testing** - It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.
- **Soak testing** - Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues. The main aim is to discover the system's performance under sustained use.
- **Spike testing** -Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the work load.

(6) Security testing - Performed to verify if the application is secured on web as data theft and unauthorized access are more common issues and below are some of the techniques to verify the security level of the system.
- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure

		• Missing Function Level Access Control • Cross-Site Request Forgery (CSRF) • Using Components with Known Vulnerabilities • Invalidated Redirects and Forwards	