SUMMER-16 EXAMINATION
Model Answer

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 judgment 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.

1. Attempt any FIVE of following: **5×4=20**

(a) Explain the Static testing and dynamic testing.
(Static testing - 2 marks; Dynamic testing - 2 marks)

Ans:

Static Testing: In static testing code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors. Main objective of this testing is to improve the quality of software products by finding errors in early stages of the development cycle. The techniques used in this such as walkthrough, Inspection.
Dynamic testing: The dynamic is testing done by executing program. Main objective of this testing is to confirm that the software product works in conformance with the business requirements. The techniques used are unit testing, integration testing, system testing.

(b) What is software testing? State objectives of software testing.
(Definition - 1 mark; any 3 objectives - 3 marks)

Ans:

Software testing is a method of assessing the functionality of a software program.

OR

Software testing is the process of validating and verifying that a software program or application or product, meets the business and technical requirements that guided its design and development.

Objectives of software testing

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.

(c) What is a 'test case'? State its specification parameter.
(Definition- 1 mark; Parameters - 3 marks)

Ans:

Test case is a well-documented procedure designed to test the functionality of the feature in the system.

OR

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

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: The prerequisite for running in the system.
4. Input: Actual inputs must be provided, instead of general inputs.
5. Expected outputs: which should be produced when there is no failure?
6. Actual outputs: What is the actual output when the code is executed.
7. Status: If Expected and actual result is same status is Pass otherwise it is Fail.

(d) What is software test automation? State types of test automation tools.
(*Test automation - 2 marks; Types – 2 marks*)

Ans:

Test automation is the use of special software to control the execution of tests and the comparison of actual outcomes with predicted outcomes. The objective of automated testing is to simplify as much of the testing effort as possible with a minimum set of scripts. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually.

Types of test automation tools:

- **Static automation tools:** These tools do not involve actual input and output. Rather, they take a symbolic approach to testing, i.e. they do not test the actual execution of the software. e.g. Flow analyzers, Coverage analyzers, Interface analyzer
- **Dynamic automation tools:** These tools test the software system with live data. e.g. Test driver, Test beds**,** Emulators

(e) Give the defect classification and its meaning.
(Classification - 4 marks)

Ans:

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

(f) Explain unit testing. State its additional requirements.
(Unit testing - 2 marks; requirements - 2 marks)

Ans:

Unit Testing: Software product is made up of many units, each unit needed to be tested to find whether they have implemented the design correctly or not.

Additional Requirements: The module under consideration might be getting inputs from another module or the module is calling some another module. Some interface modules has to be simulated if required like drivers and stubs.

Drivers: The module where the required inputs for the module under test are simulated for the purpose of module or unit testing is known as a Driver module. The driver module may print or interpret the results produced by the module under test.
Stubs: The module under testing may also call some other module which is not ready at the time of testing. There is need of dummy modules required to simulate for testing, instead of actual modules. These are called stubs.

(g) Explain the performance testing and its criterias.
(Performance testing - 2 marks; criteria - 2 marks)
Ans:
Performance testing: Performance testing is intended to find whether the system meets its performance requirements under normal load or abnormal level of activities. Normal load must be defined by the requirement statement defined by the customer and system design implements them. Performance criteria must be expressed in numerical terms. Design verification can help in determining whether required measures have been taken to meet performance requirements or not. This is one area where verification does not work to that many extents and one needs to test it by actually performing the operation on the system. It can serve different purposes like it can demonstrate that the system meets performance criteria.
Criteria of Performance testing:
Stress Testing:
In stress testing the resources are used less than the requirement. If the system has limited resources available, the response of the system may deteriorate due to non-availability of the resources. It tries to break the system 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 system 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. 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 users hits it simultaneously. Load 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.

(h) What is test planning and test management?
(Test planning - 2 marks; Test Management - 2 marks)

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.

2. Attempt any FOUR of the following : 4×4=16

(a) Explain the alpha testing. State its limitations.
(Alpha testing - 2 marks; limitation - 2 marks)

Ans:

Alpha testing is done by the customers in development environment in front of the development team.

Limitations:

1. Data provided by the customer may not represent actual data or business data. If data is created by testers using any data definition technique, probability of such data occurring in real life must be checked and validated by the customer.
2. Laboratory environment may not represent real life environment.
3. Key users deployed by customer may not be the people who are going to use the system in reality and also may not be aware of actual working expectations from a new system.

(b) State the Entry and Exit criterias for the software testing.
(Entry criteria - 2 marks; exit criteria - 2 marks)

Ans.

When to Start and Stop Testing of Software (Entry and Exit Criteria)
Process model is a way to represent any given phase of software development that prevent and minimize the delay between defect injection and defect detection/correction.

Entry criteria, specifies when that phase can be started also included the inputs for the phase.

- Tasks or steps that need to be carried out in that phase, along with measurements that characterize the tasks.
- Verification**,** which specifies methods of checking that tasks have been carried out correctly.
- Clear entry criteria make sure that a given phase does not start prematurely.
- The verification for each phase helps to prevent defects. At least defects can be minimized.

Exit criteria, which stipulate the conditions under which one can consider the phases as done and included are the outputs for the phase.

Exit criteria may include:

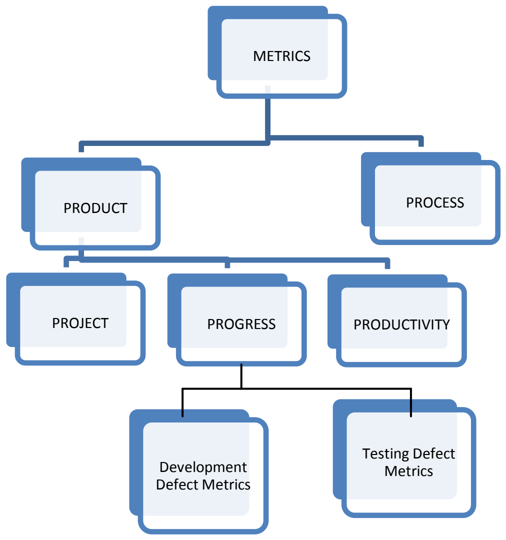
- All test plans have been run
- All requirements coverage has been achieved.
- All severe bugs are resolved.

(c) Explain the Boundary Value Analysis technique used in black box testing with example.
(BVA explanation - 2 marks; example - 2 marks)

Ans:
Most of the defects in software products have around conditions and boundaries. 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. Boundary value analysis help identify the test cases that are most likely to uncover defects.

Same examples of Boundary value analysis concept are: If the can accept he numbers between 1 to 100.Then
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.
Under this technique, boundary values 0,1, 2, 99,100,101can be tested.
Out of which 1,2,99,100 are valid values and 0,101 are invalid values.

(d) State the different metrics types with its classification.
(Stating metrics - 1 mark; Classification with explanation - 3 marks)
Ans:



Metrics are basically classified as:
1. Product Metrics: Product metrics are measures of software product at any stage of its development, from requirements to installed system.
2. Process Metrics: Process metrics are measures of the software development process such as the overall development time, type of methodology used or the average level of experience of the programming staff.

Product Metrics is classified as

- **Project Metrics:** A set of metrics that indicates how the project is planned and executed.
- **Progress:** A set of metrics that tracks how the different activities of the project are progressing.
  Progress Metrics is classified as 1. Test defect metrics 2. Development defect metrics
  1. Test defect metrics: help the testing team in analysis of product quality and testing
  2. Development defect metrics: help the development team in analysis of development activities.
- **Productivity:** A set of metrics that takes into account various productivity numbers that can be collected and used for planning and tracking testing activities.

OR

Other type of classification is:
1. Product vs. Process Metrics
2. Objective vs. Subjective Metrics
3. Primitive vs. Computed Metrics
4. Private vs. Public Metrics

(e) State any four testing principles.
(Any 4 principles - 1 mark each)
[Note - any other relevant principles shall also be considered**]**

Ans:

Testing Principles:
1. **Testing shows presence of defects:** Testing can show the defects are present, but cannot prove that there are no defects. Even after testing the application or product thoroughly we cannot say that the product is 100% defect free.
2. **Exhaustive testing is impossible:** Testing everything including all combinations of inputs and preconditions is not possible. So, instead of doing the exhaustive testing we can use risks and priorities to focus testing efforts.
3. **Early testing:** In the software development life cycle testing activities should start as early as possible and should be focused on defined objectives.
4. **Defect clustering:** A small number of modules contains most of the defects discovered during pre-release testing or shows the most operational failures.
5. **Pesticide paradox:** If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs. To overcome this “Pesticide Paradox”, it is really very important to review the test cases regularly and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects.

6. **Testing is basically context dependent**: Different kinds of sites are tested differently. For example, safety – critical software is tested differently from an e-commerce site.
7. **Absence – of – errors fallacy:** If the system built is unusable and does not fulfill the user's needs and expectations then finding and fixing defects does not help.

(f) Which types of test are first candidates for test automation? Why?
(Stating candidates - 1 mark; explanation - 3 marks)

Ans.

Stress, reliability, scalability and performance testing: These types of testing require the test case to be run from a large number of different machines for an extended period of time, such as 24 hours, 48 hours, and so on. It is just not possible to have hundreds of users trying out the product they may be not willing to perform the repetitive tasks, nor will it be possible to find that many people with the required skill sets. Test cases belonging to these testing types become the first candidates for automation.

Regression tests: Regression tests are repetitive in nature .These test cases are executed multiple times during the product development phase. Given the repetitive nature of test cases, automation will save significant time and effort in the long run. The time thus gained can be effectively utilized for other tests.

Functional tests: These kinds of tests may require a complex set up and thus require specialized skill, which may not be available on an ongoing basis. Automating these once, using the expert skill sets, can enable using less-skilled people to run these test on an ongoing basis.

3. Attempt any FOUR of following: $4\times 4 = 16$

(a) Explain what the static white box and black box testing is required.
(Need of static black box testing - 2 marks; need of static black box testing - 2 marks)

Ans:

Static black box testing includes testing the specifications.

Testing the specification is static black-box testing. The specification is a document, not an executing program, so it's considered static. It's also something that was created using data from many sources usability studies, focus groups, marketing input, and so on. The tester doesn't necessarily need to know how or why that information was obtained or the details of the process used to obtain it, just that it's been boiled down into a product specification. Tester can then take that document, perform static black-box testing, and carefully examine it for bugs. The development team may emphasize diagrams over words or it may use a self-documenting computer language such as Ada. Whatever their choice, tester can still apply all the techniques. If the product does not have specifications as a tester, this is a difficult position. His goal is to find bugs early ideally finding them before the software is coded but if the product doesn't have a spec, this may seem impossible to do. Although the spec may not be written down, someone, or several people, know what they're trying to build. It may be the developer, a project manager, or a marketer. He should use them as the walking, talking, and product spec and apply the same techniques for evaluating this "mental" specification as though it was written on paper. He can even take this a step further by recording the information that is gathered and circulating it for review.

This includes two ways of performing the reviews of specifications such as:

1. Performing a High-Level Review of the Specification
2. Low-Level Specification Test Techniques

Static White-Box Testing: Examining the Design and Code

Static testing refers to testing something that isn't running, examining and reviewing it. White-box (or clear-box) testing implies having access to the code, being able to see it and review it.

Static white-box testing is the process of carefully and methodically reviewing the software design, architecture, or code for bugs without executing it. It's sometimes referred to as STRUCTURAL ANALYSIS.

The reason to perform static white-box testing is:

1. To find bugs early and to find bugs that would be difficult to uncover or isolate with dynamic black-box testing. Having a team of testers concentrate their efforts on the design of the software at this early stage of development is highly cost effective.
2. A side benefit of performing static white-box testing is that it gives the team's black-box testers ideas for test cases to apply when they receive the software for testing. They may not necessarily understand the details of the code, but by listening to the review comments they can identify feature areas that sound troublesome or bug-prone.

(b) Explain the risk management in the test planning.

(Description of risk management in the test planning - 4 marks)

Ans:

Risk management in test planning:

A common and very useful part of test planning is to identify potential problem or risky areas of the project—ones that could have an impact on the test effort.

Suppose that you and 10 other new testers, whose total software test experience was reading this book, were assigned to test the software for a new nuclear power plant. That would be a risk. Maybe no one realizes that some new software has to be tested against 1,500 modems and there's no time in the project schedule for it is the another risk.

As a software tester, he will be responsible for identifying risks during the planning process and communicating your concerns to your manager and the project manager. These risks will be identified in the software test plan and accounted for in the schedule. Some will come true, others will turn out to be benign. The important thing is to identify them early so that they don't appear as a surprise late in the project.

These risks should be identified well ahead of time and the effective management is necessary for them. Proactive and reactive risks should be studied properly and should be evaluated to make them and their impact low.

(c) Differentiate between verification and validation.
(Any four points of differences between verification and validation - 4 marks; 1 mark each.)
Ans:

Verification
1. It makes sure that the product is designed to deliver all functionality to the customer.
2. Verification is done at the starting of the development process. It includes reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.
3. It answers the questions like: Am I building the product right?
4. Am I accessing the data right (in the right place; in the right way).
5. It is a Low level activity
6. Performed during development on key art facts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards.
7. Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

Validation

1. Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs.
2. Validation is done at the end of the development process and takes place after verifications are completed.
3. It answers the question like: Am I building the right product? Am I accessing the right data (in terms of the data required to satisfy the requirement).
4. It is a High level activity.
5. Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment.
6. Determination of correctness of the final software product by a development project with respect to the user needs and requirements.

(d) State the contents of 'Test Summary Report' used in test reporting.
(Test summary report contents - 4 marks)

Ans:

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
   - A summary of the activities carried out during that cycle.
   - Defects that are uncovered during that cycle based on severity and impact
   - Progress from the previous cycle to the current cycle in terms of defect fixed
   - Outstanding defects that not yet to be fixed in cycle
   - 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:
- Phase wise test summary, which is produced at the end of every phase
- Final test summary report.

A Summary report should present
- Test Summary report Identifier
- Description

Identify the test items being reported in this report with test id

1). Variances

Mention any deviation from test plans, test procedures, if any.

2). Summary of results

All the results are mentioned here with the resolved incidents and their solutions.

3). Comprehensive assessment and recommendation for release should include Fit for release assessment and recommendation of release

(e) Elaborate the advantages (any four) of using the test automation tools.
(Any four appropriate advantages of the test automation tools - 4 marks; 1 mark each)

Ans:

Advantages of using the test automation tools are as given below:

1. **Speed**. The automation tools tests the software under tests with the very faster speed. There's a vast difference between the speed of user entering the data and the automated tools generating and entering the data required for the testing of the software. Speed of these software also completes the work faster.
2. **Efficiency**. While testers are busy running test cases, testers can't be doing anything else. If the tester have a test tool that reduces the time it takes for him to run his tests, he has more time for test planning and thinking up new tests.
3. **Accuracy and Precision**. After trying a few hundred cases, tester's attention span will wane and he may start to make mistakes. A test tool will perform the same test and check the results 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 be used to simulate the real world and greatly reduce the physical resources necessary to perform the testing.
5. **Simulation and Emulation**. Test tools are often used to replace hardware or software that would normally interface to your product. This "fake" 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 tools and automation never tire or give up. they can keep going and going and on and on without any problem; whereas the tester gets tired to test again and again.

(f) Explain defect management process.

(Defect management process: description - 4 marks)

Ans:

Defect management process diagram:

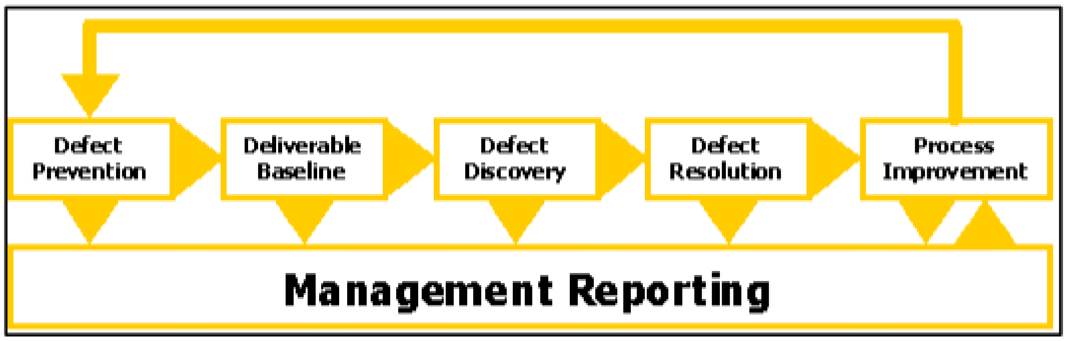


Figure 1: Defect management Process

As shown in the above diagram the defect management process is divided into following tasks:

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(s) 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.

4. Attempt any TWO of following: 2 x 8=16

(a) With the suitable example, explain how ‘Basis Path Testing’ is used to derive the code complexity for the testing.
(Basis path testing -2 marks; description of how it is used to derive the code complexity for the testing - 6 marks)
[Note - any other relevant answer shall also be considered**]**

Ans:

Basic 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 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. }
```

(b) Explain the Integration Testing and its two types. In each type, explain with example the steps of integration.

(Integration testing - 2 marks; any of the Two types; explanation with example of the steps of integration - 3 marks each - 6 marks)

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.

Disadvantages:

- ➢ Driver Modules must be produced.
- ➢ The program as an entity does not exist until the last module is added.

3). **Bi-Directional Integration.**

- 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 analyzed. 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.

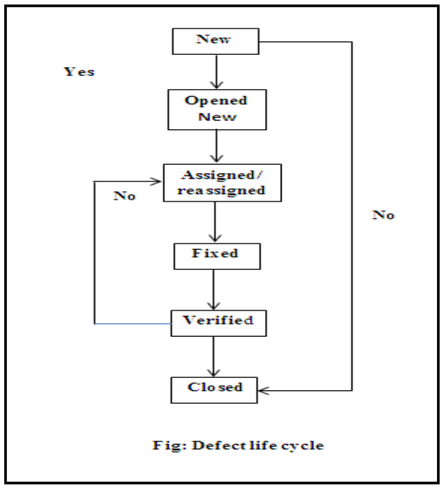
5). Non- Incremental Integration.

- 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.

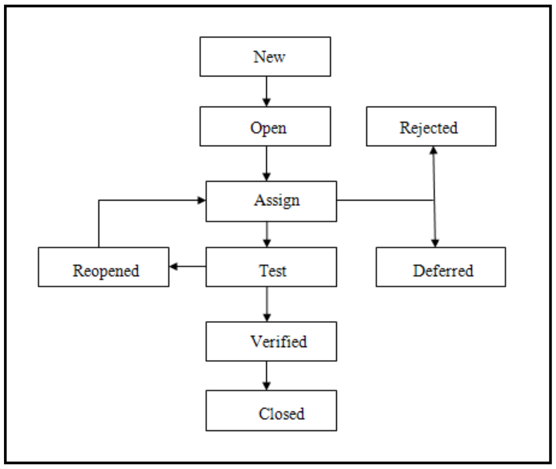
(c) Explain the defect tracking with defect life cycle diagram and the different defect states.
(Defect tracking with defect life cycle diagram - 4 marks; different defect states and their meaning - 4 marks)

Ans:

Defect's (bug's) life cycle is as shown in the diagram below:



OR



- **Different defect states and their meanings are as shown below:**

Status	Alternative Status
NEW	
ASSIGNED	OPEN
DEFERRED	
DROPPED	REJECTED
COMPLETED	FIXED, RESOLVED, TEST
REASSIGNED	REOPENED
CLOSED	VERIFIED

The defect states are as explained below:

- **NEW:** Tester finds a defect and posts it with the status NEW. This defect is yet to be studied/approved. The fate of a NEW defect is one of ASSIGNED, DROPPED and DEFERRED.

- **ASSIGNED / OPEN:** Test / Development / Project lead studies the NEW defect and if it is found to be valid it is assigned to a member of the Development Team. The assigned Developer's responsibility is now to fix the defect and have it COMPLETED. Sometimes, ASSIGNED and OPEN can be different statuses. In that case, a defect can be open yet unassigned.
- **DEFERRED:** If a valid NEW or ASSIGNED defect is decided to be fixed in upcoming releases instead of the current release it is DEFERRED. This defect is ASSIGNED when the time comes.
- **DROPPED / REJECTED:** Test / Development/ Project lead studies the NEWdefect and if it is found to be invalid, it is DROPPED / REJECTED. Note that the specific reason for this action needs to be given.
- **COMPLETED / FIXED / RESOLVED / TEST:** Developer 'fixes' the defect that is ASSIGNED to him or her. Now, the 'fixed' defect needs to be verified by the Test Team and the Development Team 'assigns' the defect back to the Test Team. A COMPLETED defect is either CLOSED, if fine, or REASSIGNED, if still not fine.
- If a Developer cannot fix a defect, some organizations may offer the following statuses:
- **Won't Fix / Can't Fix:** The Developer will not or cannot fix the defect due to some reason.
- **Can't Reproduce:** The Developer is unable to reproduce the defect.
- **Need More Information:** The Developer needs more information on the defect from the Tester.
- **REASSIGNED / REOPENED:** If the Tester finds that the 'fixed' defect is in fact not fixed or only partially fixed, it is reassigned to the Developer who 'fixed' it.
  A REASSIGNED defect needs to be COMPLETED again.
- **CLOSED / VERIFIED:** If the Tester / Test Lead finds that the defect is indeed fixed and is no more of any concern, it is CLOSED / VERIFIED.

5. (A) Attempt any TWO of the following: 2 x 6 =12

(a) Explain the terms Mistake, Error, Defect, Bug, Fault and Failure in relation with software testing.

(Description of mistake; error; defect; bug; fault and failure - 6 marks)

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.**
Failure: the inability of a system or component to perform its required functions within specified performance requirements.

- **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 behaviour 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);
- }

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

(b) What do you mean ‘Defect Impact’? Explain how to estimate the defect impact.

(Defect impact definition - 2 marks; how to estimate the defect impact - 4 marks)

Ans:

Defect Impact is a classification of software defect (bug) to indicate the degree of negative impact on the quality of software.

Or

- **Defect Impact:** The degree of severity that a defect has on the development or operation of a component or system.

How to Estimate the defect impact
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.

One should assume that an individual critical risk has a low probability of becoming a problem only when there is specific knowledge justifying why it is low. For example, the likelihood that an important requirement was missed may be high if developers have not involved users in the project. If users have actively participated in the requirements definition, and the new system is not a radical departure from an existing system or process, the likelihood may be low.

For 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.

(c) What is test deliverables and milestones? Explain any four test deliverables.

(Test deliverables and milestones - 2 marks; List of test deliverables and explanation - 4 marks)

Ans:

Test Deliverables are the artifacts which are given to the stakeholders of software project during the software development lifecycle. There are different test deliverables at every phase of the software development lifecycle. Some test deliverables are provided before testing phase, some are provided during the testing phase and some after the testing cycles is over.

The different types of Test deliverables are:

Test cases Documents
Test Plan
Testing Strategy
Test Scripts
Test Data
Test Traceability Matrix
Test Results/reports
Test summary report
Install/config guides
Defect Reports
Release notes

1. The test plan describes the overall method to be used to verify that the software meets the product specification and the customer's needs. It includes the quality objectives, resource needs, schedules, assignments, methods, and so forth.

2. Test cases list the specific items that will be tested and describe the detailed steps that will be followed to verify the software.
3. Bug reports describe the problems found as the test cases are followed. These could be done on paper but are often tracked in a database.
4. Test tools and automation are listed and described which are used to test the software. If the team is using automated methods to test software, the tools used, either purchased or written in-house, must be documented.
5. Metrics, statistics, and summaries convey the progress being made as the test work progresses. They take the form of graphs, charts, and written reports.
   Milestones: milestones are the dates of completion given for various tasks to be performed in testing. These are thoroughly tracked by the test manager and are kept in the documents such as Gantt charts, etc.

(B) Attempt any ONE of following: **1 x 4 =4**

(a) Explain the regression testing. State when the regression testing shall be done.

(Regression Testing - 2 marks; 2 Reason why regression testing is done - 2 marks)

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.

(b) Write important six test cases for the 'Login Form' of the Facebook website.

(Any valid 4 test cases for Login form of Facebook website - 4 marks; 1 mark each)

[Note - any other relevant test cases shall also be considered**]**

Ans:

Test cases for facebook login are as given below:

Step	Test step	Test data	Expected output	Actual output	Status
1	Navigate to login page of facebook	---	---	---	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

6. Attempt any Four of following : 4 x 4 =16

(a) Explain the client server application testing.

(Client server application testing description - 4 marks)

Ans:

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

- Multiple users can access the system at a time and they can communicate with
- the server.
- Configuration of client is known to the server with certainty.
- Client and server are connected by real connection.
- 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, where as 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.

(b) State the different errors being found by the static and dynamic black box testing.

(errors found by static black box testing - 2 marks and errors found by dynamic black box testing - 2 marks)
[Note - any other relevant answer shall also be considered**]**

Ans:

Errors found by static black box testing :
Testing the specification is static black-box testing. The specification is a document, not an executing program, so it's considered static. It's also something that was created using data from many sources usability studies, focus groups, marketing input, and so on. Teh tester doesn't necessarily need to know how or why that information was obtained or the details of the process used to obtain it, just that it's been boiled down into a product specification. Tester can then take that document, perform static black-box testing, and carefully examine it for bugs.
The development team may emphasize diagrams over words or it may use a self-documenting computer language. It performs this with the help of :

1. Performing a High-Level Review of the Specification
2. Low-Level Specification Test Techniques

Overall all the errors in the specifications are found by this testing technic. Word to word the specifications are scanned and are understood to get the clear meaining of them.
Errors found by dynamic black box testing:
<u>Dynamic Black-Box Testing is as good as Testing the Software While Blindfolded</u>.
Testing software without having an insight into the details of underlying code is dynamic black-box testing. It's dynamic because the program is running you're using it as a customer would. And, it's black-box because you're testing it without knowing exactly how it works with blinders on.
The testers keep on entering inputs, receiving outputs, and checking the results. Another name commonly used for dynamic black-box testing is behavioral testing because its testing how the software actually behaves when it's used.

- To do this effectively requires some definition of what the software does namely, a requirements document or product specification. Tester doesn't need to be told what happens inside the software "box" , tester just need to know that inputting A outputs B or that performing operation C results in D. A good product spec will provide tester with these details.
- Once tester know the ins and outs of the software tester is about to test, testers next step is to start defining the test cases. Test cases are the specific inputs that he will try and the procedures that he will follow when he tests the software.
- Partitioning of the data effectively is done for the software and checked.
- Data testing is done with respect to boundry conditions , boundry value analysis, sub boundry conditions.

- State testing is done by giving the inputs and gaining the outputs.
- It tests the software's logic flow. Generates the state transtition maps.
- Software is tested for load and stress conditions.

(c) State the contets of standard template of a test plan.
(template contents of a test plan - 4 marks)
[Note - any other relevant template shall also be considered**]**

Ans:

TEST PLAN TEMPLATE

1. Introduction
   1.1 Scope

What features are to be tested and what features will not be tested what combinations of environment are to be tested and what not.

2. References
3. Test Methodology and Strategy/Approach
4. Test Criteria
   4.1 Entry Criteria
   4.2 Exit Criteria
   4.3 Suspension Criteria
   4.4 Resumption Criteria

5. Assumptions, Dependencies, and Risks
   5.1 Assumption
   5.2 Dependencies
   5.3 Risk and Risk Management Plans
6. Estimations
   6.1 Size Estimate
   6.2 Effort Estimate
   6.3 Schedule Estimate
7. Test Deliverables and Milestones
8. Responsibilities
9. Resource Requirement
   9.1 Hardware Resources
   9.2 Software Resources
   9.3 People Resources
   9.4 Other Resources
10. Training Requirements
   10.1 Detail of Training Required

10.2 Possible Attendees
10.3 Any Constrains
11. Defect Logging and Tracking Process
12. Metrics Plan
13. Product Release Criteria

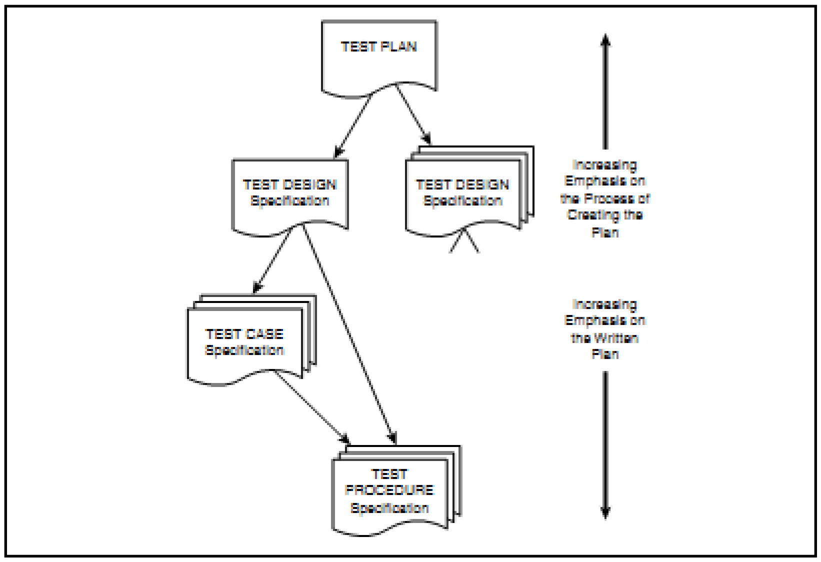
(d) Explain the 'Test infrastructure' components with neat diagram.

(test infrastructure diagram - 2 marks ; description - 2 marks)

[Note - any other relevant answer shall also be considered**]**

Ans :

Test infrasturcture componets diagram :


TEST PLAN
TEST DESIGN Specification
TEST DESIGN Specification
TEST CASE Specification
TEST PROCEDURE Specification
Increasing Emphasis on the Process of Creating the Plan
Increasing Emphasis on the Written Plan

The top, or project level, test plan, the process of creating it is more important than the resulting document. The next three levels, the test design specification, the test case specification, and the test procedure specification are described in detail in the following sections.
As you can see in Figure, moving further away from the top-level test plan puts less emphasis on the process of creation and more on the resulting written document. The reason is that these plans become useful on a daily, sometimes hourly, basis by the testers performing
the testing. At the lowest level they become step-by-step instructions for executing a test, making it key that they're clear, concise, and organized how they got that way isn't nearly as important.
This standard is what many testing teams have adopted as their test planning documentation intentional or not—because it represents a logical and common-sense method for test planning.
The important thing to realize about this standard is that unless tester is bound to follow it to the

letter because of the type of software he is testing or by your corporate or industry policy, tester should use it as a guideline and not a standard.

Test Design

The overall project test plan is written at a very high level. It breaks out the software into specific features and testable items and assigns them to individual testers, but it doesn't specify exactly how those features will be tested. There may be a general mention of using automation or black-box or white-box testing, but the test plan doesn't get into the details of exactly where and how they will be used. This next level of detail that defines the testing approach for individual software features is the test design specification.

Test Cases

Dissecting a specification, code, and software to derive the minimal amount of test cases that would effectively test the software. The test case specification "documents the actual values used for input along with the anticipated outputs. A test case also identifies any constraints on the test procedure resulting from use of that specific test case." Essentially, the details of a test case should explain exactly what values or conditions will be sent to the software and what result is expected. It can be referenced by one or more test design specs and may reference more than one test procedure. The ANSI/IEEE 829 standard also lists some other important information that should be included:

- **Identifiers.**
- **Test item**.
- **Input specification.**
- **Output specification.**
- **Environmental needs.**
- **Special procedural requirements.**
- **Intercase dependencies**.

Test Procedures

After tester documents the test designs and test cases, what remains are the procedures that need to be followed to execute the test cases. The test procedure specification "identifies all the steps required to operate the system and exercise the specified test cases in order to implement the associated test design."

The test procedure or test script spec defines the step-by-step details of exactly how to perform the test cases. Here's the information that needs to be defined:

- **Identifier.** A unique identifier that ties the test procedure to the associated test cases and test design.
- **Purpose.** The purpose of the procedure and reference to the test cases that it will exe-cute.
- **Special requirements**. Other procedures, special testing skills, or special equipment needed to run the procedure.
- **Procedure steps.** Detailed description of how the tests are to be run:
- **Log.** Tells how and by what method the results and observations will be recorded.
- **Setup.** Explains how to prepare for the test.
- **Start.** Explains the steps used to start the test.

- **Procedure.** Describes the steps used to run the tests.
- **Measure.** Describes how the results are to be determined for example, with a stopwatch or visual determination.
- **Shut down.** Explains the steps for suspending the test for unexpected reasons.
- **Restart.** Tells the tester how to pick up the test at a certain point if there's a failure or after shutting down.
- **Stop.** Describes the steps for an orderly halt to the test.
- **Wrap up.** Explains how to restore the environment to its pre-test condition.
- **Contingencies.** Explains what to do if things don't go as planned.

(e) Explain with sample example the use of Decision Tables in the black box testing.
(Decision table ; use - 2 marks ; sample example - 2 marks)

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

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**:
Pin Number	T	T	T	F
Payment Detail	T	F	F	T
Overdue details	F	T	T	F

(f) The student passing the diploma shall be awarded class of passing as – Distinction, Fisrt Class, Second Class, Pass Class. The class will be applicable only if the student has his/her result 'pass' in all of subjects.
Write the test cases (minimum-4) for the class awarding algorithm (module).

(Any four valid test cases - 4 marks - 1 mark each)

[Note - any other relevant test cases shall also be considered**]**

Ans:

Step	**Test step**	**Test data**	**Expected output**	**Actual output**	**Status**
1	Enter the marks in the range of 40-49%	Any value between 40 to 49	Should display class as Pass class	Displayed the class as Pass class	Pass
2	Enter the marks in the range of 50-59%	Any value between 50-59	Should display as Second class	Displayed Second class	Pass
3	Enter marks in the range of 60-74%	Any value between 60-74	Should display as first class	Display as First class	Pass
4	Enter the marks in the range of 75 to 100	Any value between 75-100	Should display as Distinction	Displayed the result as Distinction	Pass