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.

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

i) What is Bug? Why do bug occur?

(What is bug-1 Mark, rules- 3Marks)

All software problems will be called bugs. Problems, error, bug are probably the most generic term used.

Bug occurs when one or more of the following five rules is true:

1. The software doesn't do something that the product specification says it should do.
2. The software does something that the product specification says it shouldn't do.
3. The software does something that the product specification doesn't mention.
4. The software doesn't do something that the product specification doesn't mention but should.
5. The software is difficult to understand, hard to use, slow, or in the software tester's eyes will be viewed by the end user as just plain not right.

ii) How to calculate cost of Bug?

(Explanation- 2 Marks, Diagram/example- 2 Marks)

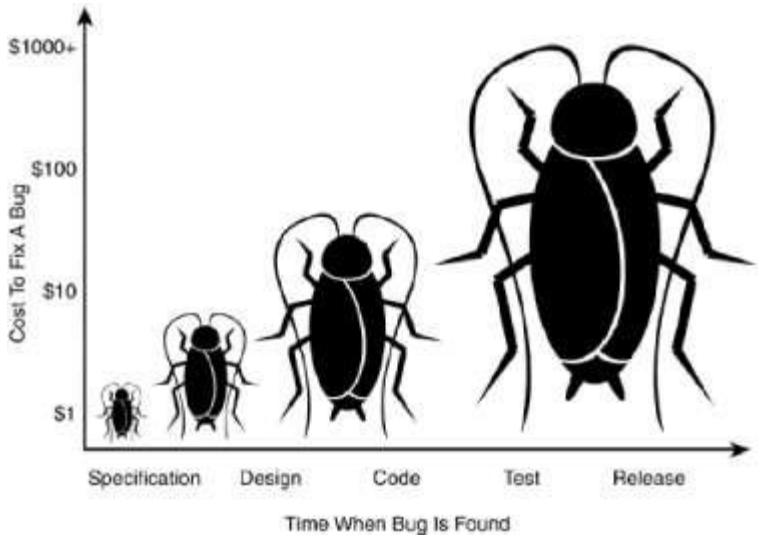
[Note:- diagram should be considered for any other representation which shows growth in cost]

The costs are logarithmic; they increase tenfold as times increases. A bug found & fixed during the early stages that are requirement or product specification stage can be found & fixed by a brief interaction with the concerned & might cost next to nothing.

During coding, a swiftly spotted mistake may take only very less efforts to fix. When the specification is being written might cost next to nothing or 10 cents in our examples. The same

bug, if not found until software is coded & tested, might cost $1 to $10. If customer finds it, the cost could easily top $100.

The cost of fixing these bugs can grow over time is as shown in diagram.



iii) Give the difference between Black Box Testing and White Box Testing.(any 4)

(Any four differences, Each Difference -1 mark)

Black Box Testing	White Box Testing
1)Tester does not have access to the program code	1)Tester has access to the program code
2)It is testing against functional specification	2) Testing based on design & implementation structure.
3)Black box is also called as - Behavioral testing -Functional Testing -opaque box testing -closed box testing	3) White box is also called as -Structural testing -clear-box testing -Glass box testing -open box testing

4) Black-box testing types:- Static black box testing Dynamic Black –box testing	4)White-box testing types: -Static White-box testing -Dynamic white Box testing
5)Mainly applicable to higher levels of testing: Acceptance System testing	5)Mainly application to lower levels of testing. Unit & Integration Testing
6) Output is depend on prediction	6) Output depends on execution of statement.

iv) Why Coding Standards and Guidelines required?

(Each reason-1 Mark, Example -1 Mark)

There are three reasons for adherence to standard or guidelines.

Reliability: it's been shown that code written to a specific standard or guidelines is more reliable & bug-free than code that isn't.

Readability/maintainability: code that follows set of standards and guidelines is easier to read, understand & maintain.

Portability: Code often has to run on different hardware or to be compiled with different compilers. If it follows a set standard, it will likely be easier or even completely painless to move it to different platforms.

Example: ANSI (American National Standards Institute)

IEC (International Engineering Consortium)

NCITS (National committee for Information Technology standards)

b) Attempt any ONE of the following: Marks 6

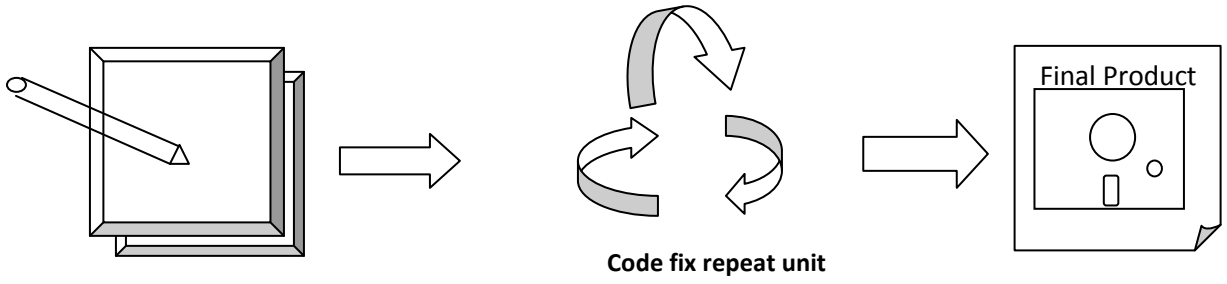
i) Explain code and fix model and waterfall model with diagram.

(Diagram of each model -1 Mark, Explanation for each model -2 Marks)

Code and Fix Model:

The code and Fix model is as shown in fig is usually the one that project teams fall in to by default if they don't consciously attempt to use something else. It's a step up, procedurally from Big Bang model.

In this model in that at least requires some idea of what product requirement are.



Typically informal product specification

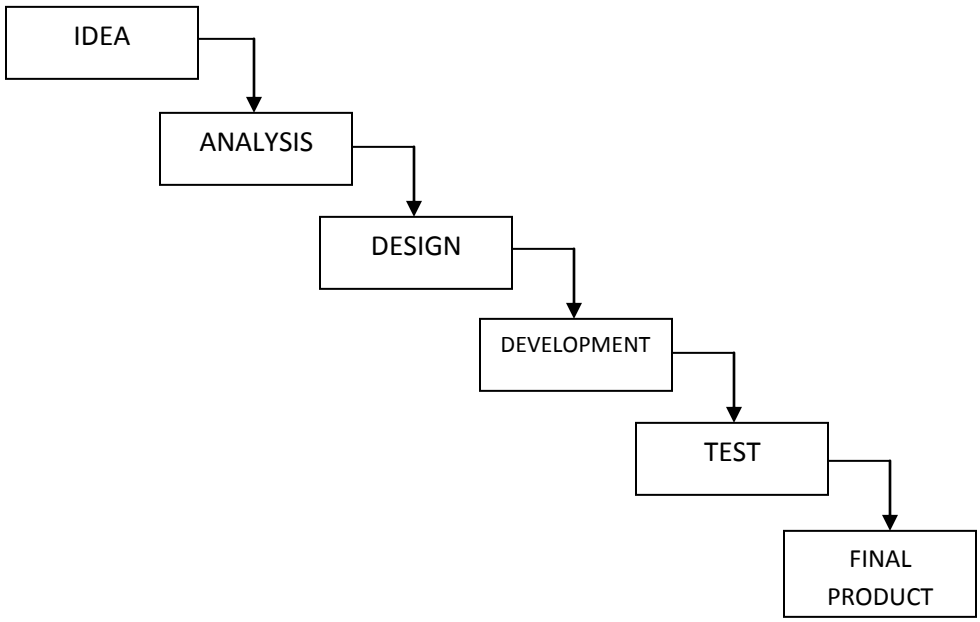
Code fix repeat unit

This model plays a significant role between coding & fixing. A team using approach usually starts with rough idea of what they exactly want, does some sample design & then produced into long repeating cycle of coding, testing & fixing bugs. Code & fix has been used on many large & well known software products.

Example: Code & Fix model is used for Word processor or spreadsheet software

This model play very significant role in coding & fixing.

Waterfall Model



This model is very simple & elegant. Fig. shows steps involved in this model.

A project using the waterfall model moves down a series of steps starting from an initial idea to final product. At the end of each step, the project team holds a review to determine if they are to move to the next step. It works well for projects with well-understood product definition.

Important thing about the waterfall method:

- There's large emphasis on specifying what product will be.
- The steps are discrete, there is no overlap.
- There is no way to go back up.

ii) Describe Interference injectors and noise generators.

(Diagram-2 Marks, Explanation - 4 Marks)

They are similar to stress & load tools but more random in what they do. The stress utility, for example has an executor mode that randomly changes the available resources. A program might run fine with lots of memory & might handle low memory situation ,but it could have problems if the amount of available memory is constantly changing. The executor mode of the stress utility would uncover these types of bugs.

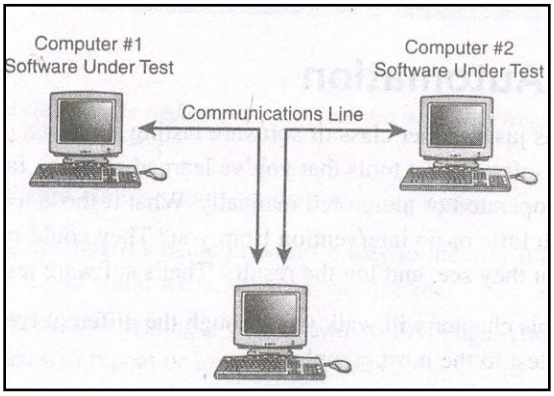


Fig. shows an interference injector & Noise generator tool. In this viewers are replaced with hardware & software that allows not only viewing data on the communication line but also modifying it. It becomes interference injector .Such setup could simulate all types of communication errors by data dropouts, noisy or bad cable & so on.

When deciding where & how to use interference injectors & noise generators then think about what external influences affect the software you are testing & then figure out ways to vary & manipulate those influences to see how software handles it.

2. Attempt any FOUR of the following: Marks 16

a) Explain static white box testing.

(Definition-2 Marks, Explanation-2 Marks)

Static testing refers to testing something, which is static i.e. not running.

White box testing implies having access to the code i.e. being able to see it & reviews 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.

Static white box testing does not involve executing the programs, but involves people going through the code to find out whether.

-The code works according to the functional requirement.

-Any functionality has been missed out in the code.

-Code handles errors properly.

- The code has been written in accordance with the design developed.

Static testing done by humans or with specialized tools. Static white-Box testing is performed under process of formal reviews.

b) What is software logic flow? Explain with state transition map.

(Explanation of software logic flow-1 Mark, STD diagram-1 Mark and explanation-2 Marks)

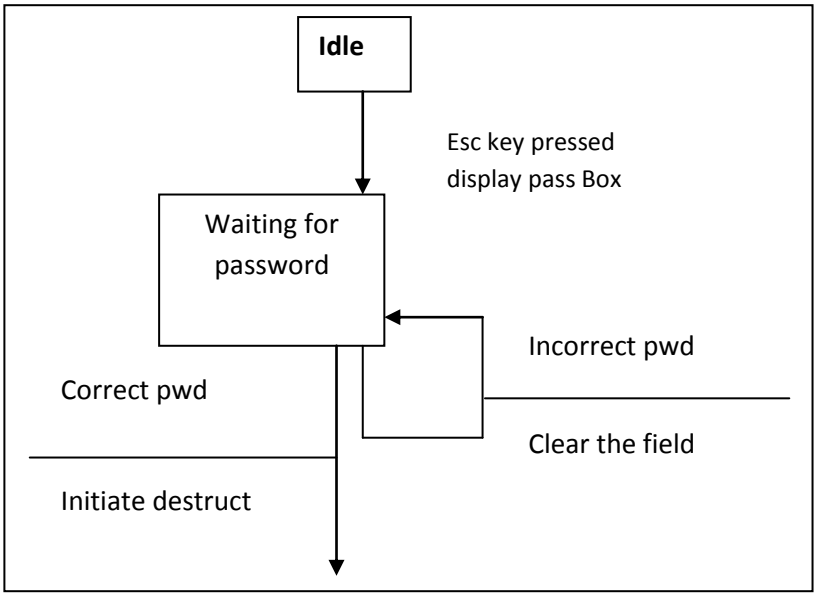
Program's states & the transition between them is called as software logic flow

The state testing is to verify program logic flow through its various states. A software state is condition or mode that software is currently in.

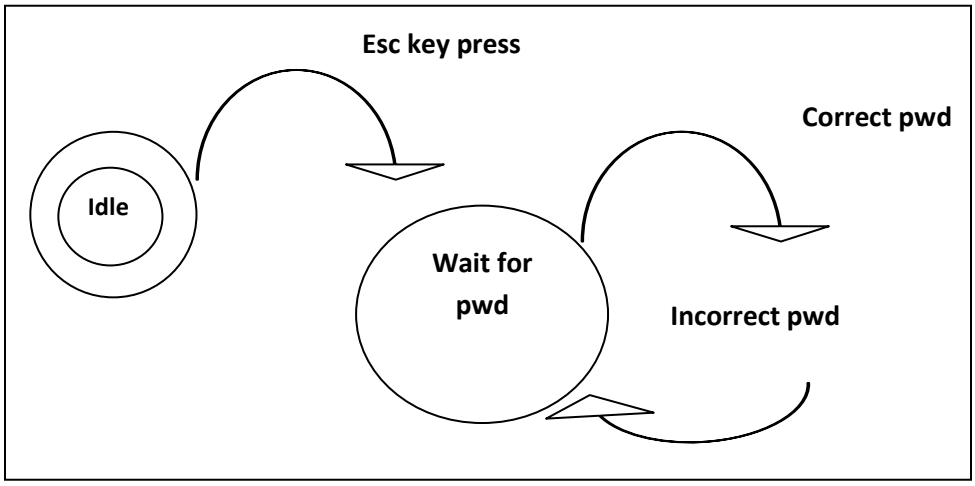
The state transition diagram/map (STD) indicates how system behaves consequence of external events to accomplish this; STD represents various modes of behavior.

The first step is to create your own state transition map of software product.

There are several different diagramming techniques for state transition diagram. Fig. shows two examples. One uses boxes & other uses circles (bubbles) & arrows. The techniques you use to draw your map aren't important as long as you & the other members of your project team can read & understand it.



OR



A state transition map should show following items.

-Each unique state that software can be the input or condition that takes it from one state to next. Set conditions & produced output when state is entered or exit.

c) What is accessibility testing? What is its need?

(Meaning- 1 Mark, Explanation and need -3 Marks)

A topic that falls under the area of usability testing is that of accessibility testing or testing for disabled.

There are many types of disabilities, the following ones make using computers and software especially difficult:

Visual impairments: Color blindness, extreme near and far sightedness, tunnel vision, dim vision, blurry vision, and cataracts are examples of visual limitations. People with one or more of these would have their own unique difficulty in using software. Think about trying to see where the mouse pointer is located or where text or small graphics appear onscreen. What if you couldn't see the screen at all?

Hearing impairments: Someone may be partially or completely deaf, have problems hearing certain frequencies, or picking a specific sound out of background noise. Such a person may not be able to hear the sounds or voices that accompany an onscreen video, audible help, or system alerts.

Motion impairments: Disease or injury can cause a person to lose fine, gross, or total motor control of his hands or arms. It may be difficult or impossible for some people to properly use a

keyboard or a mouse. For example, they may not be able to press more than one key at a time or may find it impossible to press a key only once. Accurately moving a mouse may not be possible.

Cognitive and language: Dyslexia and memory problems may make it difficult for someone to use complex user interfaces. Think of the issues outlined previously in this chapter and how they might impact a person with cognitive and language difficulties.

Due to all above reasons to access software with help of hardware's by disabled user accessibility testing is needed.

d) Describe Random Testing?

(Meaning-1 Mark and Explanation-3 Marks)

Monkey testing is Random testing performed by automated tools.

The test automation tools & techniques are designed to help you in running your test cases or ideally running your test cases automatically without the need for constant attention. Its goal is simulate what your users might do. That type of automation is called a test monkey.

The term test monkey comes from the idea that if you had a million monkeys typing on a million keyboards for a million years results into some great work.

There are "smart monkeys" & "dumb monkeys". :"smart monkeys are valuable for load & stress testing ,they will find a significant numbers of bugs, but they are also very expensive to develop." dumb monkeys" with its continuous repetition & use can expose bugs such as memory leaks.

When your software is released to the public, it will have thousands or possibly millions of people using it. Despite your best efforts at beginning test cases to find bugs, some bugs will slip by those users. What if you could supplement your test case approach with simulation of what all those users would do, before you released your product? You could potentially find bugs that would have otherwise made it past your testing. That's what a test monkey can do.

e) Explain Micro recording and playback.

(Meaning -1 Mark, Steps-3 Marks)

[Note: Diagram is optional]

Macro recorder & players are a type of driver tool. Drivers are tools used to control & operate the software being tested. With a macro program you are doing just that the macros you record are played back, repeating the action that you performed to test the software.

Figure shows a screen from the Macro Setup wizard, which walks you through the steps necessary to configure & capture your macros.

The Macro Setup Wizard allows you to set the following options for your micros:

a) Name. Giving the macro a name provides a way to identify it later. Even for a small software project you could hundreds of macros.

b) Repetitions. Repetition testing is great way to find bugs. You can set the number of times the micro will repeat or loop when it runs.

c) Triggers. You can set how the micro is started. This can be by hot key (for example, Ctrl+Shift+T), by a set of typed-in character, by clicking a shortcut, when a certain window is displayed or when the system has idled unused for a certain length of time.

d) What's captured? You can select to capture just key strokes or both keystrokes & mouse actions such as moving & clicking.

e) Playback speed. The micro can play back from 20 percent slower to 500 percent faster than how you originally recorded it. This is important if your software's performance can vary. What would happen if the software you are testing become a bit slower & the button the micro was to click on wasn't yet onscreen?

f) Playback Position. This option determines if the mouse movements & clicks should be absolute or relative to a certain window onscreen. If you are testing an application that might change onscreen positions, making your movements relative to that application is a good idea, otherwise the mouse may not click where you would expect.

f) Which are the localization issues for Foreign language Testing.

(Definition of localization-1 Mark, Major elements -3 Marks)

The process of adapting software to a specific locale taking into consideration its language, dialect, local conventions and culture is called localization.

Major elements for localization are:

1. Content
2. Data formats.

1. Content: If you are testing a product that will be localized, you need to carefully examine the content to make sure it's appropriate to the area where it will be used.

The following list shows various types of contents that you should carefully review for localization issues.

Sample document	Icons
Pictures	Sound
Video	Help Files
Map	Marketing materials
Packaging	web links

3. Data Formats: Different locals use different formats for data units such as currency, time & measurement.

Unit	Consideration
Currency	Different symbols and where they're placed
Dates	Order of month, day, year; separators; leading zeros; long and short formats
Times	12-hour or 24-hour, separators

3. Attempt any FOUR of the following: Marks 16

a) Explain data coverage in detail

(Each point -1 Mark)

When testing data coverage, code is divided into its data and states. The data includes all variables, constants, array, data structure, keyboard and mouse input, files and screen input and output, and I/O to other devices such as modems, networks and so on.

1. Data Flow:

   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 functions. If you test a function at this low level, you would use a debugger and watch variables to view the data as the program runs.

   With the black box testing, you only know what is the value of variable at the beginning and at the end.

2. Sub –Boundaries:

   Sub boundaries in data coverage means checking the input data boundaries.

   Boundary testing is where test cases are generated using the extremes of the input domain. If you perform white box testing, you need to examine the code carefully. To look for sub boundary conditions and create test cases that will exercise them.

3. Formula and Equations:

   Formula and equations are hidden deep in the code and their presents or effect is not always obvious from the outside. Checking the formula and equations by putting different inputs.

4. Error forcing:

   Last type of data testing is error forcing. Forced error testing is nothing but mutation testing. It is process of inducing error / changes to the application to find how application is working.

b) Describe bug tracking system?

(Explanation-3 Marks, Report-1Mark)

- A bug tracking system is a software application that is designed to help quality assurance and programmer to keep track of reported software bugs in their work.
- It is a methodology used by software developers to collect reports of defect or “bugs” in software programs.

- Bug tracking allows developer to further refine software design by making continual changes or upgrades to the product in order to better serve the customer base.

- Bug tracking system defers in features, essentially reported bugs are assigned unit tracking numbers.
- The bug tracking system prioritizes bugs according to various factors and categorizes each bug as being of normal, high or critical importance.
- Bug tracking system allows programmers and managers alike to see at a glance where software is failing its customer base, and how the development is handling those problems.
- It provides an organized way to facilitate software enhancement procedures and opens a proper, regulated channel to end user.
- This encourages bug tracking with minimal human resources invested in organizational requirements of the process.
- Effective bug tracking system can improve customer satisfaction, raise productivity and reduce downtime.
- There are manual and automated bug reporting and tracking.

  The automated tools show the individual bugs, their IDs, titles, status, priority, severity, and resolution.
- Bug report is used in bug tracking system to contain all details of bug.
- Bug report is useful to track a bug through its life cycle.
- Bug report form can look like as follows.

XYZ Software Ltd Bug Report Bug ID_____

SOFTWARE_____ RELEASE_____ ASSIGENED TO_____

Severity:1 2 3 4 Priority: 1 2 3 4 Reproducible: Yes/No

Title:_____

Description:_____

_____

_____

Resolution: Fixed/Duplicate/No-repro/Can't Fix/Deferred/Won't Fix

Date Resolved:_____ Resolved by:_____ Version:_____

Resolution Comment:_____

_____

_____

Retested by:_____ Version Tested_____ Date Tested_____

c) Explain Dynamic Black Box Testing

(Definition- 2 Marks, Explanation- 2 Marks)

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.

Another name commonly used for dynamic black-box testing is behavioral testing because you're testing how the software actually behaves when it's used.

You're entering inputs, receiving outputs, and checking the results.

To do this effectively requires some definition of what the software does namely, a requirements document or product specification.

You don't need to be told what happens inside the software "box" you just need to know that inputting A outputs B or that performing operation C results in D.

A good product spec will provide you with these details.

Once you know the ins and outs of the software you're about to test, your next step is to start defining the test cases. Test cases are the specific inputs that you'll try and the procedures that you'll follow when you test the software.

a) What are the Standards and Guidelines for Compatibility Testing?

(High level -2 Marks, Low level -2 Marks)

- Compatibility testing is an analysis conducted to validate the program's compatibility with the associated environment modules and other software.
- It is a non functional test which primarily focuses upon the application's suitable performance in presence of and in relation to other program.

There are really two levels of standards and guidelines:

- High level standards and guidelines
- low level standards and guidelines

High level standards and guidelines

- High level standards are the ones that you guide your product's general operations, its look and its feel, its supported features and so on will your software run under Windows, Mac, or Linux operating system? Is it web application?
- If so, what browser will it run on? Each of these is considered a platform and most have their own set of standards and guidelines that must be followed if an application is to claim that it's compatible with the platform.

Low level standards and guidelines

- Low level standards are most important as compare to high level standards and guidelines. It is gives the details of the file formats and network communication protocols.
- You Should Treat Low level compatibility standards as an extension of the software specifications. If the software specifications states The software will save and load picture files as .bmp, .gif and .jpg you need to find out standards and guidelines for these format and design test to conform that that software support all extensions picture files.

b) List and explain Web Page Fundamentals.

(Any 4 points-Each Point 1 Mark)

Web page fundamentals

1. Text
2. Hyperlinks
3. Graphics
4. Forms
5. Objects and Other Simple Miscellaneous Functionality

1. Text

When testing the text consider the audience level, the terminology, the content and subject matter, the accuracy and spelling. If web page contains contact information such as email address, phone number and address, must check them to make sure that they are correct. You should check the title of each page. It displayed correctly or not. You should also take care of text layout issues.

2. Hyperlinks

Testing of hyperlinks on web pages means click on each hyperlink and check to make sure that it jumps to correct destination and open the correct windows. Hyperlinks on the web pages usually are underlined text links and the mouse pointer should change to a hand pointer, when it is over any hyperlink.

3. Graphics

In graphics testing test all graphics load and display properly on web page. If the graphics is missing or is incorrectly named, it won't load and webpage display the error. Make sure that the text wrap properly around the graphics or not.

4. Forms

Forms are the text boxes, list boxes, and other fields for entering or selecting information on a Web page.

5. Objects and Other Simple Miscellaneous Functionality:

The Web site may contain features such as a hit counter, scrolling marquee text, changing advertisements or internal site searches.

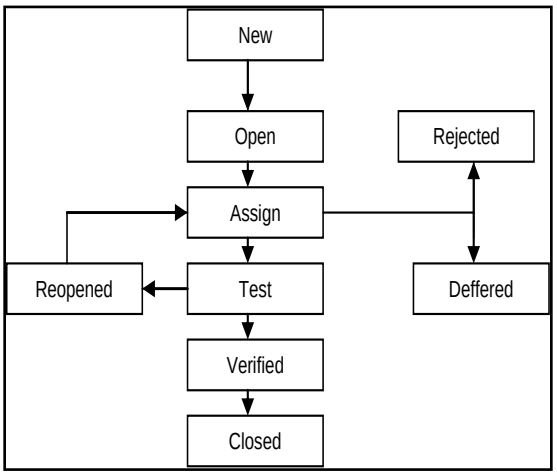
4. a) Attempt any THREE of the following : Marks 12

i) Explain the bug life cycle

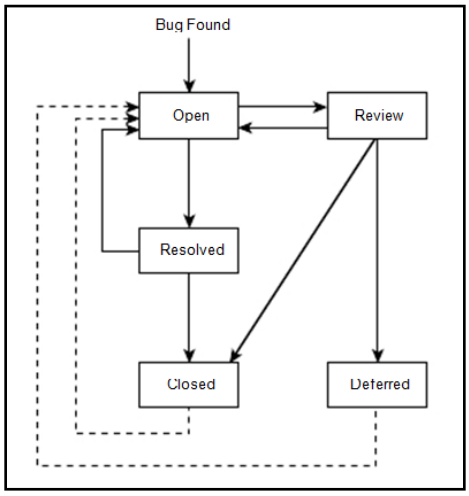
(Diagram 2 Marks, Explanation 2 Marks)

Software problems are called bugs, when tester founds the bug, that bug is reported or entered bug into database. In software development the bug has a life cycle.

Following figure shows the bug life cycle:



OR



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

1. New: When the bug is posted for the first time, its state will be "NEW". This means that the bug is not yet approved.

2. Open: After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as "OPEN".

3. Assign: Once the lead changes the state as "OPEN", he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to "ASSIGN".

4. Test: Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to "TEST". It specifies that the bug has been fixed and is released to testing team.

5. Deferred: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are

priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

6. Rejected: If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "REJECTED".

7. Verified: Once the bug is fixed and the status is changed to "TEST", the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "VERIFIED".

8. Reopened: If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "REOPENED". The bug traverses the life cycle once again.

9. Closed: Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "CLOSED". This state means that the bug is fixed, tested and approved.

ii) What is Unit and Integration Testing?

(Unit Testing- 2 Marks, Integration Testing-2 Marks)

Unit Testing

- Unit Testing is a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
- A unit is the smallest testable part of software. It usually has one or a few inputs and usually a single output. In procedural programming a unit may be an individual program, function, procedure, etc.
- Unit testing is the first level of testing and is performed prior to Integration Testing. Unit Testing is normally performed by software developers themselves or their peers.
- In rare cases it may also be performed by independent software testers.

Integration testing

- Integration Testing is a level of the software testing process where individual units are combined and tested as a group.

- The purpose of this level of testing is to expose faults in the interaction between integrated units.

- Test drivers and test stubs are used to assist in Integration Testing. Integration Testing is performed after Unit Testing and before System Testing. Either Developers themselves or independent Testers perform Integration Testing.
- There are mainly two approaches to do integration testing.
  i) Top-down Approach
  ii) Bottom-up Approach

iii) Describe Gray Box Testing.

(Definition - 1 Mark, Explanation with Example -3 Marks)

- Gray Box Testing is a software testing method which is a combination of Black Box Testing method and White Box Testing method.
- In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure in known.
- In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.

Web pages can be tested by a gray box testing. Most Web pages are built with HTML (Hypertext Markup Language). HTML isn't a programming language it's a markup language. In the early days of word processors, you couldn't just select text and make it **bold** or *italic*.

For example, to create the bolded phrase you would enter something such as this into your word processor:

[Begin bold]This is bold text.[end bold]

HTML works the same way. To create the line bold in HTML you would enter

This is bold text.

- In the Gray box testing tester is usually has knowledge of limited access of code and based on this knowledge the test cases are designed and the software application under test treat as a black box & tester test the application from outside.

iv) Discuss the Configuration and Compatibility issues for foreign language Testing

(Configuration-2Marks, Compatibility 2 Marks)

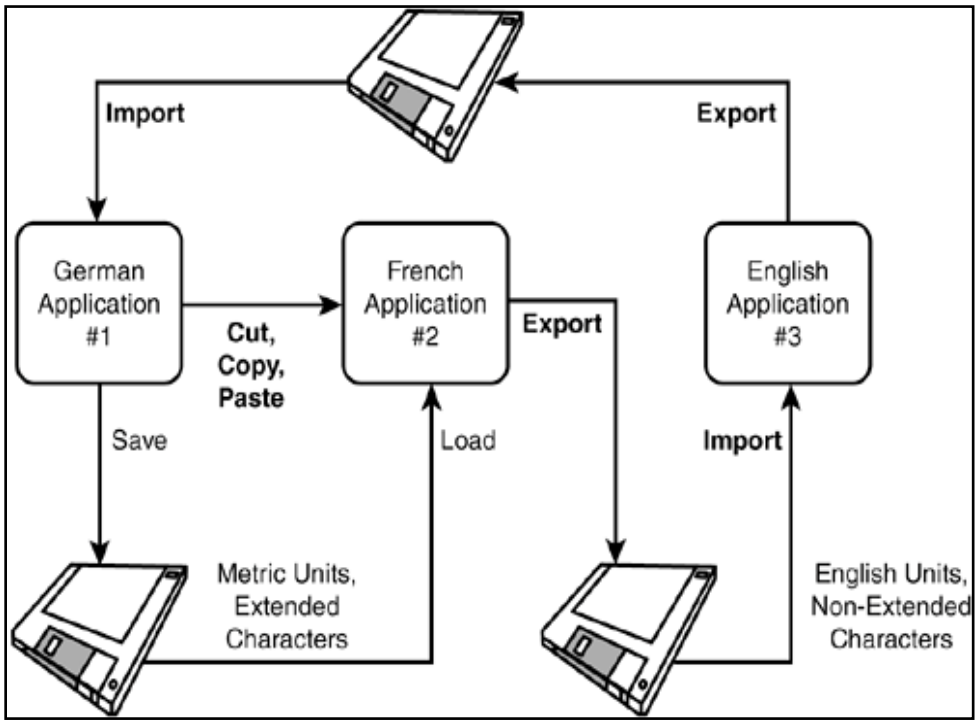
- Configuration and compatibility testing is very important when testing localized versions of software.
- The problems that can crop up when software interacts with different hardware and software are amplified by all the new and different combinations.

Foreign Platform Configurations

- Windows XP supports 106 different languages and 66 different keyboard layouts. The drop-down list for languages runs from Afrikaans to Ukrainian and includes eight different versions of English other than American English (Australian, British, Canadian, Caribbean, Irish, Jamaican, New Zealand, and South African), five different German dialects, and 20 different Spanish dialects.
- Keyboards are probably the piece of hardware with the largest language dependencies, but depending on what you're testing, there can be many others.
- Printers, for example, would need to print all the characters your software sends to them and properly format the output on the various paper sizes used in different countries.
- If your software uses a modem, there might be issues related to the phone lines or communication protocol differences. Basically, any peripheral that your software could potentially work with needs to be considered for a place in your equivalence partitions for platform configuration and compatibility testing.

Data Compatibility

- Just as with platform configuration testing, compatibility testing of data takes on a whole new meaning when you add localization to the equation.
- Figure shows how complex it can get moving data from one application to another. In this example, a German application that uses metric units and extended characters can move data to a different French program by saving and loading to disk or using cut and paste.
- That French application can then export the data for import to yet another English application. That English program, which uses English units and non-extended characters, can then move it all back to original German program.



- During this round and round of data transfers, with all the conversions and handling of measurement units and extended characters, there are numerous places for bugs. Some of these bugs might be due to design decisions.
- For example, what should happen to data moved from one application to another if it needs to change formats? Should it be automatically converted, or should the user be prompted for a decision? Should it show an error or should the data just move and the units change?

b) Attempt any ONE of the following: Marks 6

i) List different testing axioms? Explain any two in detail.

(List -2 Marks, Explanation of any two -2 marks each)

1. It's Impossible to Test a Program Completely
2. Software Testing Is a Risk-Based Exercise
3. Testing Can't Show That Bugs Don't Exist
4. The More Bugs You Find, the More Bugs There Are

5. The Pesticide Paradox
6. Not All the Bugs You Find Will Be Fixed
7. When a Bug's a Bug Is Difficult to Say
8. Product Specifications Are Never Final
9. Software Testers Aren't the Most Popular Members of a Project Team
10. Software Testing Is a Disciplined Technical Profession

1. It's Impossible to Test a Program Completely

It is not possible to test program completely due to four key reasons:

The number of possible inputs is very large.

· The number of possible outputs is very large.

· The number of paths through the software is very large.

· The software specification is subjective. You might say that a bug is in the eye of the beholder.

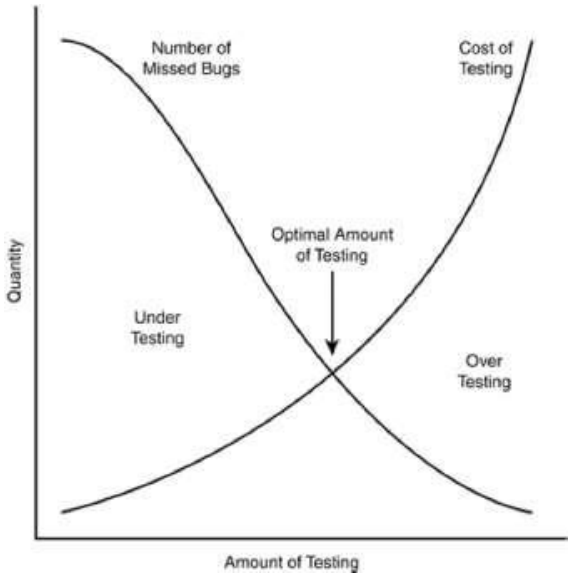
Consider the example Microsoft Windows Calculator:

Assume that you are assigned to test the Windows Calculator. You decide to start with addition. You try 1+0=. You get an answer of 1. That's correct. Then you try 1+1=. You get 2. The calculator accepts a 32-digit number, so you must try all the possibilities up to 1+99999999999999999999999999999999=

2. Software Testing Is a Risk-Based Exercise

If you decide not to test every possible test scenario, you've chosen to take on risk. In the calculator example, what if you choose not to test that 1024+1024=2048? It's possible the programmer accidentally left in a bug for that situation. If you don't test it, a customer will still use it, and he or she will discover the bug. It'll be a costly bug, too, since it wasn't found until the software was in the customer's hands.

If you attempt to test everything, the costs go up dramatically and the number of missed bugs declines to the point that it's no longer cost effective to continue. If you cut the testing short or make poor decisions of what to test, the costs are low but you'll miss a lot of bugs. The goal is to hit that optimal amount of testing so that you don't test too much or too little.



3. Testing Can't Show That Bugs Don't Exist

It can show that bugs exist, but it can't show that bugs don't exist. You can perform your tests, find and report bugs, but at no point can you guarantee that there are no longer any bugs to find. You can only continue your testing and possibly find more.

4. The More Bugs You Find, the More Bugs There Are

A tester will go for long spells without finding a bug. He'll then find one bug, then quickly another and another. There are several reasons for this:

- Programmers have bad days. Like all of us, programmers can have off days. code written one day may be perfect; code written another may be sloppy. One bug can be a tell-tale sign that there are more nearby.
- Programmers often make the same mistake. Everyone has habits. A programmer who is prone to a certain error will often repeat it.
- Some bugs are really just the tip of the iceberg. Very often the software's design or architecture has a fundamental problem. A tester will find several bugs that at first may seem unrelated but eventually are discovered to have one primary serious cause.

5. The Pesticide Paradox

The more you test software, the more immune it becomes to your tests.

The spiral model of software development the test process repeats each time around the loop. With each iteration, the software testers receive the software for testing and run their tests. Eventually, after several passes, all the bugs that those tests would find are exposed. Continuing to run them won't reveal anything new.To overcome the pesticide paradox, software testers must continually write new and different tests to exercise different parts of the program and find more bugs.

6. Not All the Bugs You Find Will Be Fixed

There are several reasons why you might choose not to fix a bug:

- **There's not enough time.** In every project there are always too many software features, too few people to code and test them, and not enough room left in the schedule to finish.If you're working on a tax preparation program, April 15 isn't going to move—you must have your software ready in time.
- **It's really not a bug.** Maybe you've heard the phrase, "It's not a bug, it's a feature!" It's not uncommon for misunderstandings, test errors, or spec changes to result in would-be bugs being dismissed as features.
- **It's too risky to fix.** Unfortunately, this is all too often true. Software is fragile, inter-twined, and sometimes like spaghetti. You might make a bug fix that causes other bugs to appear. Under the pressure to release a product under a tight schedule, it might be too risky to change the software. It may be better to leave in the known bug to avoid the risk of creating new, unknown ones.
- **It's just not worth it.** This may sound harsh, but it's reality. Bugs that would occur In frequently or bugs that appear in little-used features may be dismissed. Bugs that have work around, ways that a user can prevent or avoid the bug, are often not fixed. It all comes down to a business decision based on risk.

7. When a Bug's a Bug Is Difficult to Say

If there's a problem in the software but no one ever discovers it—not programmers, not testers, and not even a single customer—is it a bug?

To claim that the software does or doesn't do "something" implies that the software was run and that "something" or the lack of "something" was witnessed. Since you can't report on what you didn't see, you can't claim that a bug exists if you didn't see it.

8. Product Specifications Are Never Final

Software is getting larger and gaining more features and complexity, resulting in longer and longer development schedules. This result in conflict, and the result is a constantly changing product specification.

As a software tester, you must assume that the spec will change. Features will be added that you didn't plan to test. Features will be changed or even deleted that you had already tested and reported bugs on.

9. Software Testers Aren't the Most Popular Members of a Project Team

The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

Following points to keep the peace with your fellow teammates:

- Find bugs early.
- ·Temper your enthusiasm.
- ·Don't always report bad news.

10. Software Testing Is a Disciplined Technical Profession

Most software is now developed with a disciplined approach that has software testers as core members of their staff. This job that requires training and discipline, and allows for advance men

ii) On which factors the hardware should be selected which we needed?

(Any 3 Factors- each 2 Marks)

There are many hardware factors like Types, Speed, Size, Memory and cost, features

According to your software select types of hardware you will need like Printers, Scanners, sound cards etc.

Decide which brands, Models are available for that hardware. Work with your sales and marketing people to create a list of hardware to test with. With the help of PC Magazine or Mac World get an idea of what hardware is available and what is popular.

When you start configure your hardware select features, modes and options for that hardware. Color printers can print in black and white or color, they can print in different quality modes, and can have settings for printing photos or text. Every device has options, and your software may not need to support all of them. A good example of this is computer games. Many require a minimum number of display colors and resolution.

Identify Your Software's Unique Features That Work with the Hardware Configurations For example, if you're testing a word processor such as WordPad you don't need to test the file save and load feature in each configuration. File saving and loading has nothing to do with printing. A good test would be to create a document that contains different (selected by equivalence partitioning, of course) fonts, point sizes, colors, embedded pictures, and so on. You would then attempt to print this document on each chosen printer configuration.

5. Attempt any <u>TWO</u> of the following: Marks 16

a) What is 'Boundary Conditions' and 'Sub-boundary conditions'? Why these are required?

(Boundary conditions-4 Marks, Sub-boundary conditions-4 Marks)

[Note: Any other relevant example of Boundary and Sub-boundary condition can also be considered]

Boundary conditions: Many systems have tendency to fail on boundary. So it is important to test boundary conditions of application. E.g. If we want to display numbers from 10 to 49, the condition will be, If (n>=10 AND n<50). But if you have given the condition as: If(n>10 AND n<50) then 10 will not get printed. Or If (n>=10 AND n<=50) then 50 will also get printed which is wrong. So for this example we have to test 9, 10, 11 and 48,49,50 values of 'n'. These values

are respectively lower_boundary-1, lower_boundary, lower_boundary+1, upper_boundary-1, upper_boundary, upper_boundary+1.

Types of boundary conditions: Numeric, character, position, quality, speed, location, size. For this type the characteristics can be: First/Last, Min/Max, Over/Under, Highest/Lowest.

For each boundary condition include boundary value at least one in valid test case. Include value just beyond boundary at least one in invalid test case. Include test cases with input such that outputs at boundaries are produced.

Testing the Boundary Edges: Testing outside the boundary is adding one, or a bit more, to the maximum value and subtracting one, or a bit more, from the minimum value. i.e.

- First-1/Last+1
- Start-1/Finish+1
- Largest+1/Smallest-1
- Min-1/Max+1
- Just Over/Just Under

E.g. If a text entry field allows 1 to 255 characters, try entering 1, 2 character and 254,255 characters as the valid partition. Enter 0 and 256 characters as the invalid partitions.

Sub-Boundary Conditions:

Some boundaries though are internal to the software, aren't necessarily apparent to an end user but still need to be checked by the software tester. These are known as sub-boundary conditions. Two examples are Powers-of-Two and the ASCII table.

1. Power-of-two: Computers and software are based on binary numbers. Consider the following table showing common powers-of-two units and their equivalent values.

Powers-of-Two	
Term	Range or Value
Bit	0 or 1
Nibble	0-15
Byte	0-255
Word	0-4,294,967,295
Kilo	1,024

When equivalence partitions are created, include powers-of-two boundary conditions. E.g. if your software accepts a range of numbers from 1 to 1000, include in valid partition 1,2and 999, 1000, as boundary conditions. To cover any possible powers-of-two sub-boundaries, also include the nibble boundaries of 14, 15, and 16, and the byte boundaries of 254, 255, and 256.

2. ASCII Table: Consider the partial ASCII table shown below.

A Partial ASCII Table of Values

Character	ASCII Value	Character	ASCII Value
Null	0	B	66
Space	32	Y	89
/	47	Z	90
0	48	[	91
1	49	‘	96
2	50	a	97
9	57	b	98
:	58	y	121
@	64	z	122
A	65	{	123

If you're testing software that performs text entry or text conversion, it is necessary to consider ASCII table for valid and invalid partition. For example, if you are testing a text box that accepts only the characters A-Z and a-z, you should include in your invalid partition the values just below and above those in the ASCII table i.e. @, [, ', and {.

b) List and explain with example the generic code review check list.

(Listing-2 Marks, Explanation of any 6 errors-1 Mark each, At least 2 examples in each type of error)

1. Data reference errors:

They are the bugs caused by using a variable, constant, array, string or record that has not been declared or initialized.

- Is an uninitialized variable referenced?
- Are array values within the array dimensions?
- Is a variable being used where a constant actually works better
- Is memory allocated for referenced pointers?
- Is a variable ever assigned a value that is of different type?
- If a data structure is referenced in multiple functions or subroutines, is the structure defined identically in each one?

2. Data declaration errors:

They are caused by improperly declaring variables or constants.

- Are all the variables assigned the correct length, type and storage class?
- If a variable is initialized at the declaration time, is it properly initialized and consistent with its type?
- Are there any variables with same names?
- Are any variables declared that are never referenced or are referenced only once?
- Are all the variables explicitly declared within their specific module?

3. Computation errors:

Computational errors are bad math.

- Do any calculations that use variables have different data types, such as adding an integer to a floating-point number?
- Do any calculations that use variables have the same data type but are different lengths, adding a byte to a word, for example?

- Are the compiler's conversion rules for variables of inconsistent type or length understood and taken into account in any calculations?
- Is the target variable of an assignment smaller than the right-hand expression?
- Is overflow or underflow in the middle of a numeric calculation possible?
- Is it ever possible for a divisor/modulus to be zero?
- For cases of integer arithmetic, does the code handle that some calculations, particularly division, will result in loss of precision?
- Can a variable's value go outside its meaningful range? E.g. less than 0% or greater than 100%?
- For expressions containing multiple operators, is there any confusion about the order of evaluation and is operator precedence correct? Are parentheses needed for clarification?

4. Comparison errors:

Less than, greater than, equal, not equal, true, false. Comparison and decision errors are very susceptible to boundary condition problems.

- Are the comparisons correct?
- Are the comparisons between fractional and floating point values?
- Does each Boolean expression state what it should state?
- Are the operands of a Boolean operator Boolean?

5. Control flow errors:

Control flow errors are the result of loops and other control constructs in the language not behaving as expected.

- If the language contains statement groups such as begin...end and do...while, are the ends explicit and do they match their appropriate groups?
- Is there a possibility of premature loop exit?
- Is it possible that a loop never executes?
- Will the loop eventually terminate?
- Does any error cause unexpected flow through the loop?

6. Subroutine parameter errors:

Subroutine parameter errors are due to incorrect passing of data to and from software subroutines.

- Do the types and sizes of the parameters received by a subroutine match those sent by the calling code?
- If constants are ever passed as arguments, are they accidentally changed in the subroutine?
- Does a subroutine alter a parameter that is indented only as a input value?
- If global variables are resent, do they have similar definitions in all referencing subroutines?
- · Do the units of each parameter match the units of each corresponding argument.
- If a subroutine has multiple entry points, is a parameter ever referenced that isn't associated with the current point of entry?

7. Input/output errors:

These errors include anything related to reading from a file, accepting input from a key board or mouse, and writing to an output device such as a printer or screen

- · Does the software strictly adhere to the specified format of the data being read or written by the external device?
- Have all the error messages been checked for correctness, grammar and spelling?
- Are all errors handled by software in an expected way?
- If the file or peripheral is not ready, is that error condition handled?
- Does the software handle the situation of the external device being disconnected?

8. Other checks:

The list below defines a few items that didn't fit well in the other categories

- Will the software work with languages other than English?
- If the software is intended to be portable to other compilers have allowances been made for this?
- Does compilation of program produce any warning messages?
- Will the software operate with different amount of available memory?

c) What are the goals of Test Planning?

(What is test planning/purpose-4 Marks, goals-4 Marks)

[Note: Consider goals of Test Case Planning as it is a part of test planning and give some marks accordingly]

The software test plan is the primary means by which software testers communicate to the product development team what they intend to do.

The ANSI/IEEE Standard 829/1983 for Software Test Documentation states that the purpose of a software test plan is:

To prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan.

The form the test plan takes is a written document which organizes the system requirements in terms of how these requirements will be tested. It defines the elements required in test planning

such as test goals, test groups, test cases. Although the end result is a piece of paper or online document, it is simply a by-product of the detailed planning process that's undertaken to create it. The important is the planning process than the resulting document.

The ultimate goal of the test planning process is communicating the software test team's intent, its expectations, and its understanding of the testing that's to be performed.

The goals of test planning are:

- Allows the project team to consider ways to reduce the testing efforts without being under time pressure.
- Helps remove misunderstandings between developers and end-users before the solution is developed.
- Creates verifiable system components that will become part of the project documentation.
- Enables a more reliable estimate of the testing effort up front.
- Helps to identify problem areas and focuses the testing team's attention on the critical paths.
- Reduces the probability of implementing non-tested components.
- It is an integral part of a coherent requirements management process.

6. Attempt any FOUR of the following: Marks 16

a) Explain with example, what is data sharing compatibility.

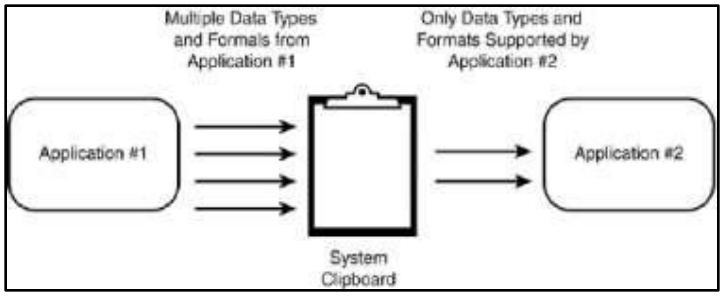
(Each example-1 Mark)

Data sharing compatibility is important to share the data among different applications. A software product that supports and follows the published standards allows users to easily transfer data to and from other software product is a good compatible product.

The different examples of data sharing compatibility are:

1. **File save and File load**: It is a common data sharing method. You save data to a floppy disk and then carry it over to another computer running different software and load it in.

2. **Cut, Copy and paste**: It is another method of sharing data among programs without transferring data to disk. The transfer happens in memory through an intermediate program called the clipboard as shown in the diagram. Whenever the user performs the cut or copy the data that's chosen is placed in the clipboard. When he does a paste it is copied from the clipboard to the destination software. If you are compatibility testing program you need to make sure that the data is copied properly out of the clipboard to the desired program.



3. **DDE** (Dynamic data exchange), **COM** (component object model) and **OLE** (object linking and embedding): These are the common methods for transferring data between two applications. With DDE and OLE data can flow from one application to the other in real time. e.g. . If author linked the pie chart into the report as an object, when the underlying numbers for the chart change, the new graphics will automatically appear in the report.

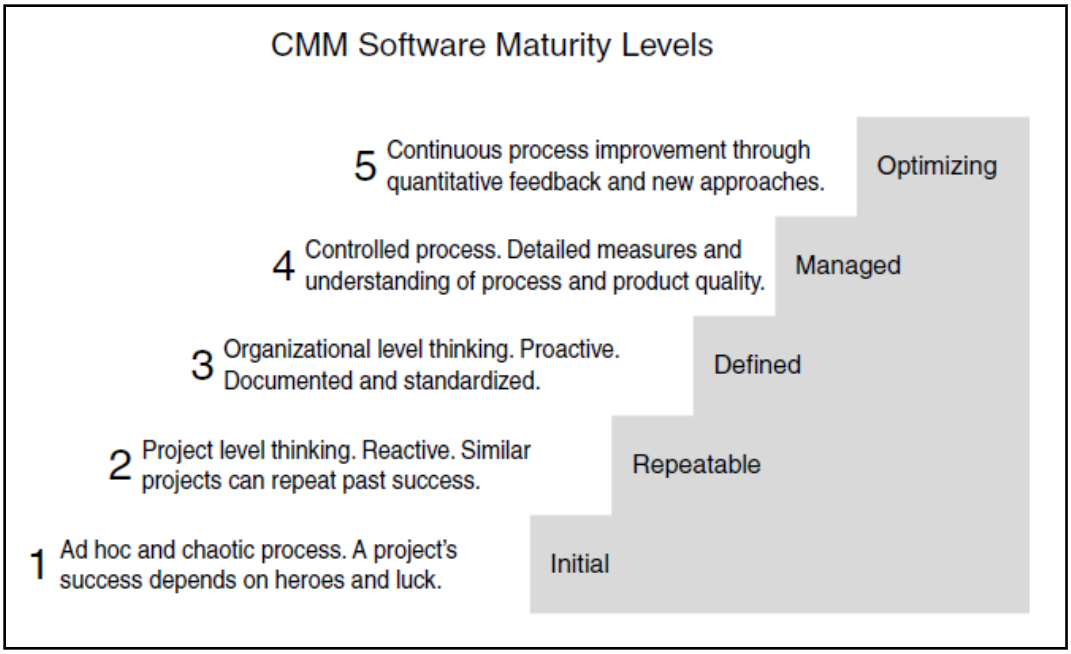
4. **File export and import**: It is the means that many programs use to be compatible with older versions of themselves and with other program .E.g. Microsoft Word can import 23 different file formats.

b) Discuss CMM model in detail.

(Diagram-1 Mark, Explanation-3 Marks)

Capability Maturity Model (CMM)

The Capability Maturity Model for Software (CMM) is an industry-standard model for defining and measuring the maturity of a software company's development process and for providing direction on what they can do to improve their software quality. It provides a means to assess a company's software development maturity and determine the key practices they could adopt to move up to the next level of maturity.



• **Level 1**: Initial. The software development processes at this level are ad hoc and often chaotic. The project's success depends on heroes and luck. There are no general practices for planning, monitoring, or controlling the process. It's impossible to predict the time and cost to develop the software. The test process is just as ad hoc as the rest of the process.

• **Level 2**: Repeatable. This maturity level is best described as project-level thinking. Basic project management processes are in place to track the cost, schedule, functionality, and quality of the project. Lessons learned from previous similar projects are applied. There's a sense of discipline. Basic software testing practices, such as test plans and test

cases, are used.

• **Level 3:** Defined. Organizational, not just project specific, thinking comes into play at this level. Common management and engineering activities are standardized and documented. These standards are adapted and approved for use on different projects. The rules aren't thrown out when things get stressful. Test documents and plans are reviewed and approved before testing begins. The test group is independent from the developers. The test results are used to determine when the software is ready.

• **Level 4:** Managed. At this maturity level, the organization's process is under statistical control. Product quality is specified quantitatively beforehand (for example, this product won't release until it has fewer than 0.5 defects per 1,000 lines of code) and the software isn't released until that goal is met. Details of the development process and the software's quality are collected over the project's development, and adjustments are made to correct deviations and to keep the project on plan.

• **Level 5:** Optimizing. This level is called Optimizing (not "optimized") because it's continually improving from Level 4. New technologies and processes are attempted, the results are measured, and both incremental and revolutionary changes are instituted to achieve even better quality levels. Just when everyone thinks the best has been obtained, the crank is turned one more time, and the next level of improvement is obtained.

c) Describe low level specification Test Technique.

(Any 4 test techniques-1 Mark each)

Low-Level Specification Test Techniques:

Specification Attributes Checklist:

- Complete: Is anything missing or forgotten? Is it thorough? Does it include everything necessary to make it stand alone?
- Accurate: Is the proposed solution correct? Does it properly define the goal? Are there any errors?
- Precise, Unambiguous, and Clear: Is the description exact and not vague? Is there a single interpretation? Is it easy to read and understand?
- Consistent: Is the description of the feature written so that it doesn't conflict with itself or other items in the specification?

- Relevant: Is the statement necessary to specify the feature? Is it extra information that should be left out? Is the feature traceable to an original customer need?
- Feasible: Can the feature be implemented with the available personnel, tools, and resources within the specified budget and schedule?
- Code-free: Does the specification stick with defining the product and not the underlying software design, architecture, and code?
- Testable: Can the feature be tested? Is enough information provided that a tester could create tests to verify its operation?

Specification Terminology Checklist: A list of problem words to look for while reviewing a specification

- Always, Every, All, None, Never: If you see words such as these that denote something as certain or absolute, make sure that it is, indeed, certain. Put on your tester's hat and think of cases that violate them.
- Certainly, Therefore, Clearly, Obviously, Evidently: These words tend to persuade you into accepting something as a given. Don't fall into the trap.
- Some, Sometimes, Often, Usually, Ordinarily, Customarily, Most, Mostly: These words are too vague. It's impossible to test a feature that operates "sometimes."
- Etc., And So Forth, And So On, Such As: Lists that finish with words such as these aren't testable. Lists need to be absolute or explained so that there's no confusion as to how the series is generated and what appears next in the list.
- Good, Fast, Cheap, Efficient, Small, Stable: These are unquantifiable terms. They aren't testable. If they appear in a specification, they must be further defined to explain exactly what they mean.
- Handled, Processed, Rejected, Skipped, Eliminated: These terms can hide large amounts of functionality that need to be specified.
- If…Then… (but missing Else):Look for statements that have "If…Then" clauses but don't have a matching "Else." Ask yourself what will happen if the "if" doesn't happen.

d) What is ISO 9000?

(Explanation-3 Marks, Any two Requirements-1 Mark)

ISO 9000 is an international standards organization that sets standards for quality management and quality assurance.ISO 9000 is a family of standards on quality management and quality assurance that defines a basic set of good practices that will help a company consistently deliver products (or services) that meet their customer's quality requirements.

ISO 9000 works well for two reasons:

- It targets the development process, not the product. It's concerned about the way an organization goes about its work, not the results of the work.
- ISO 9000 dictates only what the process requirements are, not how they are to be achieved. For example, the standard says that a software team should plan and perform product design reviews but it doesn't say how that requirement should be accomplished.

The sections of the ISO 9000 standard that deal with software are ISO 9001 and ISO 9000-3. ISO 9001 is for businesses that design, develop, produce, install, and service products. ISO 9000-3 is for businesses that develop, supply, install, and maintain computer software.

Some of the requirements in ISO 9000-3 include

- Develop detailed quality plans and procedures to control configuration management, product verification and validation (testing), nonconformance (bugs), and corrective actions (fixes).
- Prepare and receive approval for a software development plan that includes a definition of the project, a list of the project's objectives, a project schedule, a product specification, a description of how the project is organized, a discussion of risks and assumptions, and strategies for controlling it.
- Communicate the specification in terms that make it easy for the customer to understand and to validate during testing.
- Plan, develop, document, and perform software design review procedures.
- Develop procedures that control software design changes made over the product's life cycle.
- Develop and document software test plans.
- Develop methods to test whether the software meets the customer's requirements.

- Perform software validation and acceptance tests.
- Maintain records of the test results.
- Control how software bugs are investigated and resolved.
- Prove that the product is ready before it's released.
- Develop procedures to control the software's release process.
- Identify and define what quality information should be collected.
- Use statistical techniques to analyze the software development process.
- Use statistical techniques to evaluate product quality.

e) Describe the following terms:

i) Test Phases

ii) Test Strategy

(Test Phases -2 Marks, Test Strategy-2 Marks)

i) Test Phases:

To plan the test phases, the test team will look at the proposed development model and decide whether unique phases, or stages, of testing should be performed over the course of the project.e.g. In a code-and-fix model, there's probably only one test phase. In the spiral models, there can be several test phases from examining the product specifications to acceptance testing. Test planning is also one of the test phases. The test planning process should identify each proposed test phase and make each phase known to the project team.

ii) Test Strategy:

The test strategy describes the approach that the test team will use to test the software both overall and in each phase. You should decide whether to use black-box testing or White-box testing or if both then when will you apply each and to which parts of the software? Some part of the code might be tested manually and other code with tools and automation. If tools will be used, do they need to be developed or can existing commercial solutions be purchased? If so, which ones to be used? It is sometimes more efficient to outsource the entire test effort to a specialized testing company.