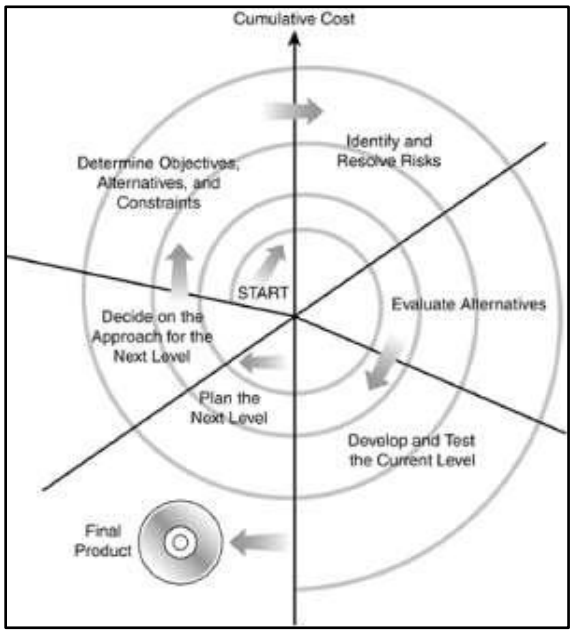
1. a) Attempt any THREE of the following. (4x3=12) Marks

a) Explain spiral model.

[1 mark for diagram, 3 marks for explanation]

Ans:



The spiral model was introduced by Barry Boehm in 1986. It's used often and has proven to be an effective approach to developing software. Spiral model has important features of other models with its own special features.

The general idea behind the spiral model is that you don't define everything in detail at the very beginning. You start small, define your important features, try them out, get feedback from your customers, and then move on to the next level. You repeat this until you have your final product.

Each time around the spiral involves six steps:

1. Determine objectives, alternatives, and constraints.
2. Identify and resolve risks.
3. Evaluate alternatives.
4. Develop and test the current level.
5. Plan the next level.
6. Decide on the approach for the next level.

The spiral model is a bit similar to waterfall (the steps of analysis, design, develop, test), a bit similar to code-and-fix (each time around the spiral), and a bit similar to big-bang (look at it from the outside). If you're a tester, you'll like this model. You'll get a chance to influence the product early by being involved in the preliminary design phases. You'll see

where the project has come from and where it's going. And, at the very end of the project, you won't feel as rushed to perform all you're testing at the last minute. You've been testing all along, so at the end only a validation has to be done.

Advantages:

1. Suitable for designing and managing large and complex projects.
2. Risk analysis is an integral part of this model and is performed in every spiral, leading to confidence in the project.

b) Describe boundary condition with example.
[3 Marks for explanation, 1 Mark for example]

Ans: 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. 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 in at least one valid test case. Include value just beyond boundary in at least one invalid test case. Include test cases with input such that outputs at boundaries are produced.

Testing the Boundary Edges:

- 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.

c) Why do bug occurs?
[Any 4 rules 1 Mark each]

Ans: 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.

d) List four computational errors.
[Any 4 computational errors 1 Mark each]

Ans: Computational errors:

- 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? For example, could the result of a probability be 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?

b) Attempt any ONE of the following. (6 x1=6)Marks

a) List all the properties of good software tester.
[Any 6 properties of software tester-1 Mark each]

Ans: List of properties that most software tester should have

1. They are explorers: Software testers aren't afraid to venture into unknown situations. They love to get a new piece of software, install it on their PC, and see what happens.

2. They are troubleshooters: Software testers are good at figuring out why something doesn't work. They love puzzles.

3. They are relentless: Software testers keep trying. They may see a bug that quickly vanishes or is difficult to re-create. Rather than dismiss it as a fluke, they will try every way possible to find it.

4. They are creative: Testing the obvious isn't sufficient for software testers. Their job is to think up creative and even off-the-wall approaches to find bugs.

5. They are (mellowed) perfectionists: They strive for perfection, but they know when it becomes unattainable and they're OK with getting as close as they can.

6. They exercise good judgment: Software testers need to make decisions about what they will test, how long it will take, and if the problem they're looking at is really a bug.
7. They are tactful and diplomatic: Software testers are always the bearers of bad news. They know how to work with Programmers tactfully.
8. They are persuasive: Bugs that testers find won't always be viewed as severe enough to be fixed. Testers need to be good at making their points clear, demonstrating why the bug does indeed need to be fixed, and following through on making it happen.

b) What are the benefits of automation and tools and explain.
[1 mark for each benefit]

Ans: Benefits of automation testing:

1. Speed: Think about how long it would take you to manually try a few thousand test cases for the windows Calculator. You might average a test case every five seconds or so. Automation might be able to run 10, 100 even 1000 times that fast.
2. Efficiency: While you're busy running test cases, you can't be doing anything else. If you have a test tool that reduces the time it takes for you to run your tests, you have more time for test planning and thinking up new tests.
3. Accuracy and Precision: After trying a few hundred cases, your attention may reduce and you'll start to make mistakes .A test tool will perform the same test and check the result perfectly, each and every time.
4. Resource Reduction: Sometimes it can be physically impossible to perform a certain test case. The number of people or the amount of equipment required to create the test condition could be prohibitive. A test tool can used to simulate the real world and greatly reduce the physical resources necessary to perform the testing.
5. Simulation and Emulation: Test tools are used to replace hardware or software that would normally interface to your product. This "face" device or application can then be used to drive or respond to your software in ways that you choose-and ways that might otherwise be difficult to achieve.
6. Relentlessness: Test tool and automation never tire or give up. It will continuously test the software.

2. Attempt any FOUR of the following. (4x4=16) Marks

a) Give four low level specification techniques.
[Any four specification techniques 1 mark each]

Ans: 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.

b) Write difference between static and dynamic testing.
[Any 4 differences, 1 Mark each]

Ans:

Static Testing	Dynamic Testing
1. During static testing, you have a checklist to check whether the work you are doing is going as per the set standards of the organization.	1. Dynamic Testing involves working with the software, giving input values and checking if the output is as expected.
2. This test is done without actually executing the application.	2. This test is done by executing the application.
3. Verification activities fall into the category of Static Testing.	3. Validation activities fall into the category of Dynamic Testing.
4. Methodologies used are Review's, Inspection's and Walkthrough's.	4. Methodologies used are Unit Tests, Integration Tests, System Tests and Acceptance Tests.

c) When you say that user interface is comfortable?
[Explanation 4 Marks]

Ans: Comfortable: Software should be comfortable to use. Following things should be consider while deciding comfortable software.

- Appropriateness: Software should look and feel proper for what it's doing and who it's for. A financial business application should not be colorful and of loud sound like games. Software neither be too garish nor too plain.
- Error handling: A program should warn users before a critical operation and allow users to restore data lost because of a mistake. People take the Undo/Redo feature for granted, it should exists in program.
- Performance: Being fast is always not a good thing. More than one program gives error messages can't be read by the user. If an operation is slow, it should at least give the user feedback on how much longer it will take and show that it's still working and hasn't frozen. E.g. The status bar shows how much of the work has been completed.

d) What is beta testing? What are the advantages of beta testing?
[2 Marks for explanation, 2 Marks for advantages]

Ans: Beta testing is the term used to describe the external testing process in which the software is sent out to a selected group of potential customers who use it in a real-world environment. Beta testing usually occurs at the end of the product development cycle and ideally should just be a validation that the software is ready to release to real customers.

Advantages:

- It is possible to know the pulse of the software users like what features they like , why they use the product and how they use it.

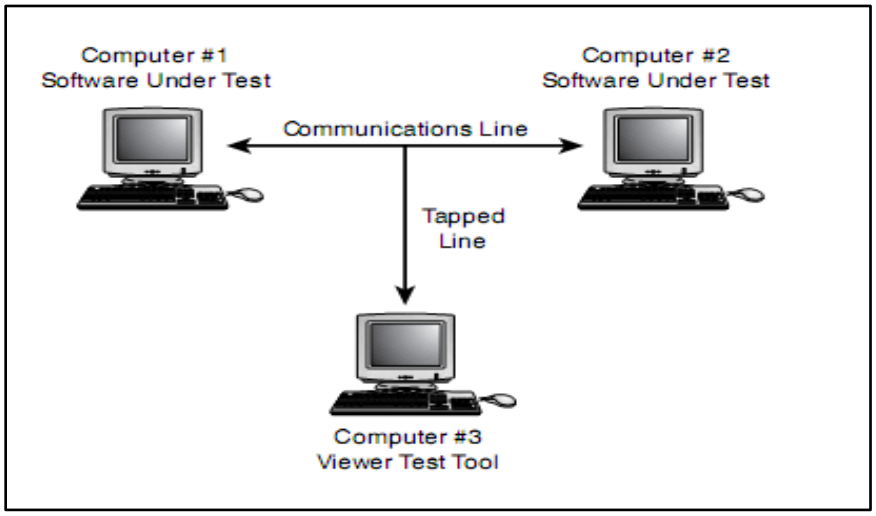
- Software developers can provide their software products to end users to expose them to real life usage. This will help software developers to determine whether the product can withstand the different configurations of the end-user computers.
- Beta testing at the user end will also help software developers test the product under different conditions.
- Beta testing will also help the software developers to know and understand end user opinion and experience.
- Software developers will also understand whether the end user benefited from the features included in the software.
- Software developers may also get a chance to estimate the possible profit levels based on the feedback received by the end users.

e) Explain viewers and monitors test tools.
[Explanation 3 mark, examples 1 mark]

Ans: Viewers and Monitors: A viewer or monitor test tool allows you to see details of the software's operations that you wouldn't normally be able to see. Anything that lets you see into the system and look at data that the average user wouldn't be able to see can be classified as a viewer test tool.

Examples:

1. A code coverage analyzer is the example of a viewing tool. Most code coverage analyzers are invasive tools because they need to be compiled and linked into the program to access the information they provide.
2. The following figure shows another type of viewer - Communication Analyzer. It allows to view data moving across network. This tool simply taps into network cable, pulls of data as it passes by and displays it on another computer.



3. The code debuggers that come with most compilers are also considered viewers because they allow programmers or white-box testers to view internal variable values and program states.

f) List benefits of coding standards and guidelines.
[1 Mark for what is standard and guideline and 3 Marks for benefits]

Ans: Coding Standards and Guidelines:

There are also problems where the code may operate properly but may not be written to meet a specific standard or guideline. Standards are the established, fixed, have-to-follow-them. Guidelines are the suggested best practices, the recommendations, the preferred way of doing things. Standards are compulsory to be followed but guidelines are not.

Benefits of standards and guidelines:

- Reliability. It's been shown that code written to a specific standard or guideline is more reliable and secure than code that isn't.
- Readability/Maintainability. Code that follows set standards and guidelines is easier to read, understand, and maintain.
- Portability. Code often has to run on different hardware or be compiled with different compilers. If it follows a standard, it will likely be easier to move it to a different platform.

3. Attempt any FOUR of the following. (4x4=16) Marks

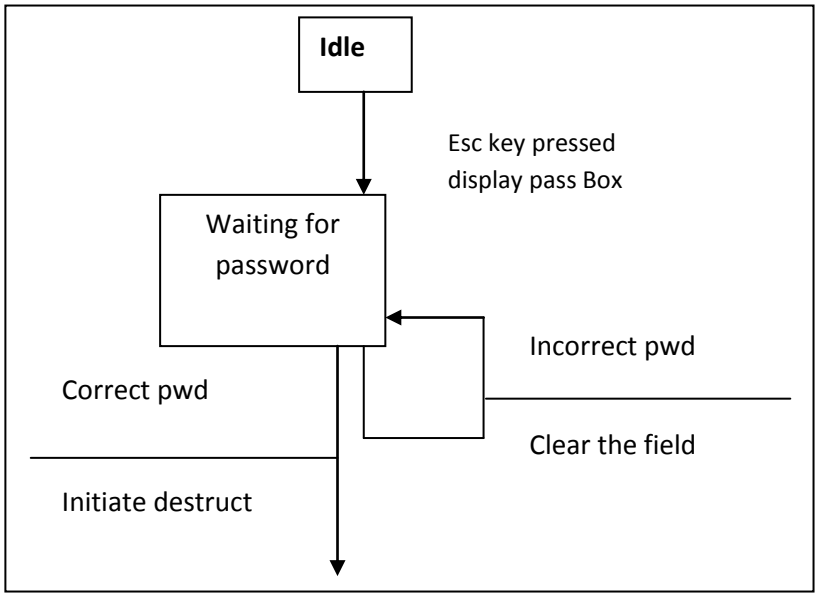
a) Explain state testing with state transition diagram.
[2 Marks for diagram & 2 Marks for explanation]

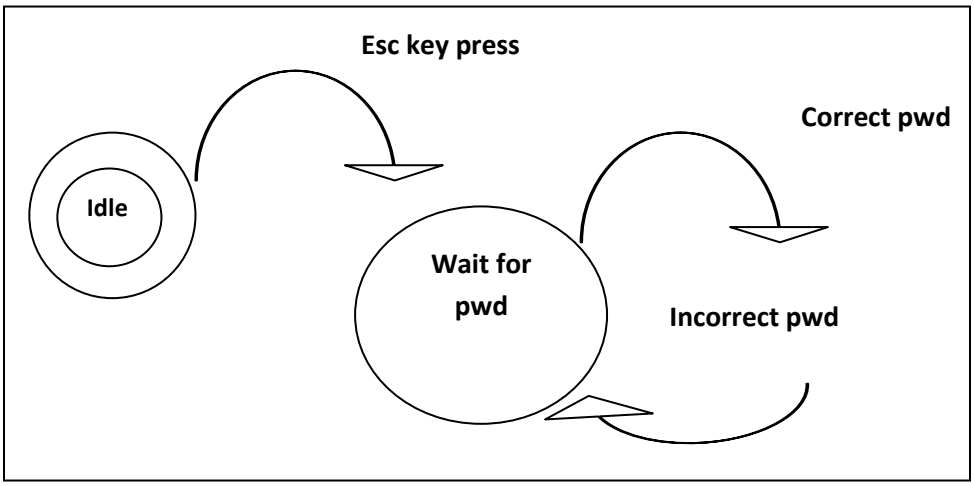
Ans: The state testing is to verify program logic flow through its various states. A software state is condition or mode that software is currently in.

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

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.





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.

b) Write four test cases to test the "rediff home page".
[1Mark for each test case]

[Note: please consider any other test cases related with rediff home page including Login Form]

Ans: TEST CASE NAME: (rediff home page)

Test case ID	Description	Input Data	Expected Result	Actual Result	Status
1	Click on rediffmail link		Username and password page should get displayed	Username and password page is displayed	pass
2	Click on News link		News page should get displayed	News page is displayed	pass
3	Click on shopping link		Shopping page should get displayed	Shopping page is displayed	pass
4	Click on sign in link		Sign in dialog box should get displayed	Sign in dialog box is displayed	pass

c) Explain compatibility testing.
[Definition 1 Mark, Explanation 1 Mark, 2 Marks for examples]

Ans: Software compatibility testing means checking that your software interacts with & shares information correctly with other software

This interaction could occur between two programs simultaneously running on the same computer or even different computers connected through the internet thousands of miles apart

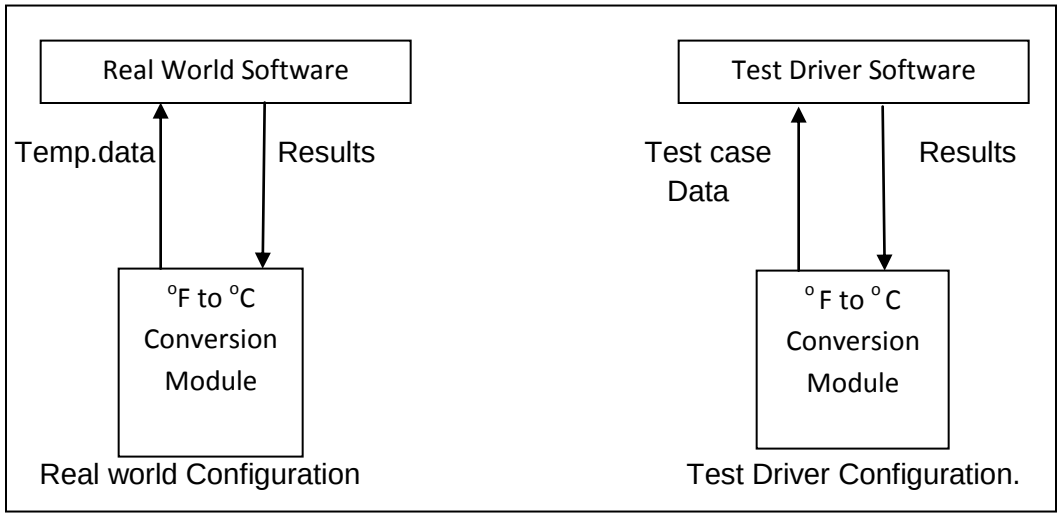
Ex:

- Cutting text from a web page & pasting it into a document opened in your word processor.
- Saving accounting data from one spread sheet program.
- Having photograph touch-up software work correctly on different versions of the same operation system.
- Having your word processor load in the names & addresses from your contact management program from your contact management program & print out personalized invitation & envelopes.

d) Describe the bottom up approach of integration testing.
[1 Mark for definition, 3 Marks for diagram & their explanation]

Ans: In bottom-up approach requires the lower-level units be tested & integrated first. These units referred as modules.

Diagram shows example of bottom-Up approach



In bottom-Up testing we can write own modules called test driver that exercise the modules you are testing as shown in diagram.

-These drivers send test-case data to the modules under test, read back the result & verify that whether they are correct.

e) What is usability testing?
[1 mark for definition, 3 marks for explanation]

Ans: Usability testing is to test functionality & effectiveness in the interaction with software.
Usability testing is a technique used in interaction design to evaluate a product by testing it on users. This can be seen as an irreplaceable usability practice, since it gives direct input on how real users use the system. This is in contrast with usability inspection methods where experts use different methods to evaluate a user interface without involving users.

Usability testing focuses on measuring a human-made product's capacity to meet its intended purpose. Examples of products that commonly benefit from usability testing are foods, consumer products, web sites or web applications, computer interfaces, documents, and devices. Usability testing measures the usability, or ease of use, of a specific object or set of objects, whereas general human-computer interaction studies attempt to formulate universal principles.

4. a) Attempt any <u>THREE</u> of the following. (4x3=12) Marks

a) Write need of test case planning.
[1 mark for each point]

Ans: Test planning is important for following reasons:

1. **Organization**: Proper planning will organize test cases so that all testers & other project team members can review & use them effectively.
2. **Repeatability**: Same test cases run several times to look new bugs & to make sure that old ones have been fixed.
3. **Tracking**: Tracking of -how many test cases did you plan to run?
   -How many did you run on last software release?
   -How many passed & how many failed?
   -Were any test cases skipped?
4. **Proof of testing**: In a few high-risk industries, the software test team must prove that it did indeed run the tests that it planned to run. It could actually be illegal to release software in which a few test cases were skipped. Proper test case planning & tracking provides a means for proving what was tested.

b) Explain the branch coverage with example.
[Definition 1 Mark, 3 Marks for example]

Ans: Attempting to cover all the paths in the software is called path testing.
The simplest form of path testing is called branch coverage testing.
Consider example.

```
PRINT "Hello World"
IF Date$= "01-01-2000" THEN
PRINT "Happy New Year"
END IF
PRINT " The date is :"Date$
PRINT " The time is:" Time$
END
```

If you test this program with goal 100 percent statement coverage, you would need to run only a single test case with the Date$ variable set to January 1,2000.
The program would execute following path:
Lines 1,2,3,4,5,6,7
Code coverage analyzer would state that tested every statement & achieved 100 percent coverage.
If we need to test case for that is notase January 1, 2000 .If you did, program would execute the other path.
Lines 1, 2, 5, 6, 7

c) Explain accessibility testing.
[1 mark for each point]

Ans: A serious 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.

d) Enlist major program elements required for localization.
[2 mark for each point]

Ans: 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

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

b) Attempt any **ONE** of the following: (6x1=6) Marks

a) Define and explain in brief various software testing terms
[Any three terms 2 Marks each]

Ans: 1. Precision & Accuracy
2. Verification & Validation
3. Quality & Reliability
4. Testing & Quality Assurance (QA)

Precision and Accuracy

Suppose that you're testing a calculator. Should you test that the answers it returns are precise or accurate? Both? If the project schedule forced you to make a risk-based decision to focus on only one of these, which one would you choose?
What if the software you're testing is a simulation game such as baseball or a flight simulator? Should you primarily test its precision or its accuracy?

Whether the software you test needs to be precise or accurate depends much on what the product is and ultimately what the development team is aiming at (excuse the pun). Software calculator likely demands that both are achieved—a right answer is a right answer. But, it may be decided that calculations will only be accurate and precise to the fifth decimal place. After that, the precision can vary. As long as the testers are aware of that specification, they can tailor their testing to confirm it.(In this point only one example is sufficient)

Verification and Validation

Verification and validation are often used interchangeably but have different definitions. These differences are important to software testing.
Verification is the process confirming that something—software—meets its specification. Validation is the process confirming that it meets the user's requirements. These may sound very similar, but an explanation of the Hubble space telescope problems will help show the difference.

Quality and Reliability

Quality is "a degree of excellence" or "superiority in kind." If a software product is of high quality, it will meet the customer's needs. The customer will feel that the product is excellent and superior to his other choices.
Software testers often fall into the trap of believing that quality and reliability are the same thing. They feel that if they can test a program until it's stable, dependable, and reliable,

they are assuring a high-quality product. Unfortunately, that isn't necessarily true.
Reliability is just one aspect of quality
To ensure that a program is of high quality and is reliable, a software tester must both verify and validate throughout the product development process.

Testing and Quality Assurance (QA)
These two terms are the ones most often used to describe either the group or the process that's verifying and validating the software. consider these definitions:

- The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.
- A software quality assurance person's main responsibility is to create and enforce standards and methods to improve the development process and to prevent bugs from ever occurring.

b) Explain configuration testing in short.
[Definition 1 Mark, Explanation 5 Marks]

Ans: configuration testing is the process of checking the operation of software you are testing with all the various types of hardware.
Consider the different configuration possibilities for standard Windows- based PC used in homes & business.
The PC: There are several well known computer manufacturers such as Dell, gateway & other.
Each one build PCs using components designed or obtained from other manufacturers.
Components: Most PCs are modular & built up from various system boards, components cards & other internal devices such as hard disk drivers, CD-ROM, network cards.
Peripherals: Peripherals are printers, scanners, mouse, keyboard & other devices that plug into system & network cards.
Interfaces: The components & peripherals plug into your PC through various types of interface connectors such as ISA, PCI, USB
Option & memory: Many components & peripherals can be purchased with different hardware & memory sizes.
Device Drivers: All components & peripherals communicate with the operating system & software application through low level software is called device drivers.
But at the time of configuration testing many problems occur for several reasons, all requiring someone to carefully examine the code while running the software under different configuration to find bugs.
General process that should use when planning your configuration testing.
1. Decide the types of hardware you will need.
2. Decide what hardware brands, models & device drivers are available.
3. Decide which hardware features ,modes & options are possible.
4. Pare down the identified hardware configurations to a manageable set.
5. Identify your software's unique features that work with the hardware configurations.

6. Design the test cases to run on each configuration .
7. Execute the tests on each configuration.
8. Return the test until result satisfies your team.

5. Attempt any **TWO** of the Following. (8x2=16) Marks

a) What actions should be performed to apply stress and load testing on your software?
[Stress testing 4 Marks, load testing 4 Marks]

Ans: Stress testing and load testing are basically performed on the software to check the performance of the software.

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

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

Load testing is testing the software under customer expected load.

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

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

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

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

b) Explain the generic code review checklist.
[Any four 2 Marks each]

Ans: 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?

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?
- Do any calculations that use variables have same data type but with different lengths.
- Is the target variable smaller than the right hand expression?
- Is overflow or underflow possible in middle of calculation?
- Can a variable's value go outside its meaningful range?

4. Comparison errors:

- 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:

- 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:

- 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?

7. Input/output errors:

- Have all the error messages been checked for correctness, grammar and spelling?
- Are all errors handled by software in an expected way?

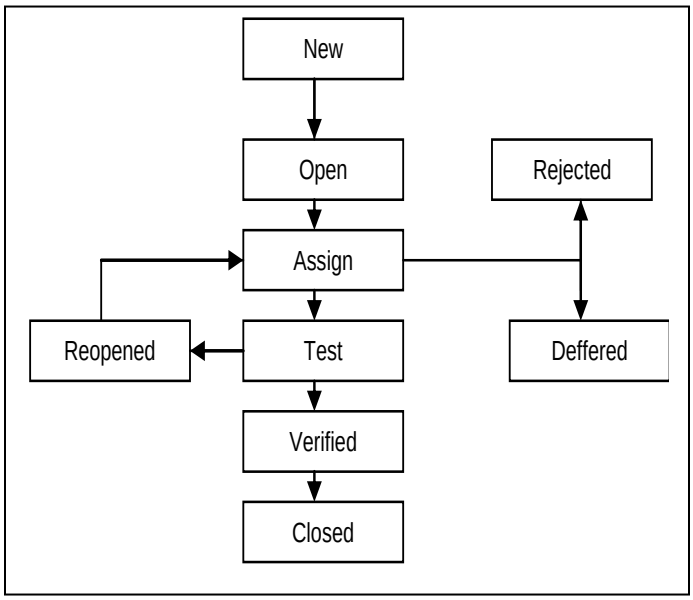
- If the file or peripheral is not ready, is that error condition handled?
- Does the software handle the situation of the external device being disconnected?

8. Other checks:

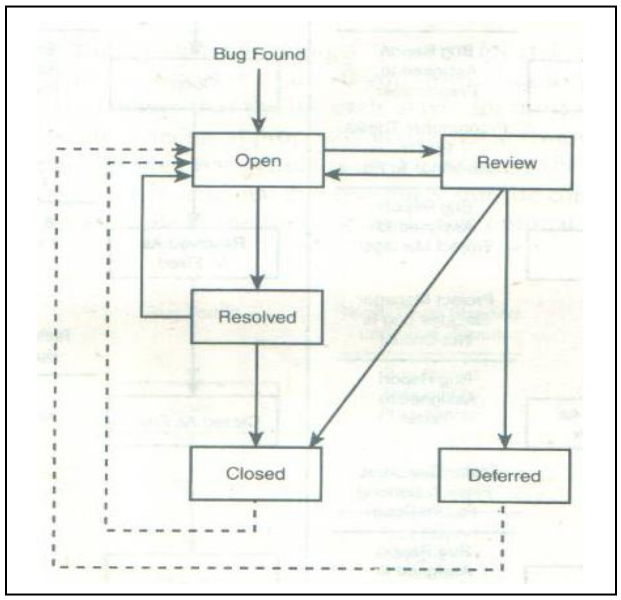
- Will the software work with languages other than English?
- If the software is intended to be portable to other compilers have allowances been made for this?
- Does compilation of program produce any warning messages?
- Will the software operate with different amount of available memory?

c) Draw and explain a bug's life cycle.
[Diagram 2 marks, Explanation 6 marks]

Ans:



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.

6. Attempt any FOUR of the Following. (4x4=16) Marks

a) Explain data sharing compatibility.

[1 Mark for each point]

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

The most common way of transferring the data from one program to other is saving and loading disk files. The different examples of data sharing compatibility are:

1. File saves and File load 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 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. 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) are the common methods for transferring data between two applications.

4. File export and import:
It is the means that many programs use to be compatible with older versions of themselves and with other programs

b) Give comparison of quality assurance and quality control.
[Any four points one mark each- 4 Marks]

Ans:

Quality Assurance	Quality Control
1. QA activities are process oriented.	1. QC activities are product oriented.
2. They measure the process, identify the weakness and suggest improvements.	2. They measure the product, identify the actual defects and suggest improvements.
3. QA is preventive.	3. QC is defective.
4.QA audit is an example of QA activities	4. Testing and reviews are examples of QC activities.
5.The direct result of these activities are changes to the process	5.The direct result of these activities are changes to the product
6.QA is inline function	6. QC is staff function.

c) Explain realities of documentation testing.
[Explanation 4 marks]

Ans: 1. Documentation often gets least attention, budget and resources. There seems to be a mentality that the software project is most important than all other stuff. In reality people buy software product .If you are responsible for testing make sure that you budget time to test the documentation that goes along with the code.

2. It is possible that the people writing the documentation are not experts in what the software does. Work closely with writers to make sure they have the information they need and they are up-to-date with the products design. Tell them about difficult to understand areas so they can better explain those areas in the documentation.

3. Printed document takes time to produce may be weeks or even months. Because of this time difference the documentation may need to be finalized –locked down- before

the software is completed. Hold your documentation release to the last possible minute, and release as much documentation as possible, in electronic format, with the software.

d) What is test-to-pass and test-to-fail?
[Test-to-pass 2 Marks, Test-to-fail 2 Marks]

Ans: The two fundamental approaches of testing software are: Test to Pass and Test to Fail.

When you test to pass you really assure that the software works minimally. You don't push its capabilities. You don't see what you can do to break it. We apply most simplest and straightforward test cases. Whenever we are writing test cases we write test cases with positive scenario and execute the test to pass test cases first to get the confidence that the software works minimally, as one will find the software doesn't work as expected most of the times even in normal circumstances.

Once the tests to pass test cases are written we write test cases for test to fail. Writing the test cases with the sole purpose of breaking the software is called as test to fail or error forcing.

E.g. you are assigned to test the very first prototype and have never been driven. You probably wouldn't get in, start it up, head for the test track, and run it wide open at full speed as hard as you can. You would probably crash and die. With a new car there would be all kind of bugs that would reveal at low speed under normal driving conditions. May be the tyres aren't of the right size, or the brakes are inadequate, or the engine is too loud.

e) Explain translation issues.
[Any four points - 1 Mark each]

Ans: Translation is just the part of overall localization effort. The different translation issues are:

1. Text Expansion:

Although English appears to be wordy at times, but when English is translated to other languages often more characters are necessary to say the same thing. You need to carefully test areas of the software that could be affected by longer text. Look for text that does not wrap correctly, is truncated or is hyphenated incorrectly. The longer text may also lead to system crash.

2. ASCII, DBCS and UNICODE:

ASCII represents only 256 characters which may not be enough to represent all the characters in all languages. The common approach is to use a code page for each language. This solution is fine for languages with less than 256 characters but Japanese, Chinese languages have thousands of symbols. So a system called as DBCs (Double byte character set) is used which uses 2 bytes instead of 1 byte. But ASCII and DBCS suffer from the problem of compatibility. To overcome this Unicode is used.

Unicode provides unique number for every character no matter what the platform, what the program, what the language.

3. Reading left to right and right to left:
Most major operating system provides the built in support for handling languages like Arabic. It is still not a simple matter of translating the text. It requires a great deal of programming to make use of operating system features. It is safe to consider it as a completely new product.

4. Text in Graphics:
The translation issue occurs when the text is used in graphics. The icons used for selecting bold, italic may mean nothing to someone from Japan who does not read English. If there are many of these icons it would be expensive to localize the program.

5. Keep the text out of Code:
It means that all the text strings, error messages that could be possibly translated should be stored in a separate file independent of the source code.

6. Extended Characters:
Handling of extended characters is a common problem with localized software and even non localized software. The extended characters may include the symbol which may not be on your typical keyboard.

7. Computations on Characters:
The different examples are word sorting and uppercase and lower case conversion. Carefully look at your software whether it performs spell check in a proper way .If the sorting is to be performed what are the sorting rules.