ST Assignment No.1

Testing can be defined in simple words as "Performing Verification and Validation of the Software Product" for its correctness and accuracyof working.

1. Software testing is as old as the hills in the history of digital computers.

2. The testing of software is an important means of assessing the software to determine its quality.

3. Since testing typically consumes 40% to50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering.

4. With the development of fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased.

5. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made.

Q1. List the error case studies and explain any 3 of them in details.

Ans.

The various error case studies related to software testing are

- *Disney Lion King*
- *Intel Pentium Floating Point Division Bug*
- *NASA Mars Polar Lander*
- *Y2K Bug*

The Disney Lion King case study identifies the problem that the software product which is developed should be compatible with the various configurations and the hardware compatibility should be tested on the various machines. The product developed by the company should be given for testing to various clients for testing.

*The Intel Pentium floating point division bug occurs in the rare case when we try to develop high end mathematical project. Enter the following equation into your PC's calculator :(4195835 / 3145727) * 3145727 – 4195835. If it doesn't give the error its fine, but in few older machines it shows the floating point division error. But stakes the reputation of the company as it cannot solve the problem and monetary loss to the company.*

The NASA Mars Polar Lander failed as each module that was designed for the launch was tested separately and functioned perfectly. But the various modules thatwere designed were never integrated and tested together. The switch installed to check the landing of the mars polar lander was tested separately and whenintegrated, the bit changed to 1 by the slight jerk caused all lander to fail.

The most common the Y2K bug was the recent one which caused lot of computers to give error as the year changed from 31.12.1999 to 1.1.2000 and the software's which were developed to take the last two digits of the years turned to 00. The changes in the *software’s were not possible as the developer who developed it eitherleft the company and the development cost was too high.*

Q2. Define Bug and list the terms that are related to software failure.

Ans.

A bug can be defined in simple term as any error or mistake that leads tothe failure of the product or software either due to the specification problem or due to communication problem, regarding what is developed and what had to be developed.

- ***Terms for software failure***

The various terms related to software failure with respect to the area of applicationare listed as Defect, Variance, Fault, Failure, Problem, Inconsistency, Error, Feature, Incident, Bug, and Anomaly.

1. Problem, error, and bug are probably the most generic terms used.

2. Anomaly, incident, and variance don’t sound quite so negative and infer more unintended operation than an all-out failure.

3. Fault, failure, and defect tend to imply a condition that’s really severe, maybe even dangerous. It doesn’t sound right to call an incorrectly colored icon a fault. These words also tend to imply blame: “It’s his fault that the software failed.

4. As all the words sound the same they are distinguished based on the severity andthe area in which the software failure has occurred.

5. When we run a program the error that we get during execution is termed on the basis of runtime error, compile time error, computational error, and assignment error.

6. The error can be removed by debugging, if not resolved leads to a problem and ifthe problem becomes large leads to software failure.

7. A bug can be defined as the initiation of error or a problem due to which fault, failure, incident or an anomaly occurs.

8. The program to find the factorial of a number given below lists few errors problem and failure in a program.

Q3. What are different factors that lead for bugs to occur in project.

Ans.

A bug can be defined in simple term as any error or mistake that leads tothe failure of the product or software either due to the specification problem or due to communication problem, regarding what is developed and what had to be developed.

A software bug occurs when one or more of the following factors are true:

1. *The software doesn't do something that the product specification says it shoulddo.*
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.*

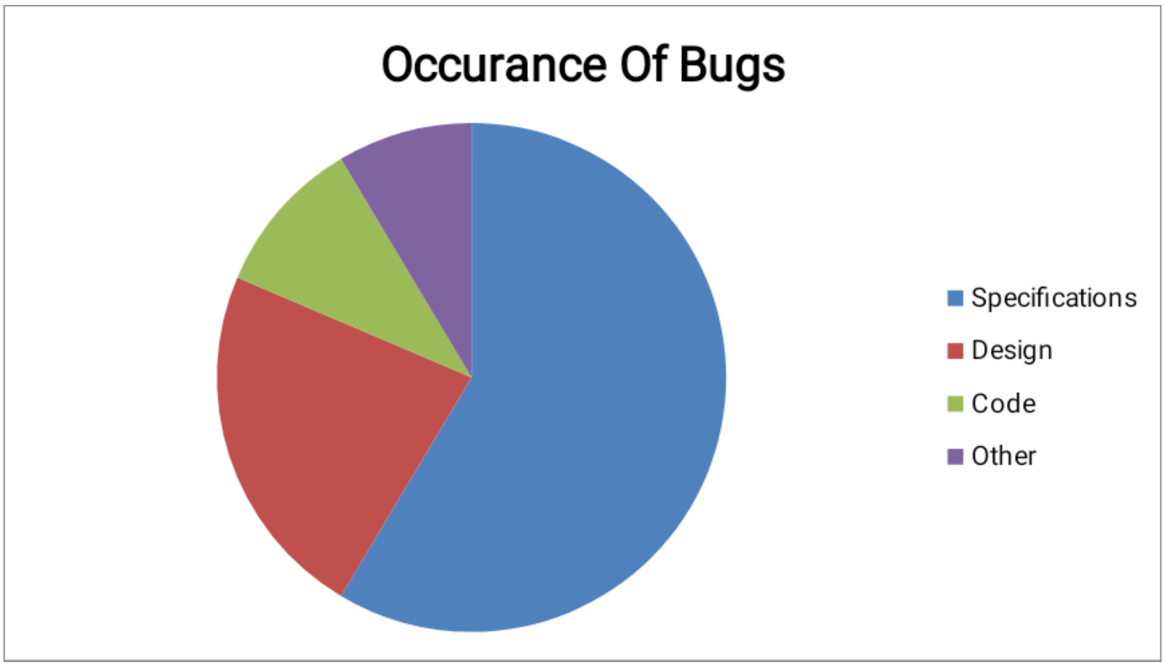
Q4.Why do bug occur in a project.

Ans.

Bugs are a mismatch between expected result and actual result. Bugs play animportant role in software testing.

In large projects bugs occur due to various reasons.

1. *One of the extreme causes is the specification.*
2. *Specifications are the largest producer of bugs.*
3. *Either specifications are not written, specifications are not thorough enough, constantly changing or not communicated well to the development team.*
4. *Another bigger reason is that software is always created by human beings.*
5. *They know numerous things but are not expert and might make mistakes.*
6. *Further, there are deadlines to deliver the project on time. So increasing pressure and workload conduct in no time to check, compromise on quality and incomplete systems. So this leads to occurrence of Bugs.*



Q5. What is cost of Bug in software testing?

Ans.

1. If the error is made and the consequent defect is detected in the requirements phase then it is relatively cheap to fix it.

2. If an error is made and the consequent defect is found in the design phase or in construction phase then the design can be corrected and reissued with relatively little expense.

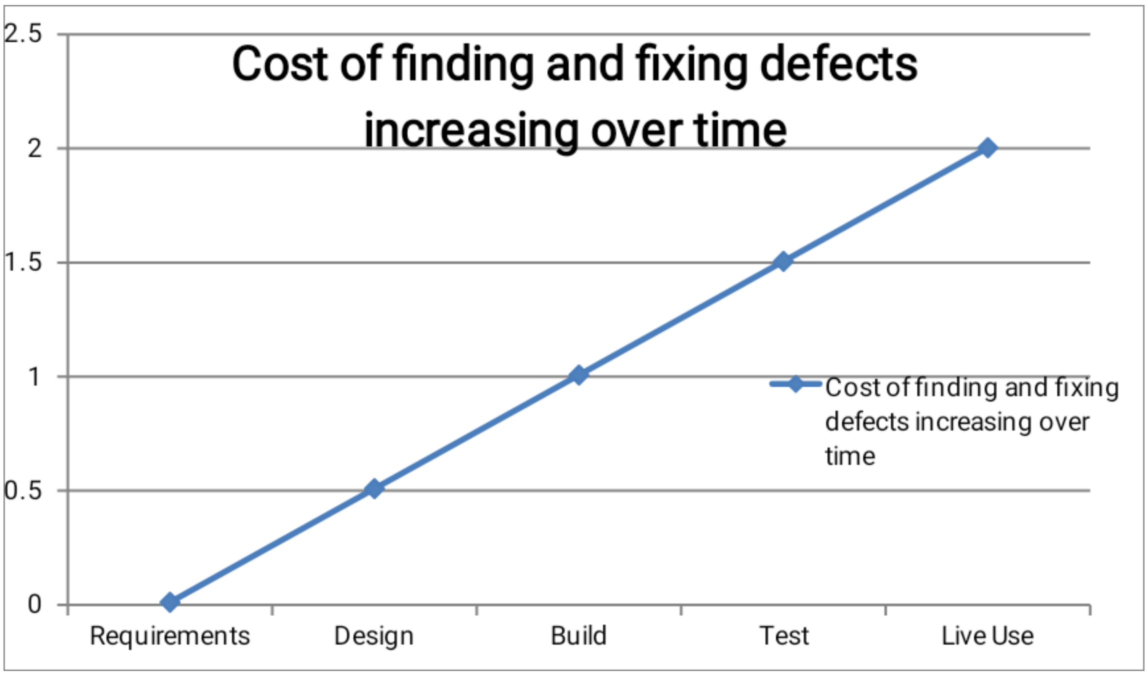


Figure 1.2

3. If a defect is introduced in the requirement specification and it is not detected until acceptance testing or even once the system has been implemented then it will be much more expensive to fix.

4. This is because rework will be needed in the specification and design before changes can be made in construction; because one defect in the requirements may well propagate into several places in the design and code.

5. All the testing work done-to that point will need to be repeated in order to reachthe confidence level in the software that we require.

6. It is quite often the case that defects detected at a very late stage, depending on how serious they are, are not corrected because the cost of doing so is too expensive.

Q6. Explain the traits/qualities of good software tester.

Ans.

List of traits that most software testers should have:

1. *They are explorers: - Software testers aren't afraid to venture into unknownsituations. 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. As they find the bugs it is their responsibility to givethe solution also.*
3. *They are relentless: - Software testers keep trying. They may see a bug thatquickly 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.Their perception of finding the bug is more than that of a programmer.*
5. *They are perfectionists: - They strive for perfection, but they know when it becomes unattainable and they want to be close as they can to the solution.*
6. *They exercise good judgment: - Software testers need to make decisionsabout what they will test, how long it will take, and if the problem they're looking at is really a bug. The job of a software tester is in time constraints.They have to find the bugs as early as possible.*
7. *They are tactful and diplomatic: - Software testers are always the bearersof bad news. They have to tell the programmers about the various bugs andthe error that exist in their program and it has to be told carefully. Good software testers know how to do so tactfully and professionally and know how to work with programmers who aren't always tactful and diplomatic.*
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 followingthrough on making it happen.*

Q7. Explain Waterfall and Spiral Model.

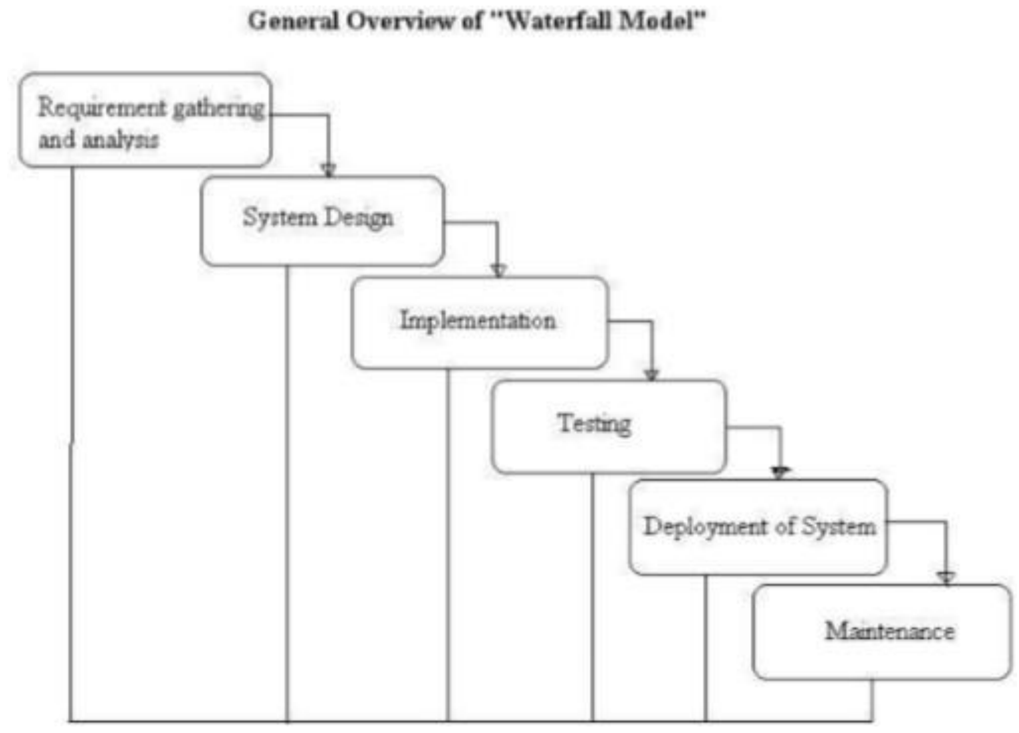
Ans.

a. **Waterfall Model:**

   1. The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall)

   2. The phases are Requirement Gathering and Analysis, System Design, Implementation, Testing, Deployment of System and Maintenance.

Diagram of Waterfall-model:



General Overview of "Waterfall Model"
Requirement gathering and analysis
System Design
Implementation
Testing
Deployment of System
Maintenance

Figure 1.6

Advantages of waterfall model:

1. Simple and easy to understand and use.

2. Easy to manage due to the rigidity of the model, each phase has specific deliverables and a review process.

3. Phases are processed and completed one at a time.

4. Works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

1. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

2. No working software is produced until late during the life cycle.

3. High amounts of risk and uncertainty.

4. Not a good model for complex and object-oriented projects.

5. Poor model for long and on-going projects.

6. Not suitable for the projects where requirements are at a moderate to high risk of changing.

When to use the waterfall model:

1. Requirements are very well known, clear and fixed.

2. Product definition is stable.

3. Technology is understood.

4. There are no ambiguous requirements

5. Ample resources with required expertise are available freely

6. The project is short.

b. Spiral Model:

1. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.

2. A software project repeatedly passes through these phases in iterations.

3. The baseline spiral starting in the planning phase, requirements are gathered and risk is assessed.

4. The general idea behind the spiral model is that you don’t define everythingin detail at the very beginning.

5. Start small, define your important features, try them out, get feedback from your customers, and then move on to the next level.

6. Repeat this until you have your final product.

Diagram of Spiral Model:

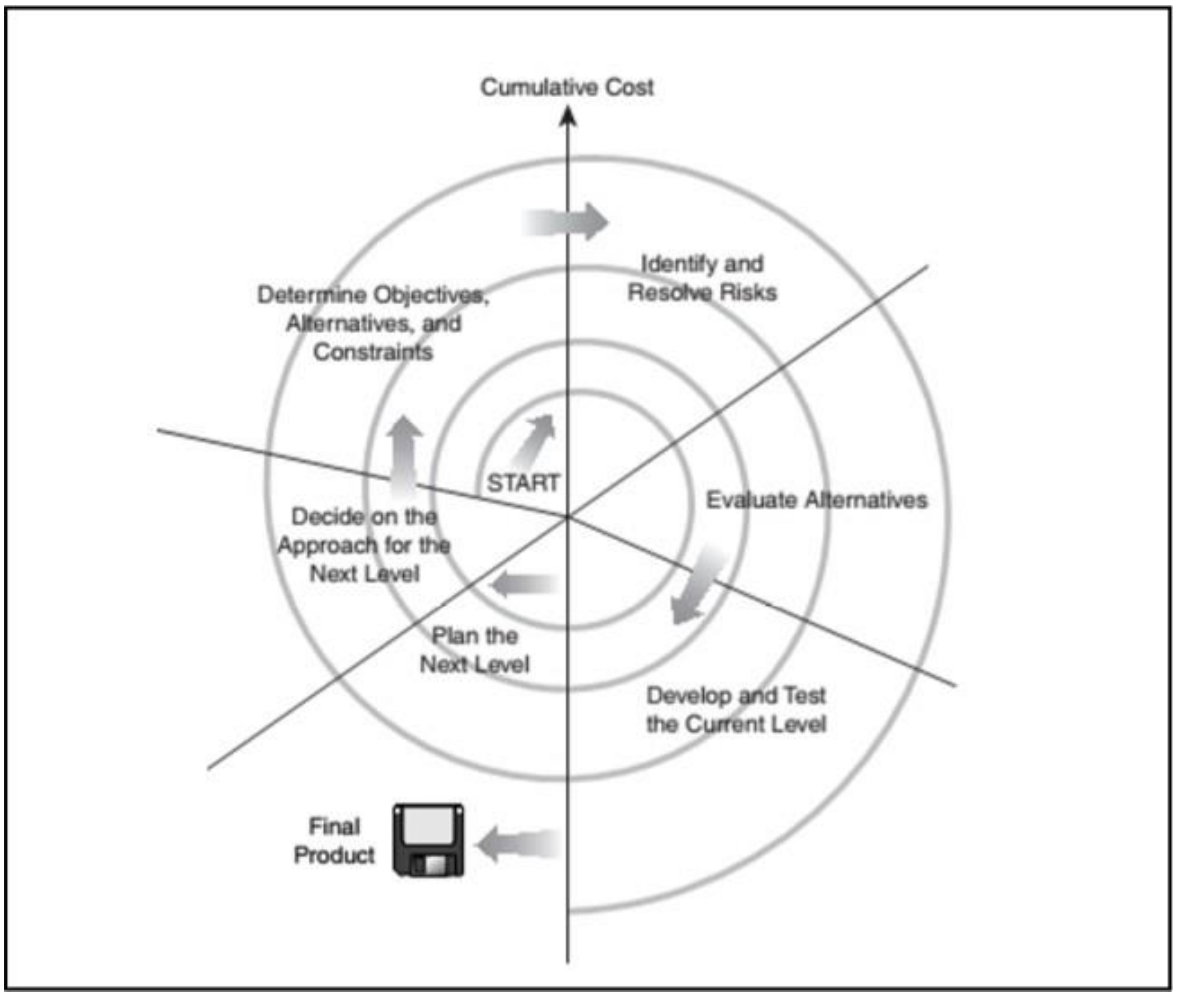

Cumulative Cost
Identify and Resolve Risks
Determine Objectives, Alternatives, and Constraints
START
Evaluate Alternatives
Decide on the Approach for the Next Level
Plan the Next Level
Develop and Test the Current Level
Final Product

Figure 1.7

Advantages of Spiral model:

1. *High amount of risk analysis hence, avoidance of Risk is enhanced.*
2. *Good for large and mission-critical projects.*
3. *Strong approval and documentation control.*
4. *Additional Functionality can be added at a later date.*
5. *Software is produced early in the software life cycle.*

Disadvantages of Spiral model:

1. *Can be a costly model to use.*
2. *Risk analysis requires highly specific expertise.*
3. *Project's success is highly dependent on the risk analysis phase.*

4. Doesn't work well for smaller projects.

When to use Spiral model:

1. When costs and risk evaluation is important

2. For medium to high-risk projects

3. Long-term project commitment unwise because of potential changes toeconomic priorities

4. Users are unsure of their needs

5. Requirements are complex

6. New product line

7. Significant changes are expected (research and exploration)

Q8. What is Software Testing.

Ans.

Software testing *is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.*

ST Assignment No.2

Q1. Describe Black Box Testing & white box testing.

Ans.

- ***Black Box Testing***

1. This approach tests all possible combinations of end-user actions.

2. Black box testing assumes no knowledge of code and is intended to simulate the end-user experience.

3. The tester can use sample applications to integrate and test the application blockfor black box testing.

4. Planning for black box testing immediately after the requirements and the functional specifications are available.

5. Black box testing should be conducted in a test environment close to the target environment.

- ***White box testing***

1. This is also known as glass box, clear box, and open box testing.

2. In white box testing, test cases are created by looking at the code to detect any potential failure scenarios.

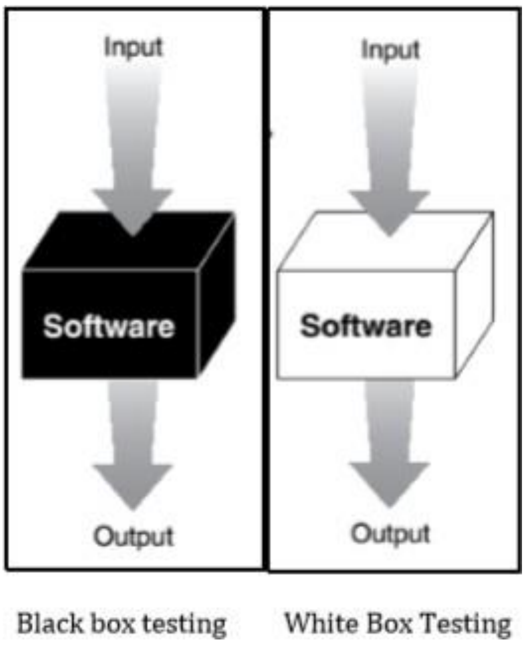
3. The suitable input data for testing various APIs and the special code paths that need to be tested by analysing the source code for the application block.

4. Therefore, the test plans need to be updated before starting white box testing and only after a stable build of the code is available.

5. White box testing assumes that the tester can take a look at the code for the application block and create test cases that look for any potential failure scenarios.

6. During white box testing, analyze the code of the application block and prepare test cases for testing the functionality to ensure that the class is behaving in accordance with the specifications and testing for robustness.

7. A failure of a white box test may result in a change that requires all black box testing to be repeated and white box testing paths to be reviewed and possibly changed.



Q2. Explain static testing and dynamic testing.

Ans.

1. *Dynamic testing is testing that is performed when the system is running.*
2. *The basic requirement is to review test plans.*
3. *Recommend tests based on the hazard analyses, safety standards and checklists, previous accident and incidents, operator task analyses etc.*
4. *Specify the conditions under which the test will be conducted.*
5. *Review the test results for any safety-related problems that were missed in the analysis or in any other testing.*
6. *Ensure that the testing feedback is integrated into the safety reviews and analyses that will be used in design modifications*
7. *Static testing is performed when the system is not running.*
8. *Static testing works with peer review, and mostly referred to as pen and pencil run.*

Q3. Explain the concept of data testing in dynamic black box testing.

Ans.

1. *The simplest view of software is to divide its world into two parts: the data and the program.*
2. *The data is the keyboard input, mouse clicks, disk files, printouts, and so on.*

3. The program is the executable flow, transitions, logic, and computations.

4. When software testing is performed on the data, the user information is checked and the data is tabulated with the expected results.
Examples of data would be:

•The words you type into a word processor

•The numbers entered into a spreadsheet

•The number of shots you have remaining in your space game

•The picture printed by your photo software

•The backup files stored on your floppy disk
•The data being sent by your modem over the phone lines

Q4. Explain the concept of boundary conditions and sub boundary conditions in dynamic black box testing.

Ans.

a) ***Boundary Condition***

1. Boundary conditions are special because programming, by its nature, is susceptible to problems at its edges.

2. The boundary conditions are defined as the initial and the final data ranges of the variables declared.

3. If an operation is performed on a range of numbers, odds are the programmer gotit right for the vast majority of the numbers in the middle, but maybe made a mistakeat the edges.

4. The edges are the minimum and the maximum values for that identifier.

For example

1. #include<stdio.h>

2. void main()

3. {

4. int i , fact=1, n;

5. printf("enter the number ");

6. scanf("%d",&n); 7.

for(i =1 ;i <=n;i++)

8. fact = fact * i;

9. printf (“the factorial of a number is ▫”%d”, fact);

10. }

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

b) Sub-Boundary Conditions

1. *They’re the ones defined in the specification or evident when using the software.*
2. *Some boundaries, though, that are internal to the software aren’t necessarily apparent to an end user but still need to be checked by the software tester.*
3. *These are known as sub-boundary conditions or internal boundary conditions.*
4. *In the given example the sub boundary condition is the value of factorial*

For example

1. #include<stdio.h>

2. void

main()3. {

4. int i , fact=1, n;

5. printf(“enter the number “);

6. scanf(“%d”,&n);

7. for(i =1 ;i

<=n;i++)

8. fact = fact * i;

9. printf (“the factorial of a number is ▫”%d”, fact);

10. }

Q5. Explain the concept of state testing in dynamic black box testing.

Ans.

1. *The data gets tested on the numbers, words, inputs, and outputs of the software.*
2. *The product or the software should be tested for the program’s logic flow through*

its various states.

3. A software state is a condition or mode that the software is currently in.

4. Consider Figures 2.2, which illustrates the software mode of the paint software in airbrush mode.

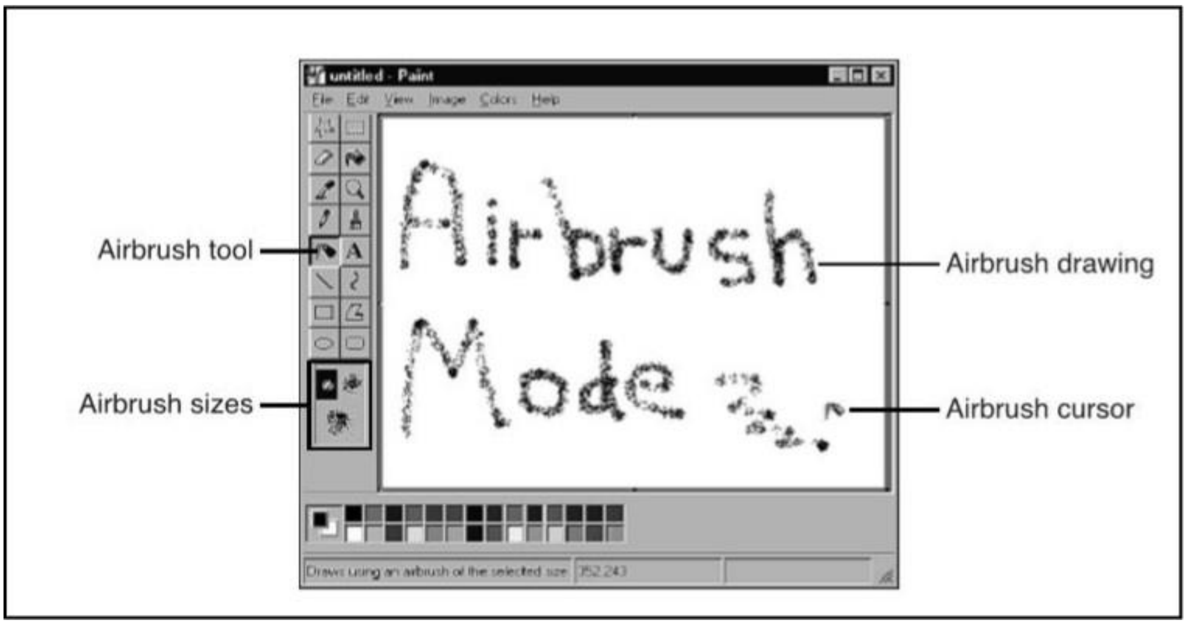


Figure 2.2
a)

Testing Software's Logic Flow

1. Testing the software's states and logic flow has the same problems. It's usually possible to visit

2. Check for the all possible states for test to pass condition

3. The complexity of the software, especially due to the richness of today's user interfaces, provides so many choices and options that the number of paths grows exponentially.

Q6. Define in short: Repetition testing , stress, load.

Ans.

1. *The software fails under are repetition, stress, and load.*
2. *These tests target state handling problems where the programmer didn't consider what might happen in the worst-case scenarios.*
3. *Repetition testing involves doing the same operation over and over.*
4. *This could be as simple as starting up and shutting down the program over*

and over.

5. It could also mean repeatedly saving and loading data or repeatedly selecting the same operation.

6. A bug can be encountered after only a couple repetitions or it might take thousands of attempts to reveal a problem.

7. The repetition testing is done to look for memory leaks.

8. A common software problem happens when computer memory is allocated to perform a certain operation but isn't completely freed when the operation completes.

9. The result is that eventually the program uses up memory that it depends on to work reliably.

10. Stress testing is running the software under less-than-ideal conditions such as low memory, low disk space, slow CPUs, slow modems.

11. The software should be tested for external resources and dependencies it has on them.

12. Stress testing is simply limiting them to their bare minimum.

Q7. Explain concept of creating state transition map.
Ans.

1. There are various techniques for state transition diagrams.

2. Figure 2.3 shows two examples. One uses boxes and arrows and the other uses circles (bubbles) and arrows.

3. The sate transition diagram should be easily understandable and should be able to explain clearly about the various stages the software passes through.

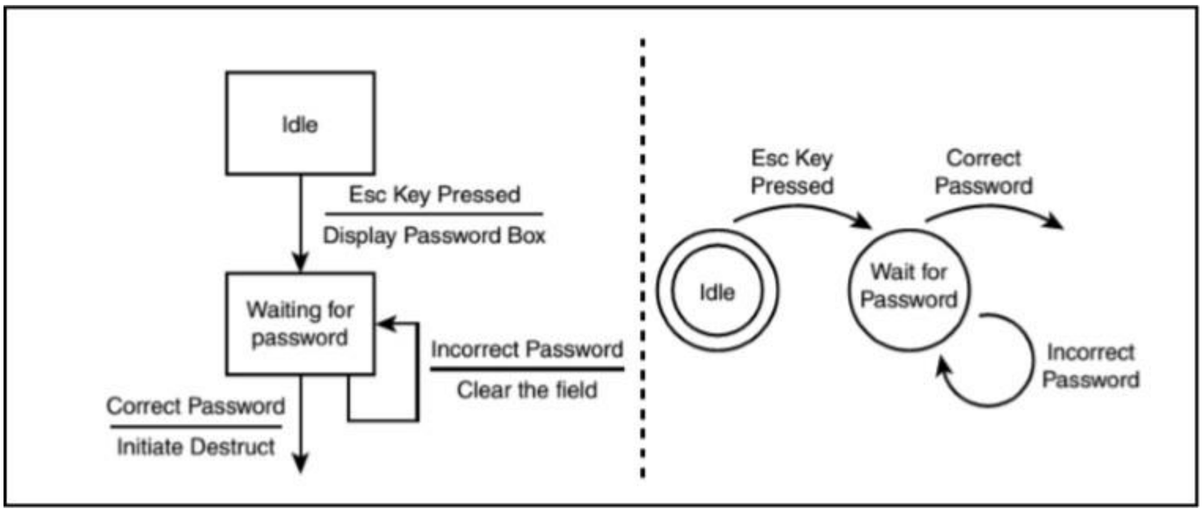


Figure 2.3

A state transition map should show the following items:

1. *Each unique state that the software can be in.*
2. *A good rule of thumb is that if the state is unsure whether something is a separate state, it probably is.*
3. *Always collapse it into another state if you find out later that it isn't.*
4. *The input or condition that takes it from one state to the next.*
5. *This might be a key press, a menu selection, a sensor input, a telephone ring, and so on.*
6. *A state can't be exited without some reason. The specific reason is what you're looking for here.*
7. *Set conditions and produced output when a state is entered or exited.*
8. *This would include a menu and buttons being displayed, a flag being set, a printout occurring, a calculation being performed, and so on.*
9. *It's anything and everything that happens on the transition from one state to the next.*

ST Assignment No.3

Q1. Explain Formal Review in static white box testing.

Ans.

1. A formal review is the process under which static white-box testing is performed.

2. A formal review can range from a simple meeting between two programmers to a detailed, rigorous inspection of the code.

- *There are four essential elements to a formal review*

1. Identify Problems: - .The goal of the review is to find problems with the software—not just items that are wrong, but missing items as well. All criticism should be directed at the code, not the person who created it. Participants shouldn't take any criticism personally. Leave your egos, emotions, and sensitive feelings at the door.

2. Follow Rules: - A fixed set of rules should be followed. They may set the amount of code to be reviewed (usually a couple hundred lines), how much time will be spent (a couple hours), what can be commented on, and so on. This is important so that the participants know what their roles are and what they should expect. It helps the review run more smoothly.

3. Prepare: - Each participant is expected to prepare for and contribute to the review. Depending on the type of review, participants may have different roles. They need to know what their duties and responsibilities are and be ready to actively fulfil them at the review. Most of the problems found through the review process are found during preparation, not at the actual review.

4. Write a Report: - The review group must produce a written report summarizing the results of the review and make that report available to the rest of the product development team. It's imperative that others are told the results of the meeting—how many problems were found, where they were found, and so on.

Q2. Explain concept of Peer Reviews, Walkthroughs, Inspections.

Ans.

a) ***Peer Reviews***

1. The easiest way to get team members together and doing their first formal reviews of the software is through peer reviews, the least formal method.

2. Sometimes called buddy reviews, this method is really more of a discussion.

3. Peer reviews are often held with just the programmer who wrote the code and one or two other programmers or testers acting as reviewers.

4. Small group simply reviews the code together and looks for problems and oversights.

5. To assure that the review is highly effective all the participants need to make sure that the four key elements of a formal review are in place: Look for problems, follow rules, prepare for the review, and write a report.

6. As peer reviews are informal, these elements are often scaled back. Still, just getting together to discuss the code can find bugs.

b) ***Walkthroughs***

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

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

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

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

c) ***Inspections***

1. *Inspections are the most formal type of reviews.*
2. *They are highly structured and require training for each participant.*
3. *Inspections are different from peer reviews and walkthroughs in that the person who presents the code, the presenter or reader, isn't the original programmer.*
4. *These forces someone else to learn and understand the material being presented,potentially giving a different slant and interpretation at the inspection meeting.*
5. *The other participants are called inspectors.*
6. *Each is tasked with reviewing the code from a different perspective, such as auser, a tester, or a product support person.*
7. *This helps bring different views of the product under review and very oftenidentifies different bugs.*
8. *One inspector is even tasked with reviewing the code backward—that is, from theend to the beginning—to make sure that the material is covered evenly and completely.*

Q3. Explain data declaration errors & computational erros.

Ans.

1. ***Data Declaration Errors***

Data declaration bugs are caused by improperly declaring or using variables or constants.

1. Are all the variables assigned the correct length, type, and storage class? For example, should a variable be declared as a string instead of an array of characters?

2. If a variable is initialized at the same time as it's declared, is it properly initialized and consistent with its type?

3. Are there any variables with similar names? This isn't necessarily a bug, but it could be a sign that the names have been confused with those from somewhere else in the program.

4. Are any variables declared that are never referenced or are referenced only once?

5. Are all the variables explicitly declared within their specific module? If not, is it understood that the variable is shared with the next higher module?

2. Computation Errors

Computational or calculation errors are essentially bad math. The calculations don't result in the expected result.

1. Do any calculations that use variables have different data types, such as adding an integer to a floating-point number?

2. Do any calculations that use variables have the same data type but are different lengths—adding a byte to a word, for example?

3. Are the compiler's conversion rules for variables of inconsistent type or length understood and taken into account in any calculations?

4. Is the target variable of an assignment smaller than the right-hand expression?

5. Is overflow or underflow in the middle of a numeric calculation possible?

6. Is it ever possible for a divisor/modulus to be zero?

7. For cases of integer arithmetic, does the code handle that some calculations, particularly division, will result in loss of precision?

8. 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%?

Q4. Differentiate between dynamic white box testing and debugging.

Ans.

1. It's important not to confuse dynamic white-box testing with debugging.

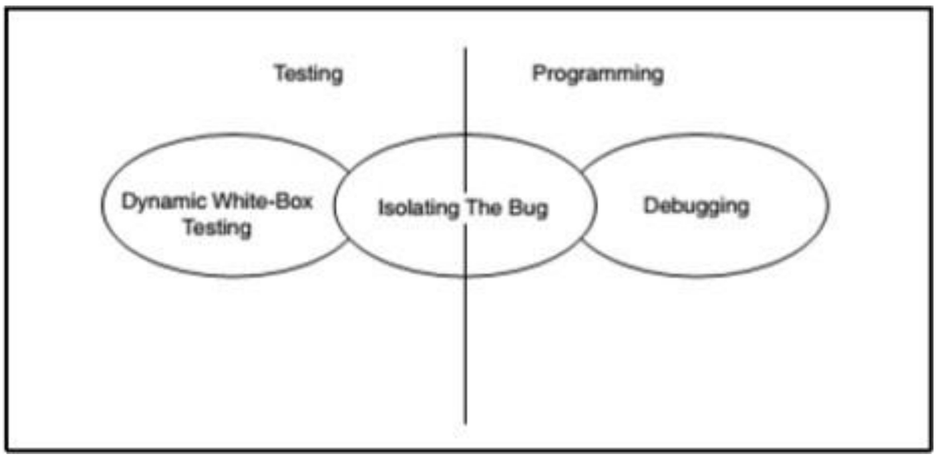
2. The goal of dynamic white-box testing is to find bugs. The goal of debugging is to fix them.

3. They do overlap, however, in the area of isolating where and why the bug occurs. You'll learn more about this later about, "Reporting What You Find," but for now, think of the overlap this way.

4. As a software tester, you should narrow down the problem to the simplest test case that demonstrates the bug.

5. If its white-box testing, that could even include information about what lines of code look suspicious.

6. The programmer who does the debugging picks the process up from there, determines exactly what is causing the bug, and attempts to fix it.



Q5. Explain Unit and Integration Testing in SDFC/ V-Model illustrating with suitable block diagram.

Ans.

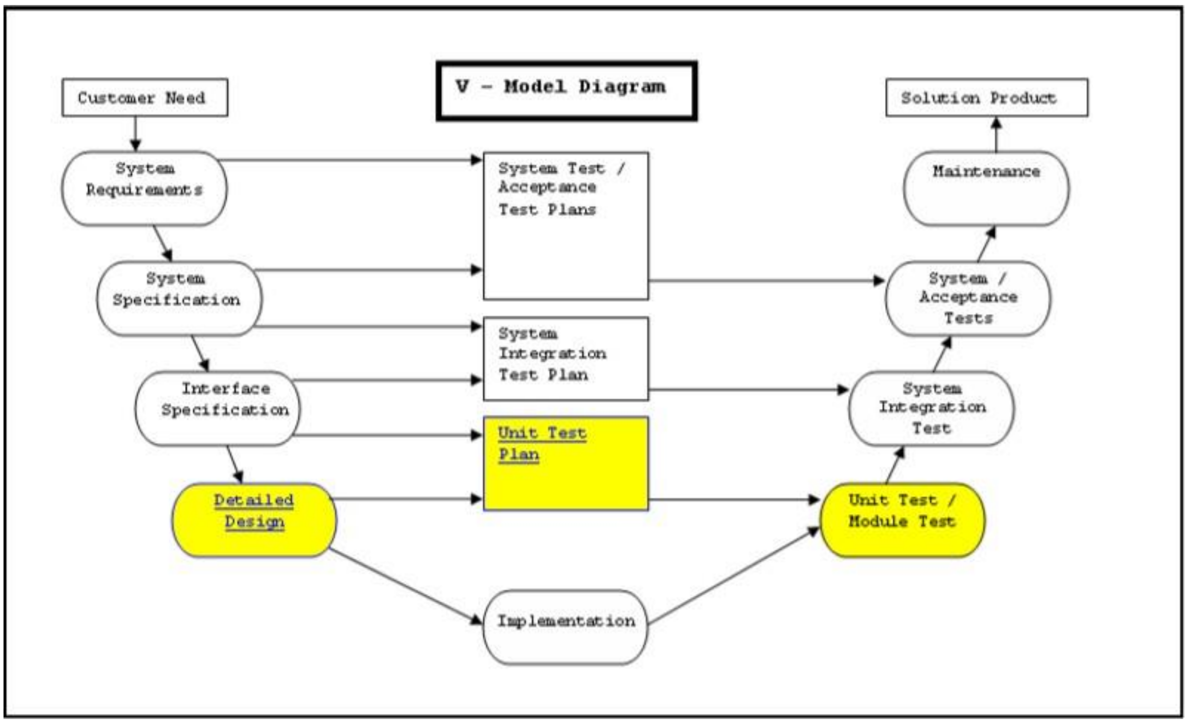
- *Unit and Integration Testing*

    1. *As a large project or an application software is divided into small modules* or units to reduce the complexity and to minimize the failure rate of thesoftware.

    2. *Unit testing tests all the modules separately for the functionality and the the correctness of each module.*

3. The integration testing is than implemented for the following reasons

- *To test that the output of one unit or module when given as an input to the second module does not affect the output and the correctness of the integrated module.*
- *The change made to one module gives the effective and correct output of the integrated module and software as a product does not fail.*
- *The parameters of one module match the parameters of the other module with respect to the permissible values, boundary conditions, correctness and utilization.*
- *The figure shows the concept of unit testing and integration testing in V model.*
- *There are two basic types of integration (i) top down integration (ii) bottom up integration.*



Q6. Describe comparison errors and control flow errors.

Ans.

- ***Comparison Errors***

Comparison and decision errors are very susceptible to boundary conditionproblems.

1. Are the comparisons correct? It may sound pretty simple, but there's always confusion over whether a comparison should be less than or less than or equal to.

2. Are there comparisons between fractional or floating-point values? If so, will any precision problems affect their comparison? Is 1.00000001 close enough to 1.00000002 to be equal?

3. Does each Boolean expression state what it should state? Does the Boolean calculation work as expected? Is there any doubt about the order of evaluation?

4. Are the operands of a Boolean operator Boolean? For example, is an integer variable containing integer values being used in a Boolean calculation?

- ***Control Flow Errors***

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

1. They are usually caused, directly or indirectly, by computational or comparison errors.

2. If the language contains statement groups such as begin...end and do...while, are the ends explicit and do they match their appropriate groups?

3. Will the program, module, subroutine, or loop eventually terminate? If it won't, is that acceptable?

4. Is there a possibility of premature loop exit?

5. Is it possible that a loop never executes? Is it acceptable if it doesn't?

6. If the program contains a multi way branch such as a switch...case statement, can the index variable ever exceed the number of branch possibilities?

Q7. Explain the following:

a. Error Forcing.
Ans.

1. The last type of data testing covered in this chapter is error forcing.

2. Testing the software in a debugger has the ability to watch variables and see what values they hold.

3. In the preceding compound interest calculation, if you couldn't find a direct way to set the number of compounding (n) to zero, you could use your debugger to force it to zero.

4. The software would then have to handle it or not.

b. Code Coverage.

Ans.

1. Black-box testing, testing the data is only half the battle.

2. For comprehensive coverage you must also test the program's states and the program's flow among them.

3. You must attempt to enter and exit every module, execute every line of code, and follow every logic and decision path through the software.

4. Examining the software at this level of detail is called Code-coverage analysis is a dynamic white-box testing technique because it requires you to have full access to the code to view what parts of the software you pass through when you run your test cases.

c. Line Coverage.

Ans.

1. The most straightforward form of code coverage is called statement coverage or line coverage.

2. If you're monitoring statement coverage while you test your software, your goal is to make sure that you execute every statement in the program at least once.

3. With line coverage the tester tests the code line by line giving the relevant output.

For example

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

d. Branch and Condition Coverage.
Ans.

- ***Branch Coverage***

1. *Attempting to cover all the paths in the software is called path testing.*
2. *The simplest form of path testing is called branch coverage testing.*
3. *To check all the possibilities of the boundary and the sub boundary conditions and it's branching on those values.*
4. *Test coverage criteria requires enough test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry to a program or subroutine is invoked at least once.*
5. *Every branch (decision) taken each way, true and false.*
6. *It helps in validating all the branches in the code making sure that no branch leads to abnormal behaviour of the application.*

- ***Condition Coverage***

1. Just when you thought you had it all figured out, there's yet another complicationto path testing.

2. Condition coverage testing takes the extra conditions on the branch statements into account.

ST Assignment No. 4

Q1. Explain the concept of Stubs and Drivers in Unit Testing?

Ans.

1. *Unit is the smallest testable part of the software system.*

2. *Unit testing is done to verify that the lowest independent entities in any softwareare working fine.*

3. *The smallest testable part is isolated from the remainder code and tested to determine whether it works correctly.*

4. *When developer is coding the software it may happen that the dependent modulesare not completed for testing, in such cases developers use stubs and drivers to simulate the called (stub) and caller (driver) units.*

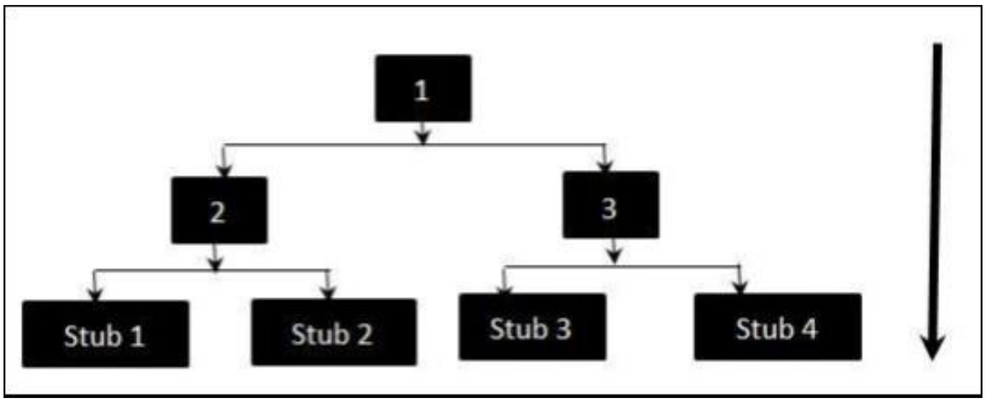
5. *Unit testing requires stubs and drivers, stubs simulates the called unit and driver simulates the calling unit.*

1. STUBS:

i. Assume you have 3 modules, Module A, Module B and module C.

ii. Module A is ready and we need to test it, but module A calls functions from Module B and C which are not ready, so developer will write a dummy module which simulates B and C and returns values to module A.

iii. This dummy module code is known as stub.



iv. The above diagrams clearly states that Modules 1, 2 and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time.

v. Hence, Stubs are used to test the modules.

2. DRIVERS:

i. Now suppose you have modules B and C ready but module A which calls functionsfrom module B and C is not ready so developer will write a dummy piece of code formodule A which will return values to module B and C.

ii. This dummy piece of code is known as driver.

Q2. Explain Top-Down and Bottom-Up integration testing technique with advantages and disadvantages.

Ans.

*i. **Top down Testing**:*

In this approach testing is conducted from main module to sub module.

ii. If the sub module is not developed a temporary program called STUB is used for simulate the sub module.

***Advantages**:*

- *Advantageous if major flaws occur toward the top of the program.*
- *Once the I/O functions are added, representation of test cases is easier.*
- *Early skeletal Program allows demonstrations and boosts morale.*

Disadvantages:

- *Stub modules must be produced*
- *Stub Modules are often more complicated than they first appear to be.*
- *Before the I/O functions are added, representation of test cases in stubs canbe difficult.*
- *Test conditions may be impossible, or very difficult, to create.*
- *Observation of test output is more difficult. ▫ Allows one to think that designand testing can be overlapped.*
- *Induces one to defer completion of the testing of certain modules.*

i. Bottom up testing:

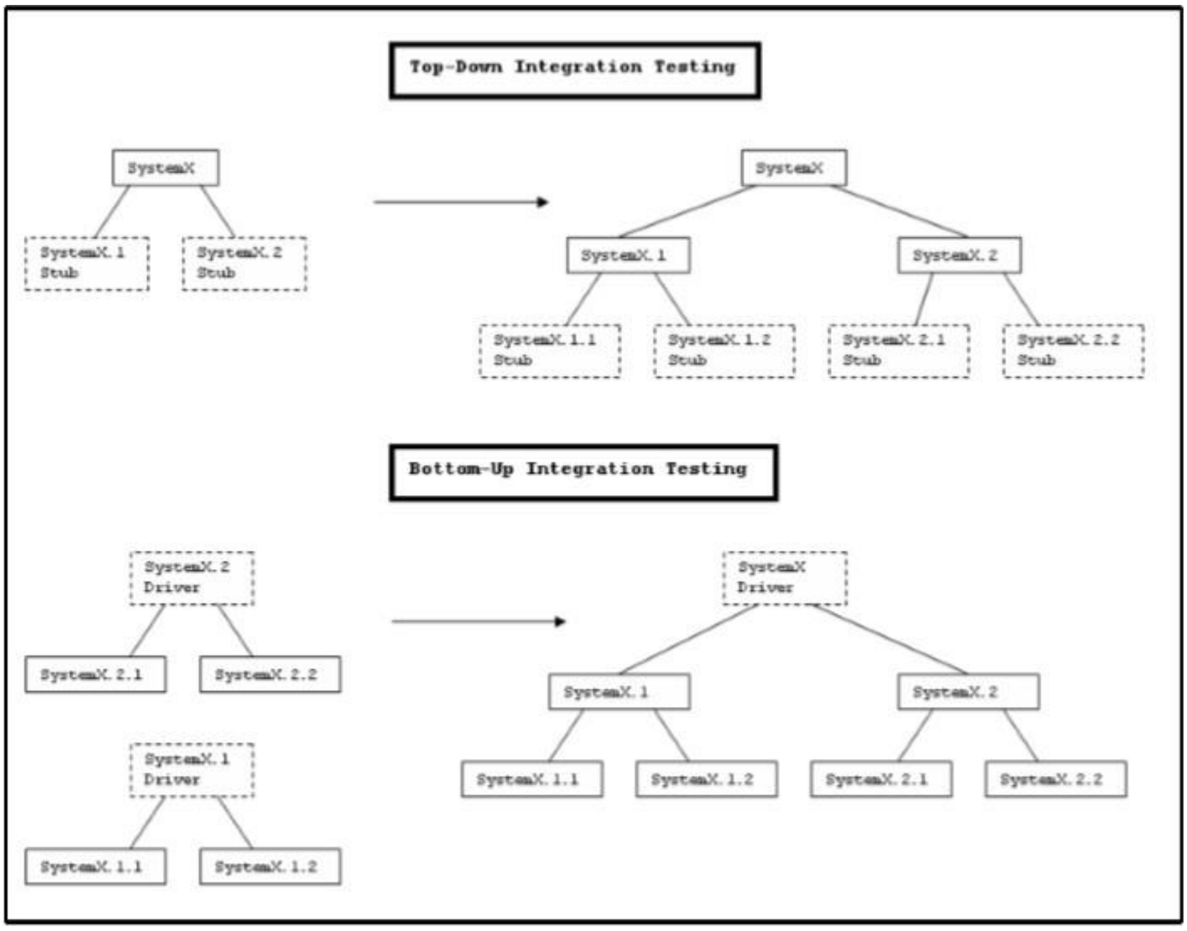
In this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulatethe main module.

Advantages:

- *Advantageous if major flaws occur toward the bottom of the program.*
- *Test conditions are easier to create. ▫ Observation of test results is easier.*

Disadvantages:

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



Q3. Describe incremental integration testing with features.
Ans.

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

Features

i. Each Module provides a definitive role to play in the project/product structure

ii. Each Module has clearly defined dependencies some of which can be known only at the runtime.

iii. The incremental integration testing's greater advantage is that thedefects are found early in a smaller assembly when it is relatively easy to detect the root cause of the same.

iv. A disadvantage is that it can be time-consuming since stubs and drivers have to be developed for performing these tests.

Q4. Explain System Testing in details and explain any 4 system testing approaches.
Ans.

1. *System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements.*
2. *In System testing, the functionalities of the system are tested from an end-to-endperspective.*
3. *System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased.*
4. *It includes both functional and Non-Functional testing.*

1. Recovery Testing

i. Recovery testing is a type of non-functional testing technique performed in order to determine how quickly the system can recover after it has gone through system crash or hardware failure.

ii. Recovery testing is the forced failure of the software to verify if the recovery is successful.

Steps:

- *Determining the feasibility of the recovery process.*
- *Verification of the backup facilities.*
- *Ensuring proper steps are documented to verify the compatibility of backup facilities.*
- *Providing Training within the team.*
- *Demonstrating the ability of the organization to recover from all critical failures.*
- *Maintaining and updating the recovery plan at regular intervals.*

2. Security Testing

i. Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended.

ii. It also aims at verifying 6 basic principles as listed below:

- *Confidentiality*
- *Integrity*
- *Authentication*
- *Authorization*
- *Availability*
- *Non-repudiation*

Techniques:

- *Injection*
- *Broken Authentication and Session Management*
- *Cross-Site Scripting (XSS)*
- *Insecure Direct Object References*
- *Security Misconfiguration*

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

3. Load Testing.

i. Load testing is performance testing technique using which the response of the system is measured under various load conditions.

ii. *The load testing is performed for normal and peak load conditions.*

Load Testing Approach:

- *Evaluate performance acceptance criteria*
- *Identify critical scenarios*
- *Design workload Model*
- *Identify the target load levels*
- *Design the tests*
- *Execute Tests*
- *Analyse the Results*

Objectives of Load Testing:

- *Response time*
- *Throughput*
- *Resource utilization*
- *Maximum user load*
- *Business-related metrics*

4. Compatibility Testing.

i. Compatibility testing is a non-functional testing conducted on the *application to evaluate the application's compatibility within different environments.*

ii. *It can be of two types - forward compatibility testing and backward compatibility testing.*

- *Operating system Compatibility Testing - Linux , Mac OS, Windows*
- *Database Compatibility Testing - Oracle SQL Server*
- *Browser Compatibility Testing - IE , Chrome, Firefox*
- *Other System Software - Web server, networking/ messaging tool, etc*

Q5. Explain criteria for acceptance testing.

Ans.

i. *Comparison testing comprises of comparing the contents of files, databases, against actual results.*

ii. *They are capable of highlighting the differences between expected andactual results.*

iii. *Comparison test tools often have functions that allow specified sectionsof the files be ignored or masked out.*

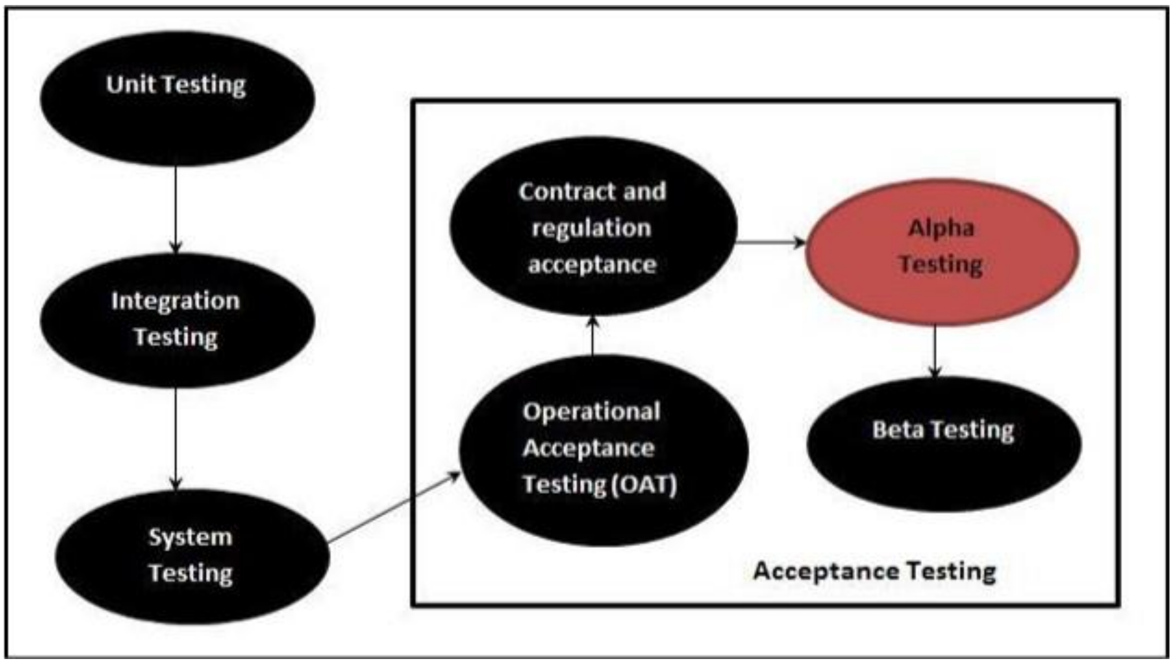
iv. *This enables the tester to mask out the date or time stamp on a screen orfield as it is always different from the expected ones when a comparison is performed.*

Q6. Explain Alpha Testing and Beta Testing.

Ans. *Alpha testing takes place at the developer's site by the internal teams, beforerelease to external customers. This testing is performed without the involvement of the development teams.*

i. *Alpha Testing - In SDLC*

The following diagram explains the fitment of Alpha testing in the softwaredevelopment life cycle.



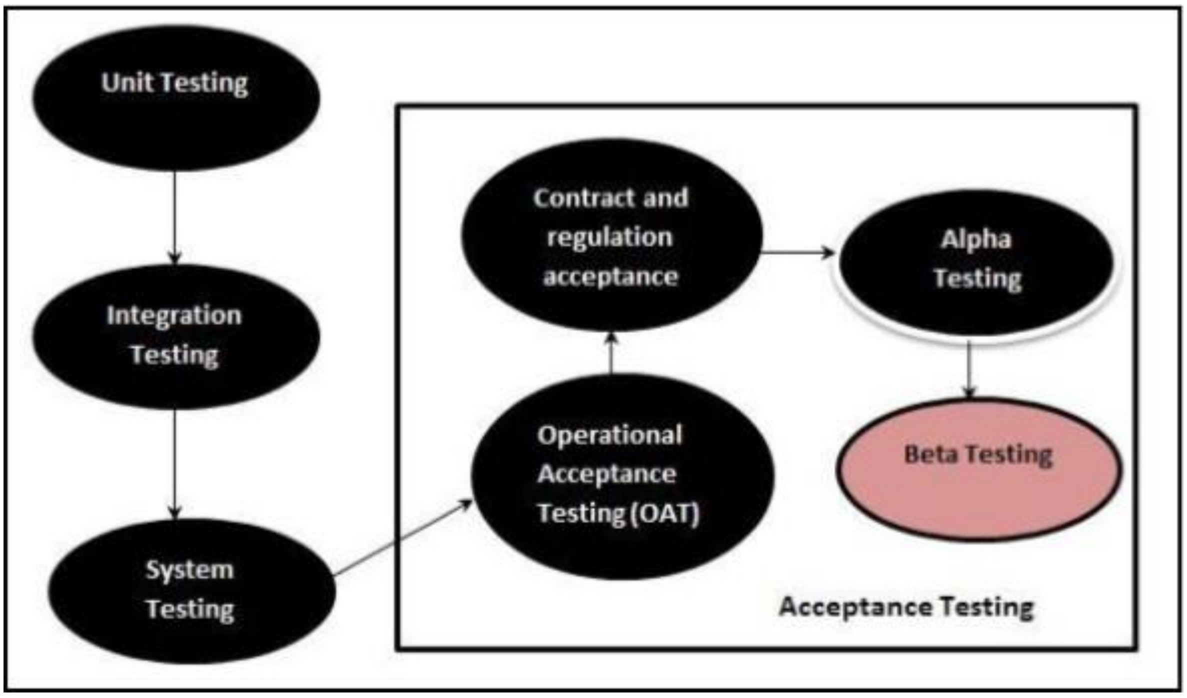
ii. How do we run it?

- *In the first phase of alpha testing, the software is tested by in-house developers during which the goal is to catch bugs quickly.*
- *In the second phase of alpha testing, the software is given to the software QA team for additional testing.*
- *Alpha testing is often performed for Commercial off-the-shelf software (COTS) as a form of internal acceptance testing, before the beta testing is performed.*

Beta Testing:

- *Beta testing also known as user testing takes place at the end users site bythe end users to validate the usability, functionality, compatibility, and reliability testing.*
- *Beta testing adds value to the software development life cycle as it allows the "real" customer an opportunity to provide inputs into the design, functionality, and usability of a product. These inputs are not only critical tothe success of the product but also an investment into future products whenthe gathered data is managed effectively.*
- *Beta Testing - In SDLC*

The following diagram explains the fitment of Beta testing in the softwaredevelopment *life cycle:*



Q7. Explain Object oriented application Testing.

Ans.

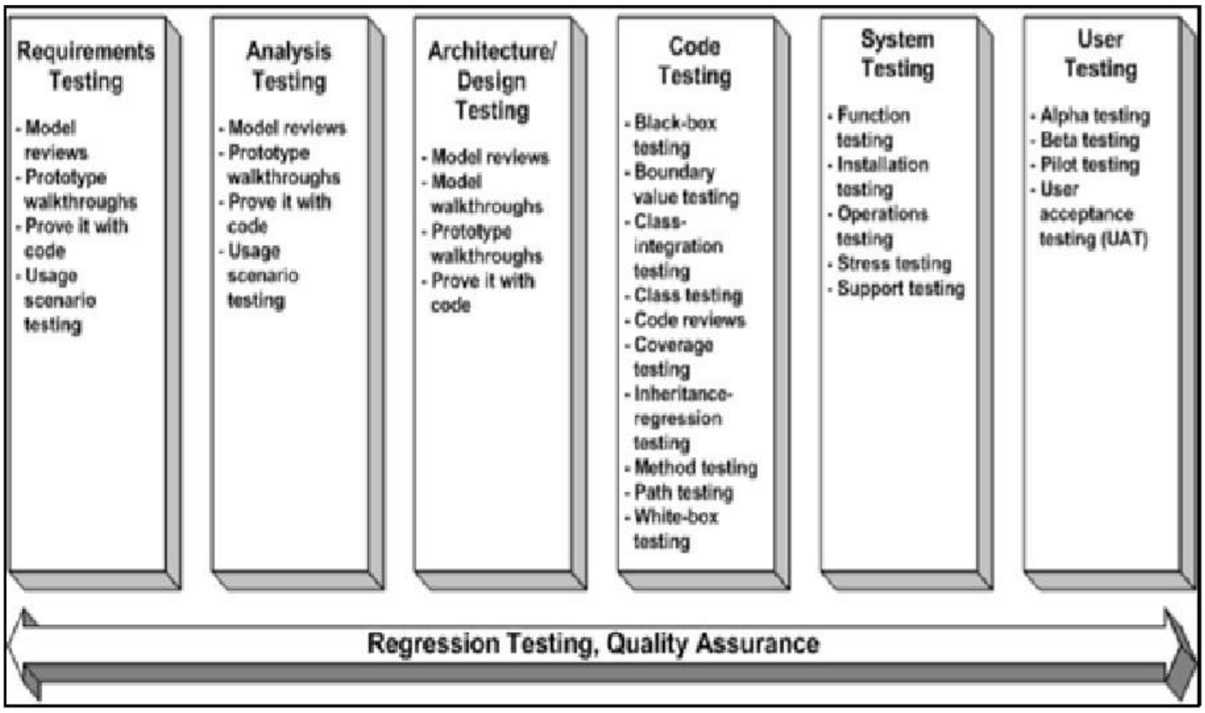
The Full-Lifecycle Object-Oriented Testing (FLOOT) methodology is a collection oftesting techniques to verify and validate object-oriented software.

The FLOOT lifecycle is depicted in Figure 9, indicating a wide variety of techniques(described in Table 9 are available to you throughout all aspects of software development.

The list of techniques is not meant to be complete: instead the goal is to make itexplicit that you have a wide range of options available to you.

It is important to understand that although the FLOOT method is presented as a collection of serial phases it does not need to be so: the techniques of FLOOT can beapplied with evolutionary/agile processes as well.

The reason why I present the FLOOT in a "traditional" manner is to make it explicitthat you can in fact test throughout all aspects of software development, not just during coding.



Q8. Explain Client server testing in terms of your final year project.

Ans.

i. *This type of testing usually done for 2 tier applications (usually developedfor LAN) Here we will be having front-end and backend.*

ii. *The application launched on front-end will be having forms and reports which will be monitoring and manipulating data.E.g: applications developed inVB, VC++, Core Java, C, C++, D2K, PowerBuilder etc.,*

iii. *The backend for these applications would be MS Access, SQL Server,Oracle, Sybase, Mysql, Quadbase.*

iv. *The tests performed on these types of applications would be– Userinterface testing Manual support testing– Functionality testing– Compatibility testing & configuration testing – Intersystem testing.*

Q9. Describe various web application testing techniques.

Ans.

1. Functionality Testing *- The below are some of the checks that areperformed but not limited to the below list:*

- *Verify there is no dead page or invalid redirects.*
- *First check all the validations on each field.*
- *Wrong inputs to perform negative testing.*
- *Verify the workflow of the system.*
- *Verify the data integrity.*

2. Usability testing *- To verify how the application is easy to use with.*

- *Test the navigation and controls.*
- *Content checking.*
- *Check for user intuition.*

3. Interface testing *- Performed to verify the interface and the dataflowfrom one system to other.*

4. Compatibility testing*- Compatibility testing is performed based onthe context of the application.*

- *Browser compatibility*
- *Operating system compatibility*
- *Compatible to various devices like notebook, mobile, etc.*

5. Performance testing *- Performed to verify the server response time* *and throughput under various load conditions.*

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

6. Security testing *- Performed to verify if the application is secured on web as data theft and unauthorized access are more common issues and below are some of the techniques to verify the security level of the system.*

- *Injection*
- *Broken Authentication and Session Management*
- *Cross-Site Scripting (XSS)*
- *Insecure Direct Object References*
- *Security Misconfiguration*
- *Sensitive Data Exposure*
- *Missing Function Level Access Control*
- *Cross-Site Request Forgery (CSRF)*
- *Using Components with Known Vulnerabilities*
- *Invalidated Redirects and Forwards*

Q10. Explain Performance Testing with example.

Ans.

Performance testing *- Performed to verify the server response time and throughput under various load conditions.*

- *Load testing - It is the simplest form of testing conducted to understand the behavior of the system under a specific load. Load testing will result in measuring important business critical transactionsand load on the database, application server, etc. are also monitored.*

- *Stress testing - It is performed to find the upper limit capacity of thesystem and also to determine how the system performs if the currentload goes well above the expected maximum.*

- *Soak testing - Soak Testing also known as endurance testing, is performed to determine the system parameters under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performanceissues. The main aim is to discover the system's performance undersustained use.*

- *Spike testing - Spike testing is performed by increasing the number ofusers suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether thesystem will be able to sustain the work load.*

ST Assignment No. 4

Q1. Explain the goal of test planning in Software Testing?

Ans.

1. Performing testing tasks would be very difficult if the programmers wrote their code without telling what it does, how it works, or when it will be complete.

2. Software testers don't communicate what tester plans to test, what resources you need, and what your schedule is, your project will have little chance of succeeding.

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

4. The test plan is simply a by-product of the detailed planning process that's undertaken to create it. It's the planning process that matters, not the resulting document.

5. The ultimate goal of the test planning process is communicating (not recording) the software test team's intent, its expectations, and its understanding of the testingthat's to be performed.

Q2. Explain Test Planning topics in details.

Ans.

1. The problem with test planning approach is that it makes it too easy to put the emphasis on the document, not the planning process.

2. Test leads and managers of large software projects have been known to take an electronic copy of a test plan template or an existing test plan, spend a few hours cutting, copying, pasting, searching, and replacing, and turn out a "unique" test planfor their project.

3. They felt they had done a great thing, creating in a few hours what other testers had taken weeks or months to create.

4. They missed the point, though, and their project showed it when no one on the product team knew what the heck the testers were doing or why.

a)High Level Expectations

1. What's the purpose of the test planning process and the software test plan?

2. What product is being tested?

3. What are the quality and reliability goals of the product?

b)People, Places and Things

1. Test planning needs to identify the people working on the project, what they do, and how to contact them.

2. If it's a small project this may seem unnecessary, but even small projects can have team members scattered across long distances or undergo personnel changesthat make tracking who does what difficult.

3. A large team might have dozens or hundreds of points of contact. The test team will likely work with all of them and knowing who they are and how to contact them isvery important.

4. The test plan should include names, titles, addresses, phone numbers, email addresses, and areas of responsibility for all key people on the project.

c)Definitions

1. The software doesn't do something that the product specification says it shoulddo.

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 mentionbut should build.

The test plan should define the frequency of builds (daily, weekly) and the expected quality level.

- *Test release document (TRD).A document that the programmers release with each build stating what's new, different, fixed, and ready for testing.*
- *Alpha release. A very early build intended for limited distribution to a few key customers and to marketing for demonstration purposes. It's not intended to be used in a real-world situation. The exact contents and quality level must be understood by everyone who will use the alpha release.*
- *Beta release. The formal build intended for widespread distribution topotential customers.*

Q3. Define the role of people, places and things in test planning with examples.

Ans.

1. *Test planning needs to identify the people working on the project, what they do,and how to contact them.*
2. *If it’s a small project this may seem unnecessary, but even small projects can have team members scattered across long distances or undergo personnel changesthat make tracking who does what difficult.*
3. *A large team might have dozens or hundreds of points of contact. The test team will likely work with all of them and knowing who they are and how to contact them isvery important.*
4. *The test plan should include names, titles, addresses, phone numbers, email addresses, and areas of responsibility for all key people on the project.*

Examples:

- **People:**

Testers: They are responsible for executing test cases and reporting any defects they find. They should be trained on the software being tested and the testing techniques they will be using.

Developers: They may be involved in reviewing and verifying defect reports and fixing any issues found during testing.

- **Place:**

Test environment: This is the place where the testing will take place. It may be a physical lab or a virtual environment, depending on the software being tested.

Production environment: This is the environment where the software will ultimately be deployed.

- **Things:**

Test cases: These are the documented steps that testers will follow to perform the testing.

Test data: This is the data that will be used during testing.

Q4. Explain the inter group responsibilities in Software Testing.

Ans.

1. *Inter-group responsibilities identify tasks and deliverables that potentially affectthe test effort.*
2. *The test team’s work is driven by many other functional groups—programmers,*

project managers, technical writers, and so on.

3. If the responsibilities aren't planned out, the project— specifically the testing— can become a comedy show of "I've got it, no, you take it, didn't you handle, no, I thought you did," resulting in important tasks being forgotten.

a) What will and won't be tested?

1. There may be components of the software that were previously released and have already been tested. Content may be taken as is from another software company.

2. An outsourcing company may supply pre-tested portions of the product.

b) Test Schedule

1. The test schedule takes all the information presented so far and maps it into the overall project schedule.

2. This stage is often critical in the test planning effort because a few highly desired features that were thought to be easy to design and code may turn out to be very time consuming to test.

3. An example would be a program that does no printing except in one limited, obscure area.

4. No one may realize the testing impact that printing has, but keeping that feature inthe product could result in weeks of printer configuration testing time.
Completing a test schedule as part of test planning will provide the product teamand project manager with the information needed to better schedule the overall project.

5. They may even decide, based on the testing schedule, to cut certain features fromthe product or postpone them to a later release.

c) Test Cases

1. The test planning process will decide what approach will be used to write them,where the test cases will be stored, and how they'll be used and maintained.

d) Bug Reporting

1. The possibilities range from shouting over a cubical wall to sticky notes to complex bug-tracking databases.

2. Exactly what process will be used to manage the bugs needs to be planned so that *each and every bug is tracked from when it's found to when it's fixed—and never, ever forgotten.*

e) Metrics and Statistics

1. Metrics and statistics are the means by which the progress and the success of the project, and the testing, are tracked.

2. The test planning process should identify exactly what information will be gathered, what decisions will be made with them, and who will be responsible for collecting them.

Examples of test metrics that might be useful are

- *Total bugs found daily over the course of the project.*
- *List of bugs that still need to be fixed.*
- *Current bugs ranked by how severe they are.*

f) Risk and Issues

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

Q5. Explain Test schedule along with the example.

Ans.

1. The test schedule takes all the information presented so far and maps it into the overall project schedule.

2. This stage is often critical in the test planning effort because a few highly desired features that were thought to be easy to design and code may turn out to be very time consuming to test.

3. An example would be a program that does no printing except in one limited, obscure area.

4. No one may realize the testing impact that printing has, but keeping that feature inthe product could result in weeks of printer configuration testing time.

5. Completing a test schedule as part of test planning will provide the product team *and project manager with the information needed to better schedule the overall project.*

6. *They may even decide, based on the testing schedule, to cut certain features fromthe product or postpone them to a later release.*

- *Example:*

Test Activity	Start Date	End Date
Unit Testing	05/10/2023	05/12/2023
Integration Testing	05/13/2023	05/16/2023
System Testing	05/17/2023	05/21/2023
Acceptance Testing	05/24/2023	05/28/2023
Regression Testing	05/29/2023	05/31/2023

Q6. Explain the goal of test case planning in Software Testing.

Ans.

1. Organization: *Even on small software projects it's possible to have many thousands of test cases. The cases may have been created by several testers overthe course of several months or even years. Proper planning will organize them so that all the testers and other project team members can review and use them effectively.*

2. Repeatability: *It's necessary over the course of a project to run the same tests several times to look for new bugs and to make sure that old ones have been fixed. Without proper planning, it would be impossible to know what test cases were last run and exactly how they were run so that you could repeat the exact tests.*

3. Tracking: *Similarly, you need to answer important questions over the course of a project. How many test cases did you plan to run? How many did you run on the last software release? How many passed and how many failed? Were any test cases skipped? And so on. If no planning went into the test cases, it would be impossible to answer these questions.*

4. Proof of testing *(or not 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, and dangerous, to release software in which a few test cases wereskipped. Proper test case planning and tracking provides a means for proving what was tested.*

Q7. Describe test design in short.

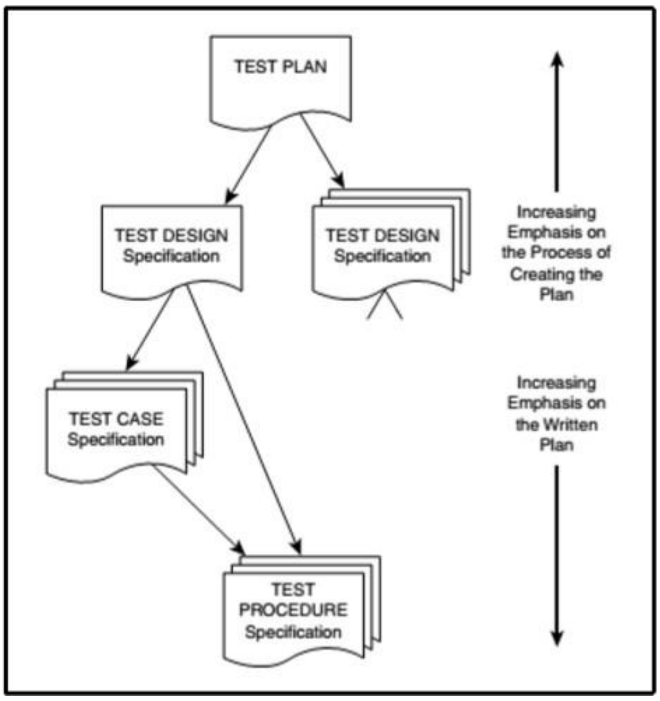
Ans.

1. *The overall project test plan is written at a very high level.*

2. *It breaks out the software into specific features and testable items and assigns them to individual testers, but it doesn't specify exactly how those features will be tested.*

3. *There may be a general mention of using automation or black-box or white-box*
4. *testing, but the test plan doesn't get into the details of exactly where and how theywill be used.*

5. *This next level of detail that defines the testing approach for individual software features is the test design specification.*


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

Q8. Define Test Cases and test procedures in short description.

Ans.

a) Test Cases

1. Identifiers*: A unique identifier that can be used to reference and locate the testdesign spec. The spec should also reference the overall test plan and contain pointers to any other plans or specs that it references.*

2. Features to be tested*: A description of the software feature covered by the test design spec—for example, "the addition function of Calculator," "font size selectionand display in WordPad," and "video card configuration testing of QuickTime." Forexample, "Although not the target of this plan, the UI of the file open dialog box willbe indirectly tested in the process of testing the load and save functionality."*

3. Approach*: A description of the general approach that will be used to test the features. It should expand on the approach, if any, listed in the test plan, describe the technique to be used, and explain how the results will be verified.*

4. Test case identification*: A high-level description and references to the specific test cases that will be used to check the feature. It should list the selected equivalence partitions and provide references to the test cases and test procedures used to run them.*

5. Pass/fail criteria*: Describes exactly what constitutes a pass and a fail of the tested feature. This may be very simple and clear—a pass is when all the test cases are run without finding a bug. It can also be fuzzy—a failure is when 10 percent or more of the test cases fail. There should be no doubt, though, what constitutes a pass or a fail of the feature.*

b) Test Procedures

1. The test procedure or test script specification defines the step-by-step details of exactly how to perform the test cases.

Here's the information that needs to be defined:

- *Identifier: A unique identifier that ties the test procedure to the associated test cases and test design.*
- *Purpose: The purpose of the procedure and reference to the test cases that it will execute.*
- *Special requirements: Other procedures, special testing skills, or special equipment needed to run the procedure.*
- *Procedure steps: Detailed description of how the tests are to be run.*