UNIT III

CHAPTER 11

Software Testing

Syllabus

Verification and Validation, Introduction to Testing, Testing Principles, Testing Objectives, Test Oracles, Levels of Testing, White-Box Testing/Structural Testing, Functional/Black-Box Testing, Test Plan, Test-Case Design

Syllabus Topic : Verification and Validation

11.1 Verification and Validation

V & V stands for Verification and Validation. It is a whole life-cycle process - it is applied at each stage of software development process.

- **Verification**
  - "Are we building the product right ?"
  - Software should conform to its specification (SRS). It is about confirming with the user specified requirements.
- **Validation**
  - "Are we building the right product ?"
  - Software should do what the user 'really requires'. It is about confirming with the proper functionality and performance of the system with good security of data.

Table 11.1.1 : Difference between Verification and Validation

Sr. No.	Verification	Validation
1.	It is the process of confirming whether the software meets its requirement specification.	It is the process of confirming whether the software meets user requirements.
2.	Inspections, walkthroughs and reviews are the examples of Verification.	Structural testing, black box testing, integration testing, system testing, acceptance testing are the examples of validation.

Sr. No.	Verification	Validation
3.	It is usually a process of static testing i.e. inspecting without executing on computer.	It is usually a process of dynamic testing i.e. testing by executing on computer.
4.	It is the process of confirming if the phases are completed correctly.	It is the process of determining if product as a whole satisfies the requirements.
5.	It is the process of examining the product to discover its defects.	It is the process of executing a product to expose its defects.
6.	It answers, "Are we building the product right"?	It answers, "Are we building the right product"?

- **V & V Goals**
  - To establish confidence that the software system is 'fit for purpose'. This does NOT mean system with zero defects. It ensures that system is good enough for its intended purpose.
- **This level of degree of confidence depends on**
  - ***Software's functions*** i.e. how useful the software is to an organisation.
  - ***User expectations*** usually users have low expectations from the software and they are not at all surprised if the system fails during the use.
  - ***Marketing environment*** While marketing the software product, the sellers must also consider the competing software, the price that the customers are willing to pay for the software and the required schedule for delivering the system.
- **Objectives of V&V Process**
  - Discover the defects in a system;
  - Assess whether or not the system is useful.
- **Advantages of V and V Process**
  - Prevents faults and stops fault multiplication.
  - Avoids the downward flow of errors.
  - Earlier detection of errors leads to Lower defect Resolution cost.
  - Improves quality and reliability.
  - Reduces the amount of Re-work.
  - Improves Risk Management
  - Allows testers to be active in the early phases of the project's lifecycle.

11.1.1 Static and Dynamic Verification

Q. Differentiate: Static and dynamic testing

Verification and Validation of software can be done by two methods - Static and Dynamic.

Table 11.1.2 : Difference between Static and Dynamic Verification

Sr. No.	Static Verification	Dynamic Verification
1.	It is about reviewing the software development documentation. It doesn't include any inputs and outputs i.e. - No verification of actual outputs against the expected outputs. This is same as Software inspections.	It is not reviewing. It is just opposite to Static verification. Dynamic Verification is about verifying whether expected and actual outputs match or not. Dynamic Verification allows the developer or user to give any type of inputs and cross checks whether the actual output of these inputs match with the expected outputs. This is same as Software Testing.
2.	It is *about prevention*	It is *about cure*.
3.	More cost-effective than dynamic testing	Less cost-effective
4.	Achieves 100% statement coverage in short time.	Achieves less than 50% statement coverage because dynamic testing finds bugs only in the executable code.
5.	Takes less time to verify as it involves only checking the documents - it doesn't involve any inputs or executions.	Dynamic testing may involve running several test cases, each of which may take comparatively longer time.
6.	It can be done before compilation	It can take place only after compilation and linking
7.	Static verification can finds syntax errors, hard code i.e. hard to maintain and test, code that does not conform to coding standards.	Dynamic testing finds fewer bugs than static testing as most of them are detected in static verification.
8.	Example : Software inspections. This is concerned with analysis of the static system representation to discover problems.	Example : Software testing. Concerned with exercising and observing product behaviour.

11.1.2 V and V Model

The below Verification and Validation model (V & V Model) defines various levels V & V activities that are under taken at different stages of SDLC phases.

- V and V is a whole SDLC life-cycle process – it is applied at each phase of software development life cycle process.
- System verification and validation is started early in the development process.
- Careful V and V planning is must to get the most out of testing and inspection processes.
- The V and V plan should identify the balance between static verification and testing (dynamic verification).
- The below Verification and Validation model (V and V Model) defines various levels of V and V activities that are under taken at different stages of SDLC phases.
- You will see that this V and V Model looks like **'V'** shaped and so it is also called as 'V-model'.

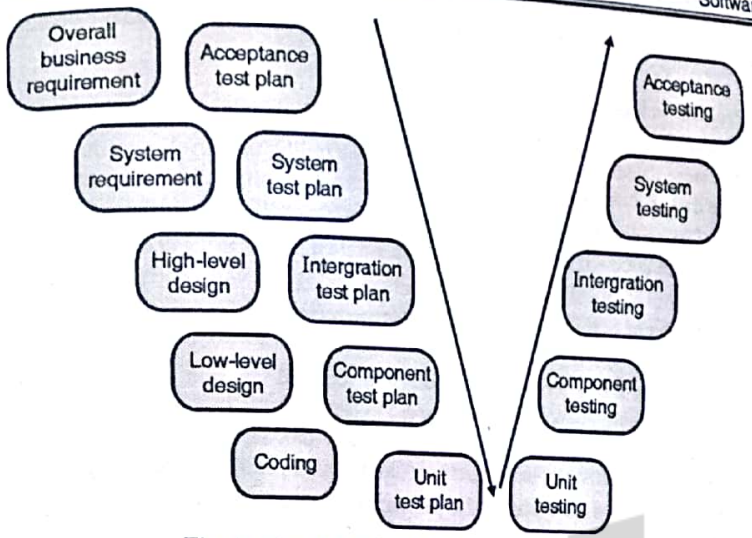


Fig. 11.1.1 : V & V Model / Levels of Testing

V-model is split into two parts :

1. Test design (on left side of V) is done early
2. Test execution (right side of V) is done in the end

By performing early design of tests and deferring only the test execution till the end will achieve three important advantages:

1. Achieves more parallelism and reduces the end-of-cycle time taken for testing.
2. By building the test design for each activity of the system, we build better upfront validation. Thus, this reduces the last-minute errors.
3. Tests are designed by the people with appropriate skill tests.

Syllabus Topic : Introduction to Testing

11.2 Introduction to Testing

- **Testing** is done by software tester to find **bugs** in a system as early as possible and notify the same to developers so that they get fixed.
- In 90's, Testing was not given that important and thus, systems developed then had lot of serious bugs.

Few Examples that define the Need of Testing

- **Excel : 77.1 x 850 = 100,000 or 65,535**

  The bug in MS Excel is that when multiplying two numbers whose product equals 65,539 will actually give the answer as 100,000.
- **Excel incorrectly assumes that the year 1900 is a leap year**

  Excel is developed from Lotus 1-2-3. When Lotus 1-2-3 was first released, the program assumed that the year 1900 was a leap year, even though it actually was not a leap year. This made it easier

for the program to handle leap years and caused no harm to almost all date calculations in Lotus 1-2-3.

- **Y2K Problem in Payroll systems designed in1974**

In 1974, when first payroll system was developed, it was learnt that most of the memory space is utilised to store dates. And so, to minimize the space utility, the developers thought of storing the year of the date in 2 digits instead of 4 digits i.e. instead of storing 1900, store 00.

This idea of space conservation was OK then, but after 25 years, in the yr 2000, the question arose that how to store 2000 – if it is stored as 00 then would it represent 1900 or 2000?

So, all these above defects in the software are caused as the Software Testing was not followed from the initial phases of SDLC. Testing must not be performed in the last stage of SDLC just for the sake of doing it.

The cost of fixing a bug (error) and making the required changes in early phases of software development is less as compared to the same detected in later phases. This is because making changes in earlier phases is easy and if the bugs are detected in later phases, it becomes difficult to repeat the previous phases once again in order to make the required changes and thus, the cost goes high.

Testing must be done from initial SDLC phases to prevent and detect the defects so as to reduce cost and schedule risks.

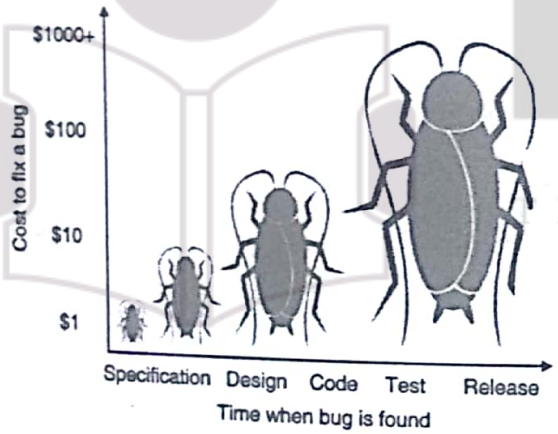


Fig. 11.2.1 : Cost of fixing bugs at different SDLC phases

Testing finds different kinds of non-conformances also called as errors. An error is a difference between the measured value and the specified value. Following are some types of errors :

1. **Syntax Error :** improper code.
2. **Logical Error :** mistake in the logic compiles and runs the program but produces incorrect output. *For example*; If we write a MACRO in C as SUM (-) by mistake and if we try to perform 'c = a SUM b' expecting to get the result of their addition but instead we get the output of subtraction as we have defined SUM to the (-).
3. **Run Time error :** This error is generated during the execution of the program such as array bound error, divide by zero error floating point error etc.

Goal of testing is to find ***bugs*** as early as possible and make sure they get fixed.

11.2.1 Bugs

A software bug occurs when :

1. The software doesn't do something that is defined in the SRS..

   Example : The SRS for a calculator states that it must perform four functions correctly i.e. addition, subtraction, multiplication, and division. Then, as a tester, if you press the ÷ key and nothing happens or if you a get a wrong answer then that's a bug according to this rule.

2. The software does something that is directly denied in the SRS.

   Example : The SRS for a calculator states that the calculator should never crash, lock up, or freeze. Then, as a tester, if you pound on the keys and the calculator stops responding to your input then that's a bug according to this rule.

3. The software does something that the SRS doesn't mention.

   Example : As a tester, if you find that besides addition, subtraction, multiplication, and division, the calculator also performs square roots since an ambitious programmer just included this function in to it because he felt it would be a great feature in the calculator. Then that's a bug according to this rule.

4. The software doesn't do something that the SRS doesn't mention but should do.

   The purpose of this rule is to catch things that were forgotten in the specification.

   Example : As a tester, if you start testing the calculator and come to know that when the battery gets weak, you no longer get correct answers for your calculations. It was assumed that the battery will be always fully charged. That means, correct calculations were not performed with weak batteries and it wasn't specified in the SRS about how it should react in such a situation but the calculator must either work properly until the batteries were completely dead or at least notify the user in some way about the weak battery. Then that's a bug according to this rule.

5. The software is difficult to understand, difficult to use or slow.

   Example : As a tester, if you find that the buttons of the calculator were too small or may be the placement of = key made it hard to use or may be the display is hard to read under bright lights, then that's a bug according to this rule.

Causes of Bugs

1. Incomplete or Erroneous Specification (IES): The root cause of software bugs is the incorrect requirement specification or constantly changing requirements.
2. Misinterpretation Of Customer Communication (MCC): Requirements are not communicated well to the entire development team.
3. The next main cause of bugs is the *design*.

   The first – second and third causes can be well described by an old saying, "If you can't say it, you can't do it."
4. Intentional deviation from SRS
5. Violation of programming standards
7. Inconsistent module interface
8. Error in designs
9. Incomplete testing
10. Inaccurate or incomplete documentation
11. Difficult human-computer interface

Bug Life Cycle

Q. Define the terms : Bug life cycle

- Bugs can occur at any stage of SDLC process. The bug then attains different states which can be termed as its life cycle.
- The bug life cycle begins when the developer makes mistakes and the cycle ends when the bug is fixed and it is no longer in existence.
- The bug life cycle is shown diagrammatically as follows :

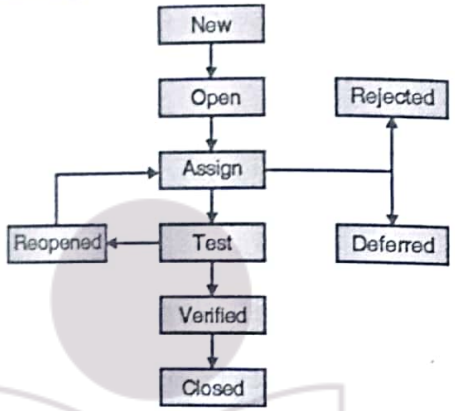


Fig. 11.2.2 : Bug Life Cycle

The different states of a bug in the can be described as follows :

1. **New :** When the bug is detected for the first time, its state is called as NEW. This means that the bug is not yet approved as a bug.
2. **Open :** After detecting the bug, the leader of the tester approves that the bug is genuine needs to be fixed; thus, he changes its state as OPEN.
3. **Assign :** After changing its state as OPEN, the test leader assigns the bug to corresponding developer or developer team. The state of the bug is now as ASSIGN.
4. **Test :** The developer then fixes the bug and re-assigns the bug to the testing team for next round of testing. The state of the bug is now as TEST. It specifies that the bug has been fixed and is released to testing team for re-testing.
5. **Deferred :** The bug changed to DEFERRED state means that the bug is expected to be fixed in next releases. The reasons for changing the bug to this state might be that the priority of the bug may be low, or lack of time in release of the software or the bug may not have major affect 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 now as REJECTED.
7. **Duplicate :** If the same bug repeats twice then that bug status is changed to DUPLICATE.
8. **Verified :** Once the bug is fixed, the tester tests the bug. If the bug is no more present, the tester changes its state to VERIFIED.
9. **Reopened :** If the bug still exists in the software even after the bug is fixed by the developer, then the tester changes its state to REOPENED. The bug traverses through the bug life cycle once again by assigning it to the developer.

10. **Closed :** Once the bug is fixed, it is tested by the tester and if he feels that the bug no longer exists in the software and it will not even crop up in the future, then he changes the bug state to CLOSED which means that the bug is fixed, tested and approved.

The tester or the developer decides the severity of the bug on the basis of the priorities assigned to the bugs. *There are different types of priorities assigned to the bugs.*

1. **Critical / Show's Stopper :** A bug that prevents further testing of the product is called as Critical bug.

   Examples : A missing menu option to access a function under test.
2. **Major / High :** A bug that does not function as expected or cause other functionalities to fail to meet the specified requirements is called as Major Bug.

   Examples: inaccurate calculations.
3. **Average / Medium :** A bug that does not conform to standards and conventions is called as Medium Bug.

   Examples : Visual and text links that may lead to different end points. .
4. **Minor / Low :** look and feel bugs that do not affect the functionality of the system is called as Minor Bug.

Syllabus Topic : Testing Principles

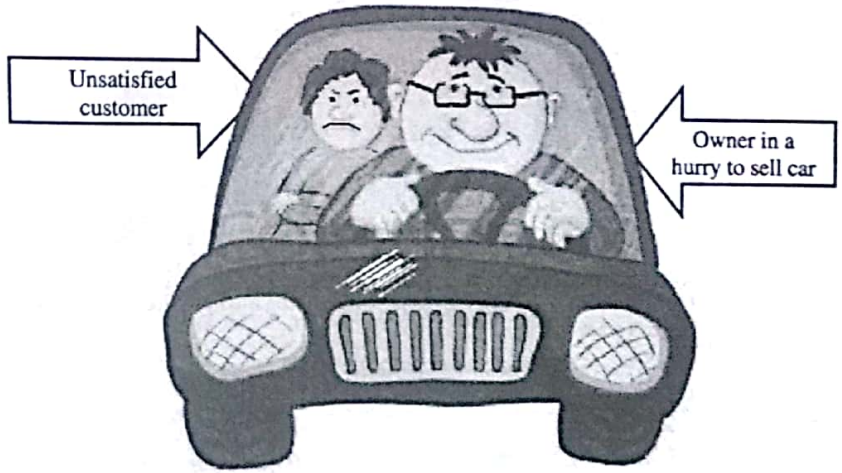
11.3 Testing Principles

Below is a list of ***testing principles*** that serves as a basis for testing objectives so as to provide quality products to customers.

1. Testing should focus on finding defects before customers find them.

   The software product should be thoroughly tested to ensure that the software product is defect free before delivering it to the customer; else if the customer comes across the defects at the installation time, then he becomes unsatisfied and will not be ready to buy it.

   Example : An interesting act between car owner and car buyer (on phone)



Car owner : Hello Mr. Amol, you can come today and take the car. Customer Amol Arrives to car owner's house and sees the car that he is going to buy in a very bad condition…

Customer : What's this? Is this the car, you want to sell. Oh no! How dirty it looks. There are so many crashes and the color has also fainted.

Car owner : Chill…Pill… Mr. Amol. The car is complete… you just need to paint it.

Customer : Ok… First let me take a test drive. …Both sit in the Car…

Car owner : Mr. John, this car has the best possible transmission and brake, and accelerates from 0 to 80 kmph in under 20 seconds!

Customer : Well. It may be true. But unfortunately it accelerates even faster when I press the brakes.

Moral : The car owner was trying to sell his second hand car without repairing the defects in it. When customer came to know about these defects, he refused to buy the car.

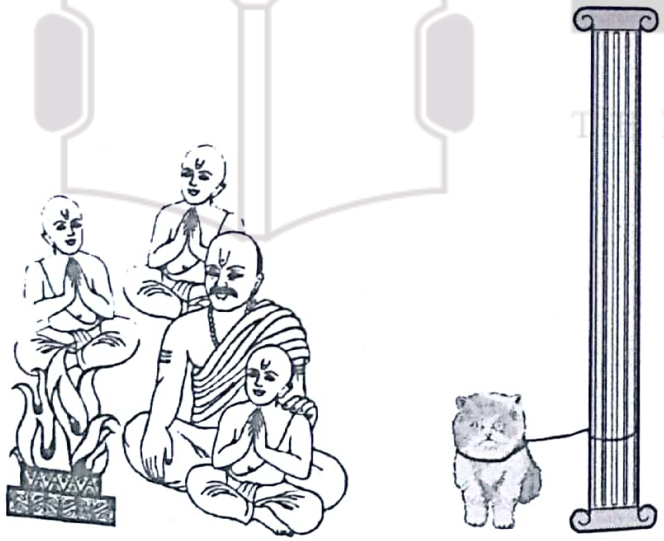
2. Testing is applied all through the SDLC process and not just at the end of the SDLC activity.

Example : This principle can be better understood from the graph describing the 'Cost of fixing bugs at various SDLC phases'.

3. Understand the reason behind test - "what to test" and "how to test"

Before performing any test, you should understand what product you will be testing, what is the expected outcome of that product, why you are performing any test. Else you will end up with inappropriate tests that will not address what the product should do.

Example : An interesting act in an Ashram between Saint and his Disciples



Saint : Ok. Disciples, I will sit here and meditate for a while. So, please, see to it that there is no disturbance.

Disciple : Ok. Sir. …A cat comes there and disturbs the Saint. So, disciples tie the cat to a pillar…..

Disciples : Ooops! This cat is creating a lot of nuisance. Let's tie it to a pillar. After five years… …Disciples are running here and there, searching for a cat…

Disciple : Oh shit! I'm unable to find any cat. What to do. A guest in the Ashram asks the disciple: Sir, what are you searching for?

Disciple : We need a cat. Only when we get a cat, then we can tie it to a pillar and only after that, our saint will start meditating.

Moral : The daily routine of catching a cat and tying it to a pillar becomes a tradition. And the further generations, without any actually understanding that why the cat was tied; they everyday searched for a cat by hook or crook and tied it to a pillar even if it didn't come there to disturb. It is just ridiculous to catch a cat without understanding why it is caught.

4. Test the Tests (testing tools) First - Defective testing tools are more dangerous than a defective product. Testing tool is also ultimately software produced by human beings. So, we cannot assure that testing tools are perfect. So, ensure that testing tools are not faulty before you use them.

Example : An interesting act in between ENT (Ear Nose Tongue) doctor and his patient

Doctor : I want to test the range within which you can hear. I will ask you from various distances to tell me your name, and you should tell me your name.

Patient : Ok. Sir. I understood. ..(doctor 30 feet far)…

Doctor : what is your name?

Doctor : I cannot listen to her answer. I think she cannot hear me. …doctor goes at 20 feet…

Doctor : what is your name?

Doctor : I cannot listen to her answer. I think she still cannot hear me. …from 10 feet…

Doctor : What is your name?

Patient : For the third time, I am repeating my answer, my name is Richa.

Moral : The ENT doctor himself had a defect in his ear. So, how can he succeed in testing the patient i.e. a faulty and a defective doctor cannot test a patient.

5. Tests develop immunity and have to be revised constantly.

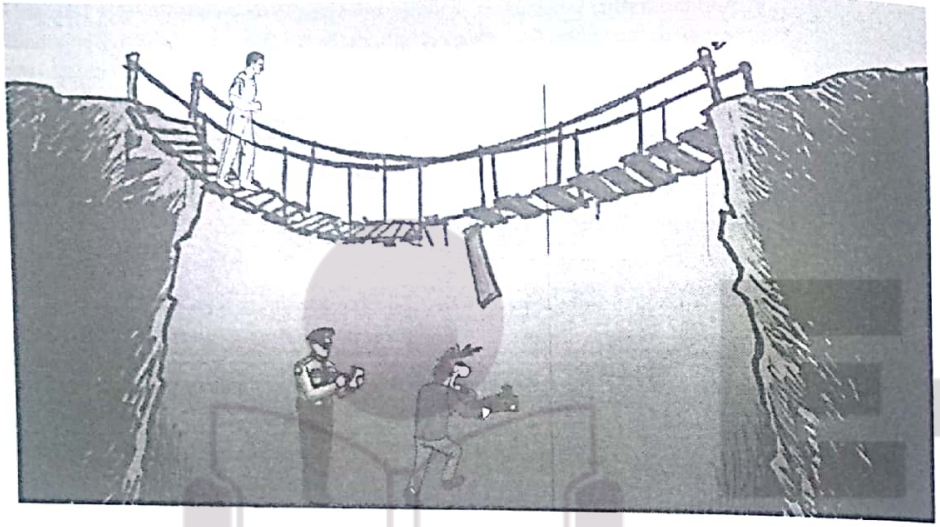
Example : Tests are like pesticides that need to be constantly revised to tackle the new pests i.e. defects

6. Testing encompasses on defect prevention because Prevention is better than cure

Example : If you know that you are having low haemoglobin, then start eating 'Gu-chan i.e. jaggery and groundnut' to avoid fainting on some day.

Example : Assume an over bridge and people are passing over it. And what happens is that whenever a person passes over the bridge, he falls down the bridge. No one knew what's the reason? Then Government appointed a police under the bridge to catch the persons who are falling

down the bridge. But what if the falling person is too fat and police is too thin. And this is not the proper solution. It's better to find the reason behind the falling so that we can prevent the accident in futures. This will not waste the police officers time spent in standing over there and also he can spend his mind and knowledge in some other work at the same time. There we say that "prevention of falling down of persons is better than waiting for the persons to fall then curing them"



7. Defect prevention and defect detection should supplement each other and not be considered as mutually exclusive.
8. Defect Prevention improves the quality of the process while defect detection and testing is needed to catch and correct the defects that escape the process. Both are necessary for producing a quality product.

Example : An interesting act between three Chinese doctor brothers. ...One is a Surgeon... famous all over the world...

Surgeon to the patient	:	Hmm... You are suffering from tumor. No problem. I will do an operation and remove this tumor. All will be well soon. ...One is a doctor... famous in the city they lived...
Doctor to the patient	:	hmm... You are suffering from Jaundice. No problem. This is the first stage. Take these tablets. You will get cured very soon. ...But, Both Surgeon and Doctor always took advice of their youngest brother.
Doctor to his youngest	:	Tell me how to prevent from Cholesterol. Brother
Younger Brother	:	Avoid oily food; take 'Oats' in the breakfast and exercise daily and have balanced diet.
Doctor	:	Wow! Thanks for your preventive measure.
Surgeon to his youngest	:	Brother. Tell me how to prevent from Blood brother Cancer.

Younger Brother	:	Avoid Smoking, avoid taking drinks and avoid intake of Nicotine through Gutka and as such.
Surgeon	:	Oh. Thank you. You know preventive measures for all the defects. Your advice is very effective.
Moral	:	Preventing an illness is more effective than curing it. People who prevent defects usually do not get much attention. They are usually unseen heroes of an organization and those who put out the fires are the ones who get visibility. This however, should not demotivate the people who do the defect prevention.

Syllabus Topic : Testing Objectives

11.4 Testing Objectives

1. To execute a program with the intent of finding an *error*.
2. To verify and validate if the system meets the requirements and be executed successfully in the Intended environment.
3. To verify and validate if the system is "Fit for purpose".
4. To verify and validate if the system does what it is expected to do.
5. Exhaustive testing is not possible that means the testing can only trace the presence of defects and never their absence.
6. Testing should begin 'in the small' and progress towards testing 'in the large'.

   Example : First, the unit testing is performed to check each and every code line; then black-box testing is performed to test the functions; then integration testing to test the individual components and the integration of components; at last the system testing is done to test the overall working of the system.
7. Intelligent and well planned automation is the key to realizing the benefits of testing. That means, tests should be planned long before the testing begins
   - First know the reason behind why you want to automate the tests
   - Analyze the various tools before choosing on s being most appropriate for your need

- Choose the tools that match your needs rather than changing your needs to match the tool capabilities.

Syllabus Topic : Test Oracles

11.5 Test Oracles

A function that determines if the application has behaved correctly in response to a test action is called a *test oracle.*

The test oracle is either of the following

- a program (separate from the system under test) which takes the same input and produces the same output
- documentation that gives specific correct outputs for specific given inputs
- a documented algorithm that a human could use to calculate correct outputs for given inputs
- a human domain expert who can somehow look at the output and tell whether it is correct
- or any other way of telling that output is correct.

Test oracle is a mechanism for determining whether a test has passed or failed.

A **test oracle** may be the existing system (for a benchmark), other software, a user manual, or an individual's specialized knowledge.

☞ **Some issues faced while testing a large and complex applications**

- It may require millions of test runs.
- Size of the outputs might exceed the expectation

So, automated oracles are essential to overcome these issues.

The use of test oracles involves comparing the actual output(s) of the system under test for a given test-case input with the expected output(s) that the oracle determines that product should have.

A good test oracle would have three capabilities :

1. A *generator* - provides expected results for each test.
2. A *comparator* - compares expected and actual results.
3. An *evaluator* - determines whether the test passed or failed

Test Oracles can be built from :

- **System specification :** System oracles are designed early to check the specified properties and Subsystem oracles are a part of architectural design and system build plan.
- Designs

Example 1 : A UML sequence diagram indicates a test case and expected outcome. A scenario based oracle can be used.

Example 2 : A UML state diagram indicates all permissible behaviors of a module. A oracle can be used with large numbers of automatically generated test cases.

- Code documentation

Economics of Test Oracles :

- Expected output for each test input
  - Easy to manually verify for one test input
  - Expensive to verify for many test inputs

- Properties applicable for multiple test inputs
  - Not easy to write

Syllabus Topic : Levels of Testing

11.6 Levels of Testing

V&V model illustrates different **levels of tests** at different **phases of SDLC.**

SDLC phases of the *V-model are as follows :*

- Business Requirements (BRS) and System Requirements (SRS) begin the life cycle. The test plan is created that focuses on meeting the functionality specified in the SRS.
- High-level Design (HLD) phase focuses on system architecture and design. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.
- Low-Level Design (LLD) phase is where the actual logic for each and every component of the system is designed. Class diagram with all the methods and relation between classes comes under LLD.
- Code (Implementation) phase is where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Various levels of testing are explained in below sub sections.

11.6.1 Level 1 : Unit Testing

- It is the most micro scale of testing where each and every unit of the system is tested.
- Unit testing tests the individual Chunks composed of objects, functions, procedures or subroutines, or any other small units of a Software Product.
- The unit testing is performed parallel to development of each unit.
- The idea behind unit testing is to test each small unit or say, every building block of a program before combining them to build a whole system.
- Unit testing is usually done by the developers if they are ought to dynamically test the code and it is done by the inspection team or review team if it is ought to just verify the code documents.
- Unit testing requires technical knowledge of the internal design and programming language used and the logic also needs to be strong.
  - *Objectives* :To test the function of a smallest executable unit
  - To conduct statement coverage, conditional coverage and path coverage.
  - To test exception & error handling

Done by : Programmers
Debuggers
Tracers

Done : After modules have been coded

Input : Internal application descriptions
Unit test plan

Output : Unit test report

Methods : Static testing or White box (structural) testing techniques.

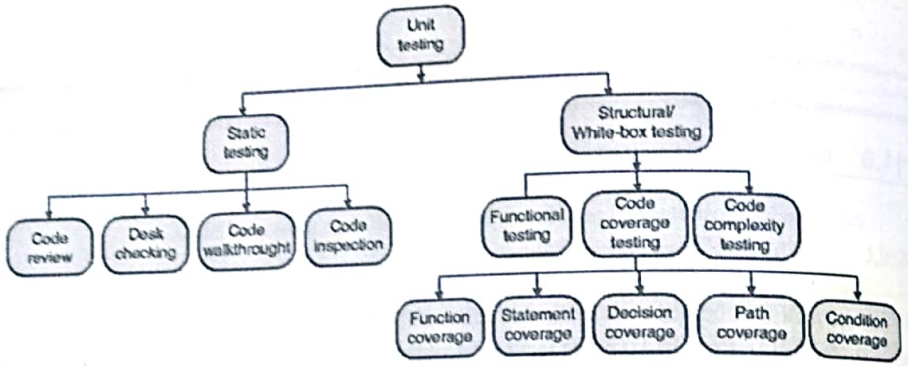


Fig. 11.6.1 : Types of Unit Testing

☞ **Advantages**

1. Unit testing covers logical analysis of software elements.
2. Unit testing allows multiple units to be tested simultaneously and also, the testing activities can proceed in parallel with development activities.
3. Unit testing is focused on testing the programming code, the finding and correcting of defects is easier.
4. Thus, unit testing helps to build an entire system with defect free units that ultimately achieve higher quality software in less time.

11.6.2 Level 2 : Component Testing

- Component testing finds the defects in the module and verifies the functioning of software.
- Component testing may be done independent from rest of the system testing. In such a case the missing software is replaced by Stubs and Drivers and simulate the interface between the software components in a simple manner.
  - o A stub is called from the software component to be tested.
  - o A driver calls the component to be tested.

Example : Suppose there is an application consisting of three modules say, module A, B, C. The developer has developed the module B and now wanted to test it. But functionality of module B depends on module A and module C. But module A and module C is not yet developed. In this case to test the module B, we should replace module A and module C by stub and drivers as required.

11.6.3 Level 3 : Integration Testing

- Integration testing is also called as Component testing because it is about testing whether the system works properly when different components are integrated to form a complete system.
- Integration is defined as set of interactions among components. Testing the interaction between modules and interaction with other systems externally is called integration testing.

- Integration testing goes through internal and external interfaces, and tests the functionality of the software.

Objectives : To verify the interfacing between modules, and within sub-systems.
Done by : Developers
Testers
Done : After modules are unit tested and also all the functions of each component are tested.
Input : Internal and external application descriptions
Integration test plan
Output : Integration test report
Methods : Top-down and bottom-up testing techniques.

- Integration testing can be classified into four broad categories depending upon the order in which the subsystems (components) are selected for testing and integration.
  1. Bottom up integration testing.
  2. Top down integration testing.
  3. Big bang integration (Non-incremental) testing.
  4. Sandwich testing.

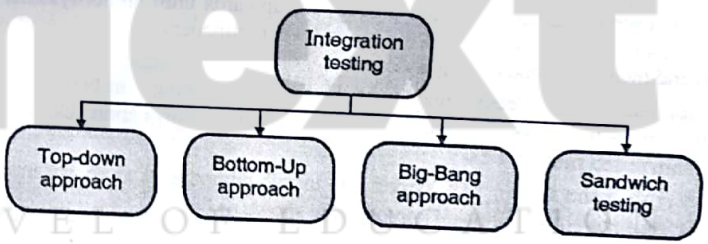


Fig. 11.6.2 : Types of Integration Testing

11.6.3(A) Top-down Testing

- This is about breaking down of a system in to its sub components so as to study its compositional sub-systems in greater detail. Consider the Fig. 11.6.3 which depicts the breaking down of a system into its sub-systems (components).

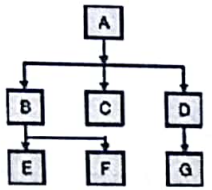


Fig. 11.6.3 : System 'A' divided into its sub components

- In top-down approach, the testers first test the top layer of the system and proceed downwards until all subsystems are incorporated into the test - Components which are at the top layer are tested first and then it is integrated with the just below components and then tested it and so on.

Example : Consider the system 'A' divided into its sub components as shown in Fig. 11.6.3. Therefore, the testing steps with top-down approach are as shown as below;

Fig. 11.6.4 : Top-down Testing

- The importance of the top – down testing is design flow is detected, quality improves, user satisfaction, requirements matches with the developed software, user view primary view of the system, then secondary view of the system and so on.

☞ **Advantages**

- Detects the flow of design.
- Useful when major flaws occur at the top of the program.

11.6.3(B) Bottom-up Testing

- In this approach, the sub systems are linked together to form larger subsystems which then in turn are linked sometimes in many levels, until a complete top-level system is formed.
- In the bottom-up approach the individual base elements i.e. the elements at the lower level of the system are first tested in great detail and proceeding upwards until all subsystems are combined together to forma a whole system i.e. all the lower level components are integrated with just above components and then combined testing is done.
- Bottom-up approach is completely opposite of the top-down testing – in bottom-up approach, the components at the bottom level are tested first; where as in top-down approach, the components at the top level are tested first.

  Example : Consider the system 'A' divided into its sub components as shown in the Fig. 11.6.3. Then, the testing steps with bottom-up approach are shown as below;

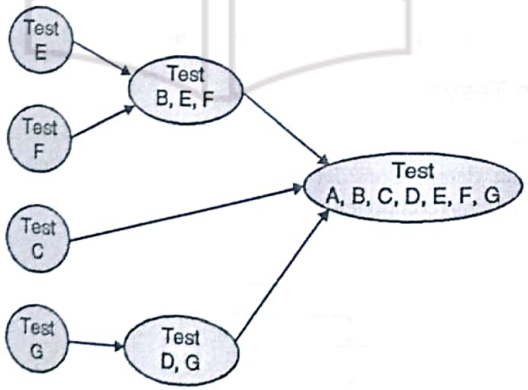


Fig. 11.6.5 : Bottom-up Testing

☞ **Test Driver**

- The test drivers are used during Bottom-up integration testing in order to simulate the behavior of the upper level modules that are not yet developed or integrated.
- Test drivers are the modules that act as temporary replacement for a calling module.

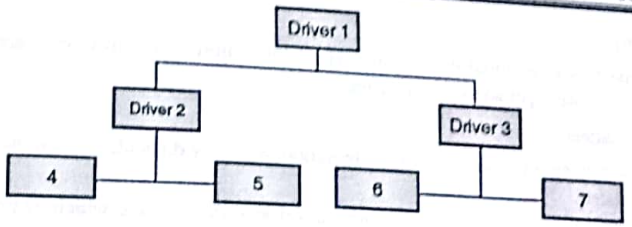


Fig. 11.6.6

- From the Fig. 11.6.6 , it is clear that the modules 4,5,6,7 cannot be integrated until the above modules are integrated. And the above modules are still under development that cannot be integrated at this point of time.

☞ **Advantages**

- Useful when major flaws occur at the bottom of the program.
- Useful for integrating the Object-oriented systems
- Test conditions are easier to create.
- Observation of test results is easier.

☞ **Drawbacks**

- Integration errors are found later than earlier.
- Design flaws that could need major reconstruction are found last.

11.6.3(C) Big-Bang Approach

- This approach is ideal for a product where the interfaces are stable with less number of defects.
- In big – bang testing, all smallest components in the product are tested individually then all of them are combined together and tested as a combinational work of the system.
- This approach is used to test how one component supports the other component.

  Example : Consider the system 'A' divided into its sub components as shown in the Fig. 11.6.3. Then, the testing steps with big-bang approach are shown as in Fig. 11.6.7;

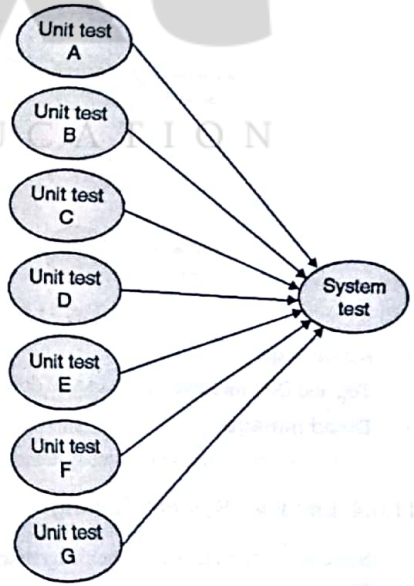


Fig. 11.6.7 : B

Advantage

This approach is well suited in scenarios where the majority of components are already available and very few components get added or modified.

Disadvantages

- When some failure is found out during integration, it is very difficult to trace the module where the problem existed.
- It is also difficult to find the developer who developed that module which again makes it difficult to make modifications so as to correct that defect.
- Certain sub-systems may take lot of time to be corrected.

11.6.3(D) Sandwich Testing

- This is the combination of top -down and bottom-up testing.
- This approach *views the system as if having three layers* and the testing converges at the target layer
  1. Target layer in the middle
  2. layer above the target
  3. layer below the target.

Example : Consider the system 'A' divided into its sub components as shown in the Fig. 11.6.3 Then, the testing with sandwich approach is shown as in Fig. 11.6.8;

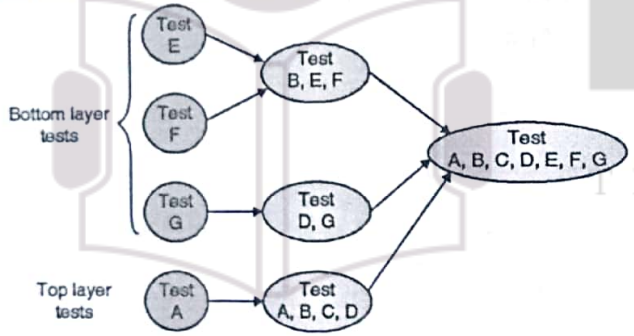


Fig. 11.6.8 : Sandwich Approach

Advantage

Top and Bottom Layer Tests can be done in parallel.

Disadvantage

Does not test the individual subsystems thoroughly before integration of the sub components.

11.6.4 Level 4 : System Testing

- System testing begins after the integration of all tested components is completed.
- The main objective of system testing is to determine if the integrated components work together as designed.
- A system is usually defined as a set of hardware, software and other parts that integrate to provide product features and solutions.

- System testing is performed on completely integrated components and solutions to estimate the system compliance with specified requirements (the requirements may be either functional or non-functional).
- This testing brings out issues that are fundamental to design, architecture and code of the whole product.

Objectives : To determine that system as a whole fulfills all functional and non-functional requirements.
Done by : Developers
Testers
Users
Done : After integration testing
Input : Detailed requirements and external application descriptions
System test plan
Output : System test report
Methods : Performance, load, stress, security testing techniques

- All the *system testing types* are ***Non-Functional Testing*** types since it is not about testing the functionality of the program code; instead, it is about testing the behaviour and security issues of the software.

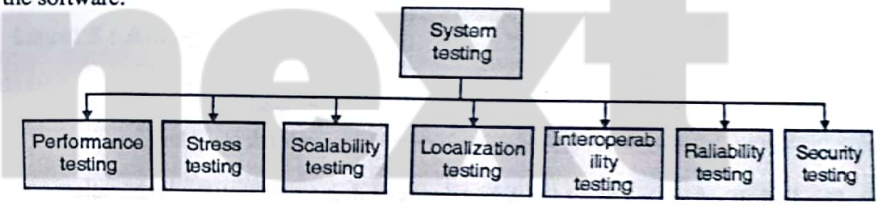


Fig. 11.6.9 : Types of System Testing

1. Performance testing

- Each and every system has implicit performance requirements such as - the software shouldn't take endless time or endless resources to execute. 'Performance bugs' are used to represent the design problems in the software may degrade the system's performance.
- Performance evaluation of a software includes resource usage, throughput, stimulus-response time and queue lengths describing the average or maximum number of tasks waiting to be serviced by selected resources (resource may be the network bandwidth, CPU cycles, memory or disk space and disk access operations).
- The goal of performance testing is to ensure that a product :
  - Number of transactions processed in any given time interval – ***Throughput.***
  - Availability of system under different load conditions – ***Availability.***
  - Time taken between request and response of a page - ***Response Time.***
  - Optimum number of resources used to accomplish a request i.e. the percentage of time a component (CPU, Channel, storage, file server) is busy - ***Capacity Planning.***

2. Stress testing

- This is done to find out whether the product's behavior degrades under extreme conditions such as maximum number of users, peak demands or extended number of operations or transactions.

- The product is over-stressed deliberately to simulate the resource crunch and to find out its behavior. It is expected to gracefully degrade on increasing the stress but the system is not expected to crash at any point of time during stress testing.
- It helps in understanding how the system behaves under extreme and realistic situations.

3. Scalability/Volume/Load testing

- This is done to find out whether the product's behavior degrades under extreme conditions such as what happens if large amounts of data are handled at a time on the same hardware and software configurations.
- This testing requires enormous amount of resource to find out the maximum capability of the system parameters.

4. Localization testing

- It is about global functioning of the product. It will verify that the application still works, even after it has been translated into a new language or adapted for a new culture.

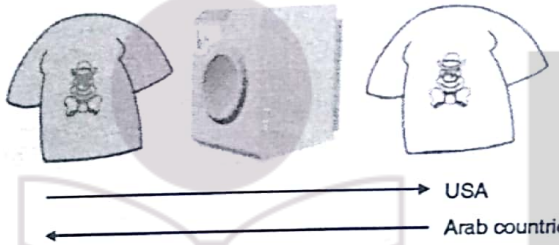


Fig. 11.6.10 : Washing Machine Advertisement demonstrating the localization concept when read from right to left and vice versa

- In the Fig. 11.6.10, you will see that the meaning of this advertisement when read from left to right is 'dirty cloth kept in washing machine gives clean cloth' but the meaning changes when it is read from right to left as is the trend of reading in Arab countries. In this case, the advertisement says, 'clean cloth kept in washing machine gives dirty cloth'. Therefore, make sure that the software works in the same manner even after it has been translated into a new language or adapted for a new culture.
- It ensures the software product should work in the same manner even if its User Interface language is transformed into any other language.

  Example : Google services the client in the same manner, no matter client opens it in English or Hindi or any other language.

5. Interoperability testing

It ensures that two or more products can exchange information, use the information, and work closely.

6. Reliability testing

To verify the ability of the sub-system or a system to confirm that it performs the user required functions without any frequent errors. It is to check that how long and how efficiently system works without any error.

7. Security testing

- Security Testing verifies that protection mechanisms built into the system will protect it from improper penetration.
- Security testing is the process of executing test cases that subvert the program's security checks.

  Example :
  - One tries to break the operating systems memory protection mechanisms
  - One tries to subvert the DBMS's data security mechanisms
- The role of the developer is to make penetration cost more than the value of the information that will be obtained.

8. Recovery testing

- It verifies whether the system recovers completely from expected or unexpected events without loss of data or functionality.
- The events that may cause data loss may be such as shortage of disk space, unexpected loss of communication or power out conditions.
- It tests how well a system recovers from crashes, hardware failures or other problems.

11.6.5 Level 5 : Acceptance Testing

- This testing is performed by end-users or representatives of the end-users. The customer defines a set of test cases that are executed to check whether the product meets the user requirements and only then, it is qualified and accepted as the product. These test cases are executed by the customers themselves in order to judge the product before deciding to buy it.
- Test cases for Acceptance testing are executed to verify if the product meets the acceptance criteria defined during the software requirement specification phase of the SDLC.

Objectives	:	To determine that the system meets the user requirements.
Done by	:	Users
Done	:	After System Testing
Input	:	Business Needs & Detailed Requirements User Acceptance Test Plan
Output	:	User Acceptance test report
Methods	:	Normally performs black-box techniques to check the system functions.

- Acceptance testing can be classified into two broad categories depending upon the place where it can be conducted;

1. Alpha testing 2. Beta testing

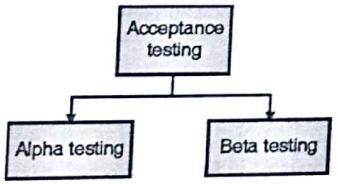


Fig. 11.6.11 : Types of Acceptance Testing

1. **Alpha testing**
   - Alpha testing is done when development is nearing to completion and some minor design changes may still be made in the system as a result of alpha testing.
   - This is conducted at the developer's site by end users. The software is used in a natural setting with the developer "looking over its shoulder" of typical users and recording errors and usage problems. Alpha tests are conducted in a controlled environment.
   - Sponsorer/customer uses the software at the developer's site.
   - Software is used in a controlled setting, with the developer always ready to fix bugs.
2. **Beta testing**
   - This is conducted at end-user sites. Unlike alpha testing, the developer is generally not present in beta testing. Therefore, beta testing is also called as 'live' testing of software as it is done in an environment where it is not controlled by any developer.
   - The end-user documents all errors that occur during beta testing and submits the report to the developer at regular intervals. As a result of this error report generated through beta testing, software engineers make modifications and then start preparing for the release of the software product.
   - Conducted at *sponsor's site* (developer is not present).
   - Software gets a realistic workout in target environ-ment.
   - Potential customer might get discouraged.

11.6.6 Integration Testing vs. System Testing

Q. Differentiate: Integration and System Testing

Table 11.6.1 : Difference Between System Testing And Integration Testing

Sr. No.	Integration Testing	System Testing
1.	When integration testing starts once the functional testing on individual components is finished.	When System testing begins after the integration of all tested components is completed.
2.	Objective : To verify the interfacing between modules, and within sub-systems	Objective : To determine if the integrated components work together as designed.
3.	Aim : Breaks down the system into sub systems or links the sub systems into a system to ensure that the interfacing of these subsystems/components/modules doesn't affect the functioning of each other and thus work properly when integrated with each other.	Aim : This testing brings out issues that are fundamental to design, architecture and code of the whole product.
4.	Methods : Top-down integration, bottom-up integration testing, big bang integration testing and sandwich testing.	Methods : Performance, load, stress, security testing techniques
5.	Integration testing is also called as Component testing because it is about testing whether the system works properly when different components are integrated to form a complete system.	All the system testing types are Non-Functional Testing types since it is not about testing the functionality of the program code; instead, it is about testing the behaviour and security issues of the software

11.7 Static Verification

- It is a ***verification*** process
- Testing a software without execution on a computer. Involves just examination/review and evaluation
- It is done to test that software confirms to its SRS i.e. user specified requirements
- It is done for ***preventing*** the defects
- Static Testing is a process of reviewing the work product and reviewing is done using a checklist
- Static Testing helps weed out many errors/bugs at an early stage
- Static Testing lays strict emphasis on conforming to specifications
- Static Testing can discover dead codes, infinite loops, uninitialized and unused variables, standard violations and is effective in finding 30-70% of errors.

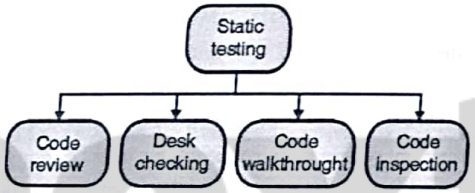


Fig. 11.7.1

11.7.1 Code Review

Q. Define the terms :Code Review

- Code reviews are extremely cost-effective strategies to reduce coding errors and produce high quality code.

Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned? Is there any possibility of buffer overflow?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for? If a break is required after each case in case statements, has it been included?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output? Can unexpected inputs cause corruption?

- Normally two types of code reviews are carried out on the code of the module: *code walk-through* and *code inspection.*

11.7.2 Desk Checking

Q. Define the terms : Desk-checking

- This is done by authors of the code. It is a method to verify the portions of the code for correctness. Such verification is done by comparing the code with the design or specification to make sure that the code does what it is supposed to do and effectively. This method of catching and correcting errors is characterized by :
- No structured method or formalism to ensure completeness
- No maintaining of a log or checklist.
- This method completely depends on authors'/developers' thoroughness, attentiveness and skills.

Advantages

- The programmer who is good in programming language is able to read and understand his system's code.
- As the desk checking is done by an individual, there are fewer scheduling and logistic expenses.

Disadvantages

- A developer is not the right person to detect problems in his or her own code.
- Developers generally prefer to write new code rather than do any form of testing.
- This method is essentially person-dependent and informal and thus may not work consistently across all developers.

11.7.3 Code Walk-Through

Q. Define the terms : Walk-through

- Code walk-through is an informal code of analysis technique. In this technique after a module has been coded, it is successfully compiled and all syntax errors are eliminated.
- In walkthroughs, a set of people look at the program code and answer the questions. If the author is unable to answer some questions, he or she then takes those questions and finds their answers.
- Some members of the development team are given the code a few days before the walkthrough meeting to read and understand the code.
- Each member selects some test cases and simulates execution of the code by hand.
- The main objectives of walk-through are to discover the algorithmic and logical errors in the code.

Guidelines to perform code walkthrough

- The team performing the code walk-through should not be either too big or too small. Ideally, it should consist of three to seven members.
- Discussion should focus on discovery of errors and not on how to fix the discovered errors.
- In order to promote cooperation and to avoid the feeling among the engineers that they are being evaluated in the code walk through meeting, managers should not attend the walk through meetings.

Advantage over desk checking

Walkthroughs bring multiple perspectives.

Disadvantage

Completeness is limited to the area where questions are raised by the team.

11.7.4 Code Inspection

Q. Define the terms : Inspection

- The inspection takes place only when the author has made sure the code is ready for inspection by performing some basic desk checking and walkthroughs.
- *List of some programming errors which can be checked during code inspection* :
  - uninitialized variables
  - Non terminating loops
  - Array indices out of bounds
  - Improper storage allocation and deallocation
  - Reading a file that doesn't exist
  - Arithmetic exception i.e. Divide by zero
  - Mismatch between function definition and function call due to mismatch between actual and formal parameters passed.

Inspections and Testing

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V and V process.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

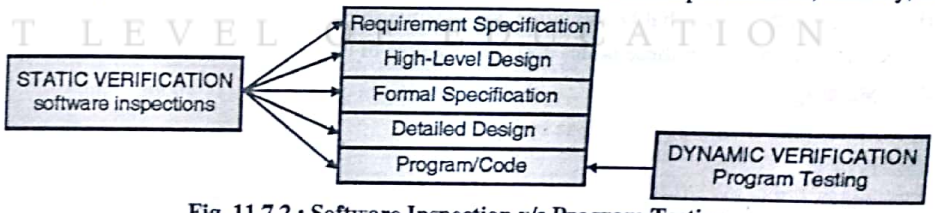


Fig. 11.7.2 : Software Inspection v/s Program Testing

Inspection pre-conditions

- A precise specification must be available.
- Team members must be familiar with the organisation standards.
- Syntactically correct code or other system representations must be available.
- An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process.
- Management should not use inspections for staff appraisal i.e. finding out who makes mistakes.

Inspection Process

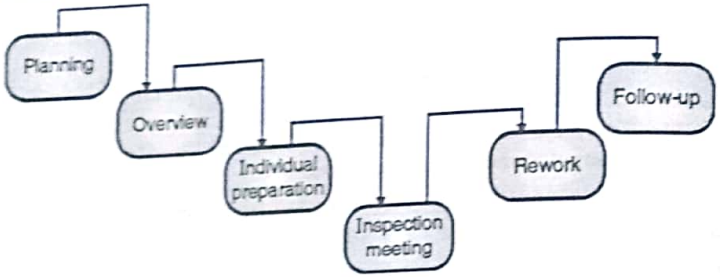


Fig. 11.7.3 : Inspection process

Step 1 : Planning

- Select the group review team - three to five people group is best
- Identify the moderator - has the main responsibility for the inspection
- Prepare work product for review plus supporting docs

Step 2 : Overview

- A brief meeting - explain purpose of the review, intro.
- All team members then individually review the work product
- Lists the issues/problems they find in the self-preparation log
- Checklists, guidelines are used

Step 3 : Preparation

- Each reviewer studies the project individually.
- He notes down the issues that he has come across while studying the project.
- He decides how to put up these issues and makes a note of it.

Step 4 : Meeting

- A reviewer goes over the product line by line
- Others raise the issues
- Discussion follows to identify if a defect
- Scribe records the issues
- If few defects, the work product is accepted; else it might be asked for another review
- Group does not propose solutions - though some suggestions may be recorded

Step 5 and 6 : Rework and Follow Up

- Defects in the defects list are fixed later by the Author. These modifications are made to repair the discovered errors.
- Once fixed, author gets it OKed by the moderator, or goes for another review
- Re-inspection may or may not be required.
- Once all defects/issues are satisfactorily addressed and reviewed, the collected data is submitted.

Inspection Roles

There are four roles in inspection: ***author*** of the code, ***moderator*** who is expected to formally run the inspection according to the process, ***inspectors*** who actually provide review comments for the code. Finally, there is a ***scribe*** who takes detailed notes during the inspection meeting and circulates them to the inspection team after the meeting.

Author or Owner	The programmer who develops and fixes the code
Inspector	Finds errors, omissions and inconsistencies in programs and documents.
Reader	Presents the code an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or Moderator	facilitates the inspection meeting.
Chief Moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Advantages

The goal of this method is to detect all faults, violations, and other side-effects. This method finds out more and more number of defects by:

- Demanding thorough preparation before an inspection.
- Enlisting multiple diverse views.
- Assigning specific roles to the multiple participants
- Going sequentially through the code in a structured manner.

Disadvantages

- Time consuming as it needs preparation as well as formal meetings.
- Logistics and Scheduling can become an issue since multiple people are involved.
- It is not always possible to go through every line of code with several parameters and their combinations to ensure the correctness of the logic, side side-effects and appropriate error handling.

Syllabus Topic : White-box Testing / Structural Testing

11.8 White-box Testing (Structural Testing)

- It is also called as 'Glass Box testing' because all the internal coding is clearly seen and tested. The internal structure of the system is as transparent and as clear as glass box and that's why it is called so. This is the same reason why it is also called as white-box testing.

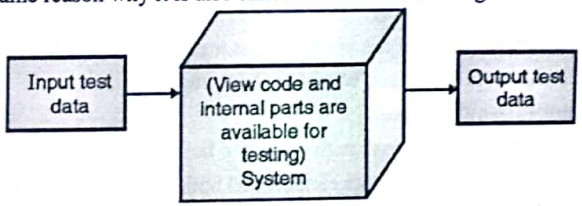


Fig. 11.8.1 : White Box Testing

- White Box Testing involves the complete knowledge of internal structure of source code and it can be used to find the number of test cases required to guarantee a given level of test coverage.
- In white-box testing (sometimes called clear-box testing), the software tester has access to the program's code and can examine it for clues to help him with his testing- he can see inside the box.
- It is error based testing. This kind of testing searches the given class's method for particular choice of interest and then describes how these choices to be tested.
- White Box test covers following contents;
  - Logical Analysis of software elements.
  - Possible paths of control flow.
- Software is viewed as a white-box, or glass-box in white-box testing, as the structure and flow of the software under test are visible to the tester.

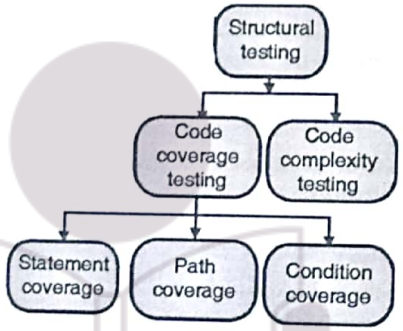


Fig. 11.8.2: Types of White box Testing

- White-box testing deals with the code, code structure, and internal design and how they are coded.
- Structural testing is actually run on the built product while Static testing is performed by humans just on the source code and not the executables.
- White-box testing needs knowledge of the internal program design and code required and these type of tests are based on coverage of code statements, branches, paths and conditions.
- Therefore, the structural tests are logic driven.

☞ **Structural testing involves**

- Testing all independent paths at least once
- Testing all logical decisions on their *true* and *false* sides
- Testing all loops at their boundaries and within their operational bounds
- Testing internal data structures to ensure their validity

☞ **Advantages of white-box Testing**

- The internal structure (code) of the application is tested thoroughly to ensure the validity.
- Forces the test developer to reason carefully about implementation
- Reveals the errors in hidden code.
- Spots the dead code or other issues with respect to best programming practices.
- Does the logical analysis of software elements on both true and false sides
- Tests all possible paths of control flow.

11.8.1 Code Coverage Testing

- This involves designing and executing test cases and finding out the percentage of code that is covered by testing.
- It describes the degree to which the source code of a program has been tested.

☞ **Various code coverage criteria**

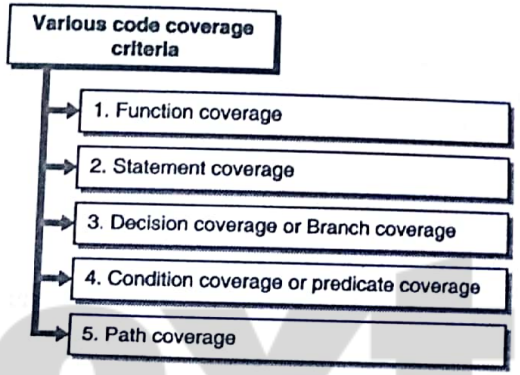


Fig. C11.1 : Various code coverage criteria

➜ **1. Function coverage**

It tests whether each function or subroutine in the program has been called.

Function Coverage = (Number of Functions Exercised/Total number of functions in program)* 100

➜ **2. Statement coverage**

- It tests whether each statement in the program has been executed or not.
- Executes all statements at least once

Statement Coverage = (Number of Statements Exercised/Total number of executable statements in program) * 100

Example :

Assume, program has 100 statements
Tests exercise 87 statements then,
Statement coverage = 87%

Example :

Statement No.	Program
1.	Read(A)
2.	If A > 5 Then
3.	B = A + 1
4.	End-If
5.	Print B

Test Case Id	Input	Expected Output
1	8	9

In this example, as all 5 statements are 'covered' by this test case, we have achieved 100% *statement coverage*

➜ 3. **Decision coverage or Branch coverage**

Has every edge in the program been executed? Such as in IF and CASE statements.

Executes each decision direction at least once.

Decision Coverage= (Number of Decision outcomes Exercised/Total number of Decision outcomes in program)* 100

Example

Assume, program has 100 decision outcomes

Tests exercise 60 decision outcomes then,

Decision coverage = 60%

Example

Statement No.	Program
1	Read(A)
2	If A < 10 or A > 20 Then
3	B = A + 1
4	End-If
5	Print B

Test Case Id	Input	Expected Output
1	8	9
2	22	23

➜ 4. **Condition coverage or predicate coverage**

- Has each Boolean sub-expression evaluated both to true and false? This does not imply decision coverage.
- For example, when there is an OR condition, if first condition is true, the second is not executed. Similarly, in AND condition if the first part is found false, then the second part is not at all evaluated. For these reasons, it is necessary to test each Boolean expression and have test cases to test the true as well as false paths.

Condition Coverage= (Total decisions exercised/Total number of decision in program) * 100

Example 1

A = 10
B = 15
If (A > 3) or (A < B) Then
B = X + Y
End If

Example 2

While (A > 0) and (Not EOF) Do
Read (X)
A = A - 1
End-While-Do

➜ 5. **Path coverage**

- It involves splitting the program into a number of distinct paths. A program can start and take any of the paths to reach the end.
- It executes all possible combinations of condition outcomes in each decision.

Path Coverage= (Total Paths exercised/Total number of executable paths in program) * 100

Example

1. Insert the card
2. IF card is valid THEN
3. display "Enter PIN"
4. IF PIN is valid THEN
5. select AccountType
6. ELSE
7. display "Invalid PIN"
8. ELSE
9. Reject card
10. END

Various paths possible are:

Path 1: 1, 2, 8, 9, 10

Path 2: 1, 2, 3, 4, 6, 7, 10

Path 3: 1, 2, 3, 4, 5, 10

Therefore, minimum path coverage tests to be achieved = 3 since there are three paths.

Example : Consider the following function :

```
int grt(int x, int y)
{
    int z = 0;
    if ((x>0) && (y>0))
    {
        z = x;
    }
    return z;
}
```

- If the function 'grt' was called at least once, then *function coverage* is satisfied.
- *Statement coverage* for 'grt' function is satisfied if every line in the function is executed.
- The grt (1,1) and grt (1,0) will satisfy *decision coverage*, as in the first case the `if` condition is TRUE and in the later it is FALSE.
- *Condition coverage* can be satisfied with tests that call grt (1,1), grt (1,0) and grt (0,0). These are necessary as in the first two cases (x>0) evaluates to TRUE while in the third it evaluates to FALSE. At the same time, the first case makes (y>0) TRUE and in the later two cases it is FALSE.

11.8.2 Code Complexity Testing

Q. Define the terms : Code complexity testing (Cyclomatic complexity)

Cyclomatic Complexity (CC) testing is a process that measures the complexity of a program. In this process, a program is represented through a flowchart – it is then converted to a flow graph containing only the nodes and edges. Steps to convert the flow chart into flow graph are listed down :

Step 1 : Identify the predicates (decision points) in the flow chart.
Step 2 : Ensure that the predicates are simple else break up a condition into simple predicates.

Step 3 : Combine all the sequential statements into a single node. The reason is that all these statements get executed once started.

Step 4 : If a set of sequential statements is followed by a simple predicate, combine all the sequential statements and the predicate check into one node and have two edges emanating from this one node. Such nodes with two edges emanating from them are called predicate nodes.

Step 5 : Make sure that all the edges terminate at some node. Add a node to represent all the sets of sequential statements at the end of the program.

Calculating CC :

- CC = P +1 // where P is Number of Predicates
- CC = E-N+2 // where E is Number of Edges & N is Number of Nodes
- CC = Count of Regions in Flowgraph

Example 1 :

```
Void check()
{ if (ch>5)}
        flg=1
else if(ch=5)
        flg=flg-5
else if(ch>2)
        flg-flg-2
        else
        flg=0
}
```

Solution :

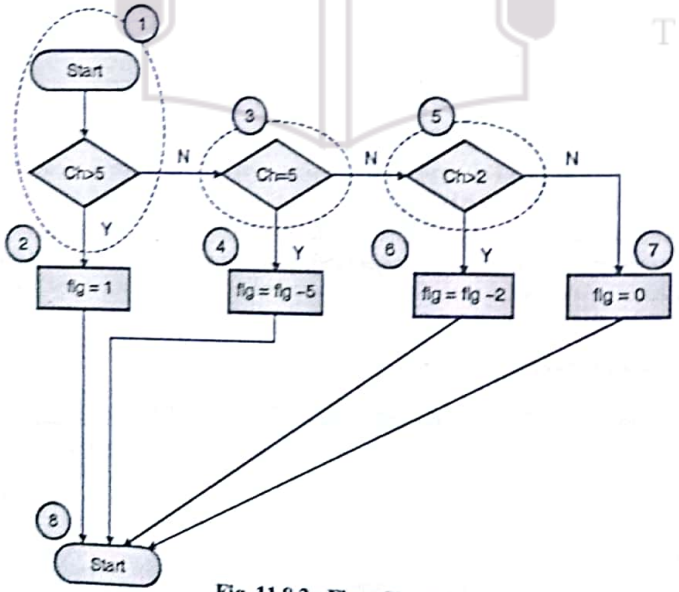


Fig. 11.8.3 : Flow Chart

CC = P + 1 = 3 + 1 = 4

CC = E – N + 2 = 10 – 8 + 2 = 4

CC = Count of Regions = 4

Number of independent paths to be tested using Path coverage are :

Path 1 : 1-2-8

Path 2 : 1-3-4-8

Path 3 : 1-3-5-6-8

Path 4 : 1-3-5-7-8

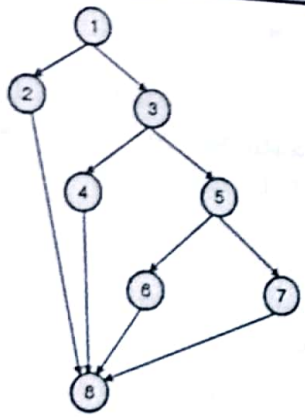


Fig. 11.8.4 : Flow graph

Test Case

TestCase Id	TestCase Condition	Input	Expected Output
TC_01	To validate path 1-2-8	ch=7	flg=1
TC_02	To validate path 1-3-4-8	ch=5	flg=flg-5
TC_03	To validate path 1-3-5-6-8	ch=3	flg=flg-2
TC_04	To validate path 1-3-5-7-8	ch=0	flg=0

Example 2

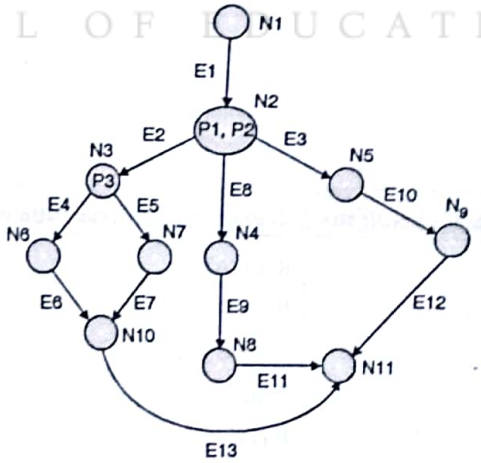


Fig. 11.8.5

CC = P + 1 = 3 + 1 = 4

CC = E – N + 2 = 13 – 11 + 2 = 2 + 2 = 4

CC = Count of Regions = 4

Example 3 :

1. Insert the card
2. IF card is valid THEN
3. display "Enter PIN"
4. IF PIN is valid THEN
5. select AccountType
6. ELSE
7. display "Invalid PIN"
8. ELSE
9. Reject card
10. END

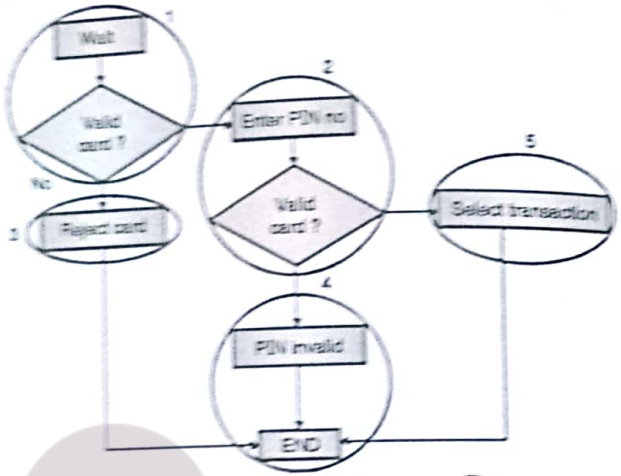


Fig. 11.8.6

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are : 10

Number of Conditions (predicates) to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are:

Path 1: 1 3 6 7
Path 2: 1 2 4 6 7
Path 3: 1 2 5 6 7

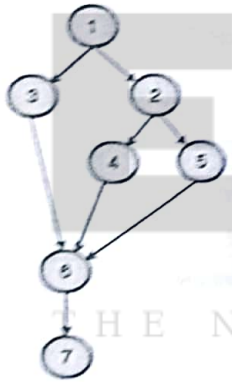


Fig. 11.8.7

Example 4 :

Program with complex predicates	Converted to Program with simple predicates

Read (A)
If A > 0 and A < 5 Then
Print "A"
End If
	Read (A)
	If A > 0 Then
	If A < 5 Then
	Print "A"
	End If
	End If

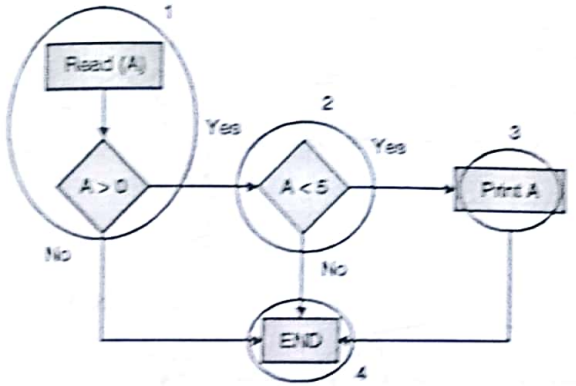


Fig. 11.8.8

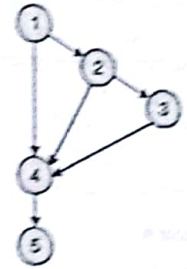


Fig. 11.8.9

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are : 6

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are : 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are:

Path 1: 1 4 5
Path 2: 1 2 4 5
Path 3: 1 2 3 4 5

Example 5

1. Read A
2. Read B
3. IF A > 0 THEN
4. IF B = 0 THEN
5. Print "No values"
6. ELSE
7. Print B
8. IF A > 21 THEN
9. Print A
10. ENDIF
11. ENDIF
12. ENDIF

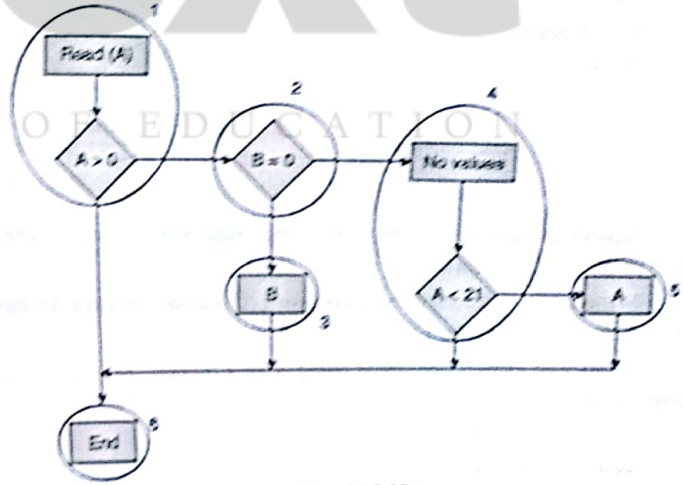


Fig. 11.8.10

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are: 12

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 3

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 6 7

Path 2: 1 2 3 6 7

Path 3: 1 2 4 6 7

Path 4; 1 2 4 5 6 7

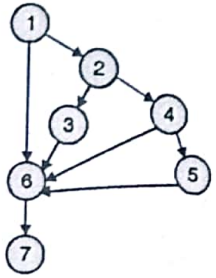


Fig. 11.8.11

Example 6

1. Read A
2. Read B
3. IF A < 0 THEN
4. Print "A negative"
5. ELSE
6. Print "A positive"
7. ENDIF
8. IF B < 0 THEN
9. Print "B negative"
10. ELSE
11. Print "B positive"
12. ENDIF

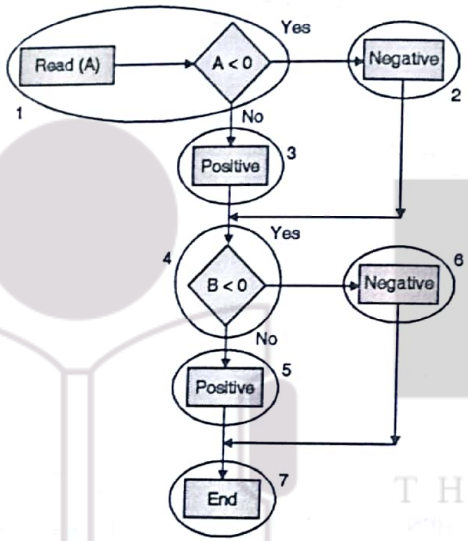


Fig. 11.8.12

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are: 12

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 4 5 7 8

Path 2: 1 3 4 6 7 8

Path 3: 1 2 4 5 7 8

Path 4; 1 2 4 6 7 8

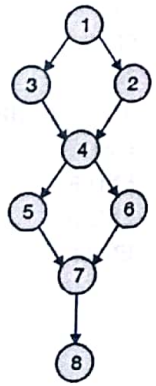


Fig. 11.8.13

Example 7 : Check the cyclometric complexity for a program of adding 100 integers. It should also check for valid boundaries and valid values. Design the Test Cases for such code.

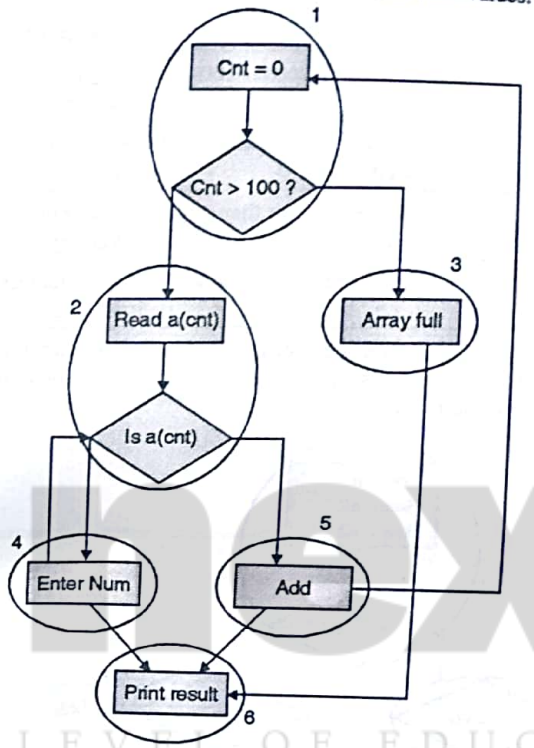


Fig. 11.8.14

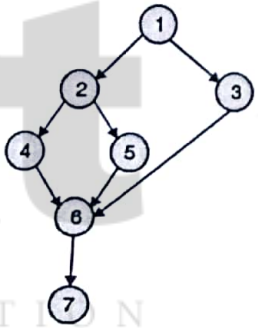


Fig. 11.8.15

Number of Statements to be tested using 'Statement Coverage Technique' derived from the program structure are: 8

Number of Conditions to be tested using 'Condition Coverage Technique' derived from the flow chart are: 2

Number of Paths to be tested using 'Path Coverage Technique' derived from the flow graph are;

Path 1: 1 3 6 7

Path 2: 1 2 4 6 7

Path 3: 1 2 5 6 7

Test description

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
No. of values in array must be 100 integers to perform addition	= 100 numbers	< 100 numbers > 100 numbers Non digit	100 numbers	99 numbers 101 numbers

Test case

Test Case Id	Conditions	Input	Expected Output
1	No. of values in array must be 100 integers to perform addition	Values till count =100	Addition of Numbers
2		Values less than count =100	Accept more numbers
3		Values greater than count =100	Array Full Warning
4	Non digits not acceptable	Any character or symbol	Enter number warning

Example 8 : A program reads three integer values as three sides of a triangle. The program prints a message indicating that whether the triangle is right angle triangle or equilateral triangle. Draw the flow graph, calculate cyclometric complexity.

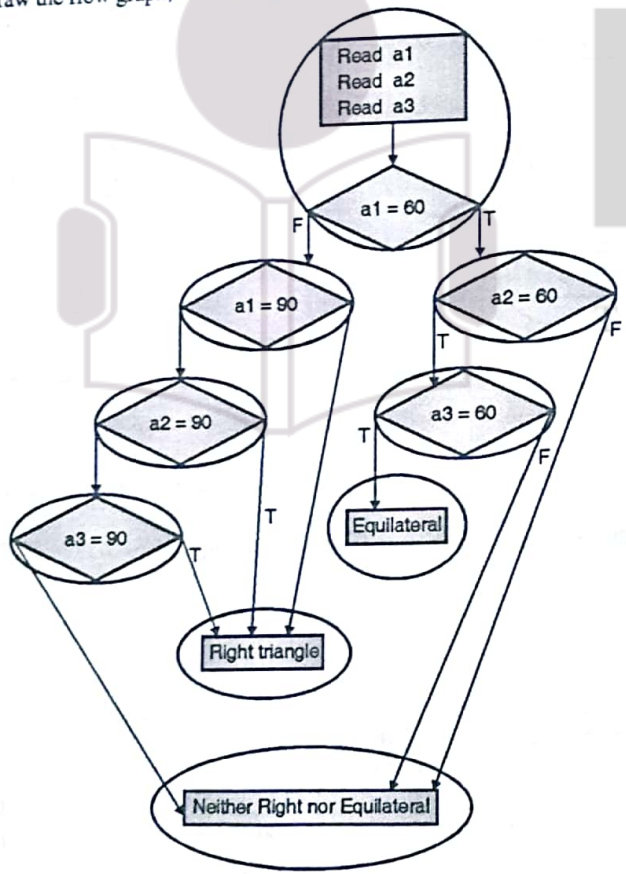


Fig. 11.8.16

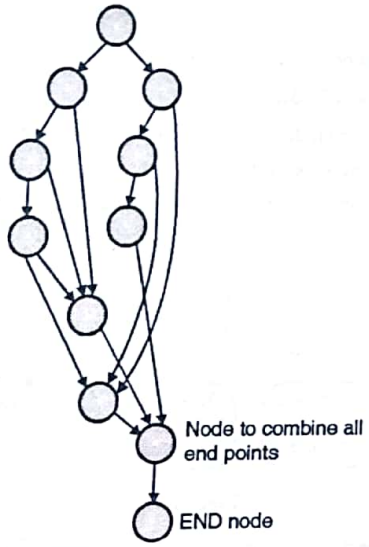


Fig. 11.8.17

Syllabus Topic : Functional / Black Box Testing

11.9 Functional (Black Box) Testing

- It is also called as '*Behavioral testing*' because it only checks the behavior of the system for certain inputs and doesn't think about the internal coding.
- It is concerned with testing the functions of the system. Therefore, it is also called as *Functional Testing*.
  - Black Box Testing involves only the observation of the output for certain input values, and there is no attempt to analyze the programming code that enables to arrive at those outputs. Therefore, it is also termed *data-driven, input/output driven* or *requirements-based testing*.

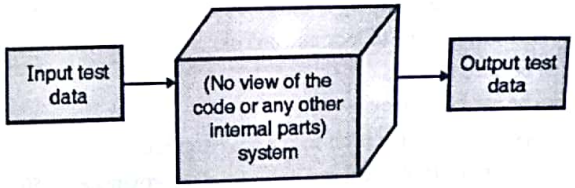


Fig. 11.9.1 : Black Box Testing

Advantages of Blackbox Testing

- Tester does not need any specific programming language knowledge.
- Black-box testing is about checking the data i.e. output data for specific input data so tester need not have any logical knowledge. He just has to check that the outputs are as expected by the user.

- Black-box tester works from the point of view of the user, not the designer.
- Black-box test cases can be designed as soon as the requirement specifications are complete.

Disadvantages of Blackbox Testing

- The tests are sometimes redundant if the software designer has already run a test case.
- The test cases are difficult to design.
- Testing every possible input is not possible because it would take a lot of time and therefore, many of the program paths go untested.

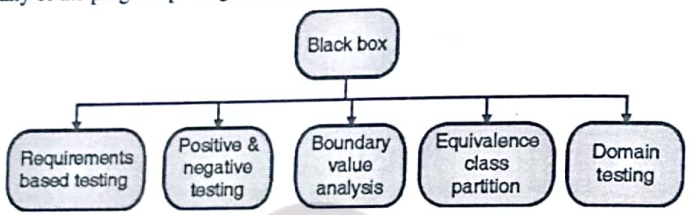


Fig. 11.9.2

11.9.1 Requirements based Testing

- It deals with validating the requirements given in the SRS of the software system.
- Actually, not all the requirements are stated by the users in the SRS - The requirements that are stated explicitly by the users in the SRS are known as ***explicit requirements*** and the requirements that are not stated in the SRS but are assumed to be incorporated in the system are known as ***implicit requirements***.
- This testing ensures that the requirements documented in SRS are consistent, correct, complete and testable.
- This testing also allows combining both explicit and implicit requirements and documenting it together in a single document known as *Test Requirements Specification*.

Example : A black-box test case can be derived from the following requirements for a 'Lock and Key' :

- Inserting key numbered 4321 and turning it clockwise should facilitate locking.
- Inserting key numbered 4321 and turning it anti clockwise should facilitate unlocking.
- Only key number 4321 can be used to lock and unlock.
- No other object can be used to lock.
- No other object can be used to unlock.
- The lock must not open even when it is hit with a heavy object.
- The lock and key must be made of metal and must weigh approximately 150 grams.

11.9.2 Positive and Negative Testing

Positive testing

- It is performed to prove that a given product exactly does what it is expected to do.
- The need of this type of testing is to verify the known test conditions.
- When a test case tests the software to obtain a set of expected output, it is called as positive test case.

- A positive test is done to confirm that the product works as per specification and user expectations.
- It also tests that whether the product gives an error or not when it is expected to give an error. This is also a process of positive testing.

Negative testing

- It is performed to prove that the product does not fail when an unexpected input is given.
- The need of this type of testing is to break the system with unknown inputs because it is important to be aware of the negative situations that may occur at the end-user level and so that the application can be tested and made error proof before delivery to the client.
- A negative test is about testing a product for not delivering an error when it is expected to or delivering an error when it is not expected to.

Example

Sr. No	Positive Testing for 'Lock and Key'	Negative Testing for 'Lock and Key'
1.	Inserting key numbered 4321 and turning it clockwise should facilitate locking.	Insert some other lock's key and turning it clockwise or anti-clock wise should **not** change the current status of the lock i.e. if the lock is already locked then it must not be unlocked using some other key's lock and if it is unlocked then it must not be locked.
2.	Inserting key numbered 4321 and turning it anti clockwise should facilitate unlocking.	Try a thin piece of wire and turning it clockwise or anti-clock wise should **not** change the current status of the lock.
3.	Inserting a hairpin and turning it clockwise or anti-clockwise should have no change in the current status of the lock.	Hitting with a stone should **not** unlock the Lock.

11.9.3 Boundary Value Analysis

Q. Define the terms : BVA (Boundary Value Analysis).

- Most of the defects in software products hover around the conditions and boundaries. By boundaries, we mean 'limits' of values of the various variables. Few errors while defining limits are :
  - Programmer's tentativeness in using the right comparison operator, for example, whether to use the <= operator or < operator when trying to make comparisons.
  - Confusion caused by the availability of multiple ways to implement loops and condition checking.
  - The requirements are not clearly understood, especially around the boundaries thus causing even the correctly coded program to not perform the correct way.
- BVA is useful in catching defects that happen at boundaries.

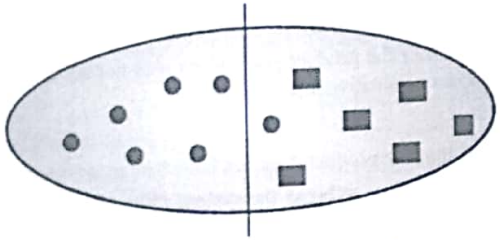


Fig. 11.9.3 : A defect at the boundary

Example : Consider a Grocery store that offers different pricing for people buying in different quantities and this pricing is different in different slabs.

No. Of Units bought	Price per Unit
First five units (i.e. from 1 to 5 units)	Rs. 5.00
Next five units (i.e. from 6 to 10 units)	Rs. 3.00
Next five units (i.e. from 11 to 15 units)	Rs. 2.00

The above table describes that :

- if you buy 4 units then it will cost 4 * 5 = Rs. 20
- if you buy 7 units, then for first 5 units, it cost 5*5=R.25 and for next 2 units it will cost 2*3 =Rs. 6 and therefore 7 units in all costs Rs.25 + Rs.6 + Rs.6 = Rs.31
- if you buy 13 units then for first five units it will cost 5*5=Rs25, next 5 units will cost 5*3=Rs.15 and next 3 units will cost 3*2=Rs.6 and therefore 13 units in all costs Rs.25 + Rs.15 + Rs.6 = Rs. 46
- The most possible defects in such ranges arise around the boundaries. In the above example, the defects may arise while buying 4, 5, 6, 7, 9, 10, 11, 14, 15 units.

11.9.4 Equivalence Class Partition

Q. Define the terms : ECP (Equivalence Class Partition).

- This technique identifies a small set of representative input values that produce as many different output conditions as possible.
- This reduces the number of permutations and combinations of input, output values used for testing. Thereby increasing the coverage and reducing the effort involved in testing.
- The set of input values that generate one single expected output is called a partition. When the behaviour of the software is the same for a set of values then the set is termed as an equivalence class or a partition.

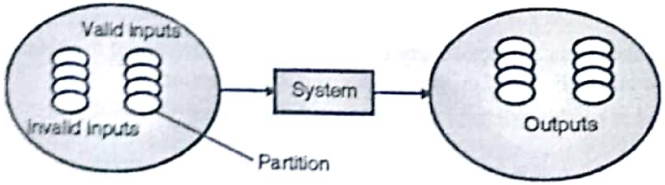


Fig. 11.9.4 : Equivalence class partition

- Testing by this technique involves :
  - Identifying all partitions for the complete set of input, output values for a product.
  - Picking up one member value from each partition for testing to maximize complete coverage.

Example 1 : Consider an example of an insurance company that has the following premium rates based on the age group. This life insurance company has base premium of Rs.1.00 for all ages but the additional monthly premium differs based on their age groups. Therefore, the total premium paid by a person is calculated as base premium + additional monthly premium.

Age group	Additional premium
Under 30	Rs. 2.00
30 - 75	Rs. 4.00
Above 75	Rs. 8.00

Therefore, the *equivalence class partitions* obtained from the above example is based on the age criteria are as given below;

- Below 30 years – valid input
- Between 30 and 75 – valid input
- Above 75 – valid input
- Age as 0 – invalid input
- Negative age – invalid input
- Age in three digit number – valid input

Example 2 : Illustrates BVA and ECP : consider the below form;

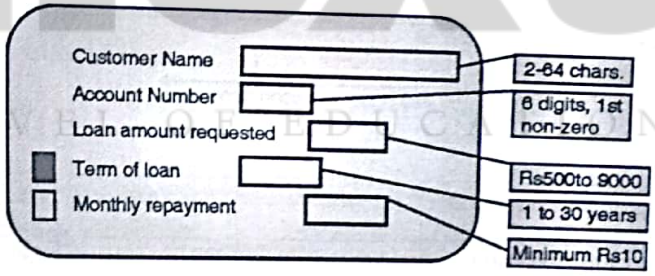
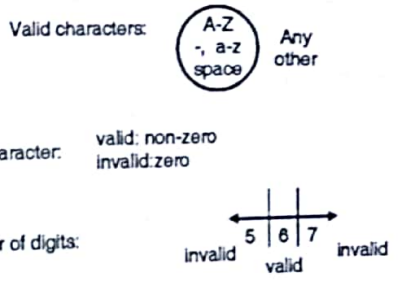


Fig. 11.9.5

The boundaries for different fields are as follows;

1. **Customer Name**



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Customer Name	2 to 64 characters	< 2 chars > 64 chars Invalid characters	2 chars 64 chars	1 char 65 char 0 char

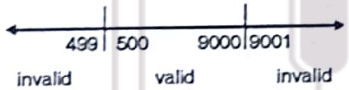
2. **Account Number**

First character: valid: non-zero
invalid:zero

Number of digits:

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Account Number	6 digits First non zero	< 6 digits > 6 digits First digit zero Non digit	100000 999999	5 digits 7 digits 0 digit

3. **Loan Amount**



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Loan Amount	Rs. 500 – 9000	< 500 > 9000 0 Non numeric null	500 9000	499 9001

4. **Term of Loan**

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Term of Loan	1-30 years	< 1 year > 1 year Non digit	1 30	0 31

5. **Monthly Repayment**

Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Monthly Repayment	Rs. 10 or more	< 10 Non digit	10	9

Advantages of ECP

- ECP reduces the huge (infinite) set of possible test cases into much smaller, but still equally effective, set.
- This technique allows in achieving good code coverage with very small number of test cases. Say, if there is an error in one value in a partition, then it can affect all the values of that specific partition.

11.9.5 Domain Testing

- This technique is purely based on domain knowledge and expertise in the domain of application. This approach needs understanding of day-to-day business activities for which the software is written.
- The testers for performing this type of testing are selected based on their in-depth knowledge of the business domain. Sometimes, it is easier to hire the testers from the domain area and train them in software, rather than take software professionals and train them in business domain.
- Domain testing is the ability to design and execute test cases that relate to the people who will buy and use the software. It is also called as business vertical testing.

11.9.6 Cause Effect Graphing

- It is a systematic approach to selecting a high-yield set of test cases that explore combinations of input conditions.
- It is a rigorous method for transforming a natural language specification into a formal language specification and exposes incompleteness and ambiguities in the specification.
- This establishes a relation between the causes and effects.
- The causes are the inputs and the effects are the outputs.
- These causes and effects are represented using basic symbols called edges and nodes. These basic symbols represent interdependency between the inputs and outputs.
- The Fig. 11.9.6 depicts various relations between the causes and effects.

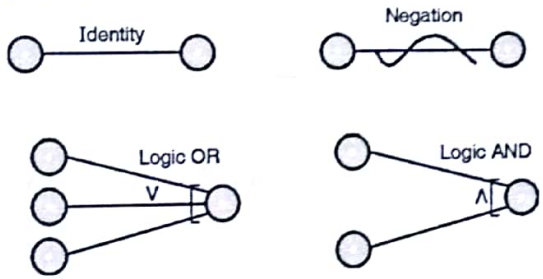


Fig. 11.9.6 : Symbols used in cause-effect graph

Methodology to draw cause effect graph

Step 1 : Decompose the system based on their functions.

Step 2 : Identify the causes also called as conditions

Step 3 : Identify the effects also called as output conditions or actions on the basis of certain conditions

Step 4 : Establish the graph of relations between causes and effects.

Step 5 : Complete the graph by adding the constraints between the causes and effects.

Step 6 : Convert the graph to a decision table and assign true in sequence to all effects

Step 7 : the columns in the decision able are converted into test cases.

Example : Cause-effect graph for 'Employees are given bonus depending upon their appointment status'.

Condition/Causes	Action/Effects
Administrative staff - Permanent	Bonus of 2 months salary
Administrative staff - Temporary	Bonus of 1 month salary
Workers - Permanent	Bonus of 1 month salary
Workers - Temporary	Bonus of ½ month salary

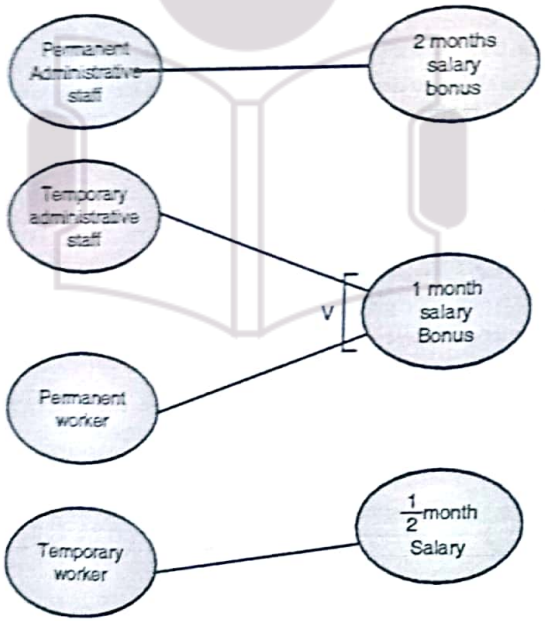


Fig. 11.9.7 : Cause-Effect graph for Employee Bonus

Example : Cause-effect graph for 'Discounts depending upon the size of order'.

Condition	Action
Order size: Over Rs. 5000	5% discount from invoice total
Rs. 2000 to Rs. 5000	3% discount from invoice total
Below Rs. 2000	Pay total invoice amount

Note that the discounts are provided only for 15 days.

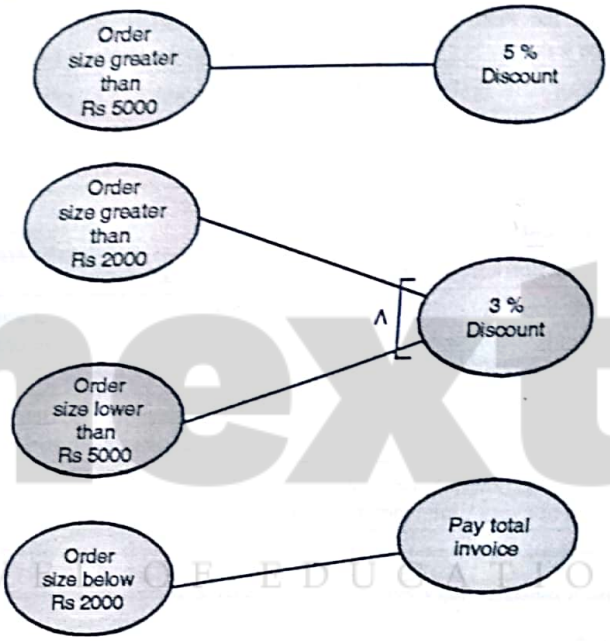


Fig. 11.9.8 : Cause-Effect graph for Discounts

Advantages

- Cause-effect graphs are very useful in the business problems where there are more than one conditions (causes).
- Cause-effect graphs also identify the critical decision processes i.e. they select the effects corresponding to causes.
- Cause-effect graphs describe all the data used in decision making so that the system can be designed to produce the data properly.

Drawbacks

- Cause-effect graphs for a complex system having many combinations of conditions will look very cumbersome.
- Large numbers of branches that represent various combinations of conditions make it difficult for the analysts to understand the formulation of some specific decisions.

11.10 Black Box vs. White Box

Q. Differentiate: whitebox and blackbox testing

Sr. No.	Black-box (Functional) Testing	White-box (Structural) Testing
1	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester
2	Knowledge of Programming language is not required	Deep Knowledge of programming language is required
3	Implementation knowledge is not required	Implementation knowledge is required
4	The tester examines the application's external functional behavior and GUI features.	The tester has to correct the code also.
5	done by independent Software Testers	done by developers
6	Black box can be applicable at all levels but dominates at higher levels of testing.	White box dominates at lower levels.
7	Requirement specification is the basis for conducting black box test	Detail design specification is the basis for conducting white-box test
8	in black box testing, sample test cases are selected and applied every time when required.	We have to write large quantity of test cases for white-box testing.

Syllabus Topic : Test Plan

11.11 Test Plan

- Test Plan tells **"What to Test"**
- The purpose of the Test Plan document is to :
  - Specify the approach to test the product
  - It specifies the deliverables extracted from the Test Approach.
  - Breakdown the product and identify features to be tested.
  - Specify the test strategies
  - Indicate the test tools to be used.
  - List the resource and scheduling plans.
  - Indicate the contact persons responsible for various areas of the project
  - Identify risks and contingency plans.
  - Specify bug management procedures for the project.
  - Specify criteria for acceptance of the product.

11.11.1 Test Plan Template

(Name of the Product)

Prepared by :

(Names of Preparers)

(Date)

Table of Contents :

INTRODUCTION :

1. TEST PLAN OBJECTIVES

- Describe the objectives supported by the Master Test Plan, eg., defining tasks and responsibilities, vehicle for communication, document to be used as a service level agreement, etc.
- [... A primary objective of testing application systems is to: assure that the system meets the full requirements, including quality requirements. At the end of the project development cycle, the user should find that the project has met or exceeded all of their expectations as detailed in the requirements.
- The secondary objective of testing application systems will be to: identify and expose all issues and associated risks, communicate all known issues to the project team, and ensure that all issues are addressed in an appropriate matter before release. ...]

2. SCOPE

2.1 Breaking down the product into features

2.2 Prioritizing the features based upon:

Features that are new and have higher susceptibility to defects

Features whose failures can have adverse business impact

Features that are difficult to test

Features that are extension of earlier features that have been defect prone

2.3 Deciding features to be tested

2.4 Deciding features not to be tested

3. TEST STRATEGY (Tells "How to Test" or defines the approach to test.) :

- what type of testing types need to be carried out, levels of testing techniques, method and tools to be used
- This includes what type of testing to be used :

3.1 System Test

3.2 Performance Test

3.3 Security Test

3.4 Functionality Test

3.5 Stress and Load Test

3.6 Recovery Test

3.7 Documentation Test

3.8 Beta Test

3.9 User Acceptance Test

4. DEPENDENCIES

- Dependencies may be categorized as external and internal based on which WBS can be formed. External dependencies are beyond the control of the Test leader. Some common external dependencies are:
- Availability of product from developers
- Hiring
- Training
- Acquisition of hardware/software required for training
- Internal dependencies are fully within the control of the tester. For example; Completing the test specification, Designing the test cases, Executing the tests

5. RISKS

5.1 Schedule Risks

Testing scheduling completely depends on the development schedule. Thus, it becomes difficult for the testing team to line-up resources i.e. people and tools in right manner at right time. This testing scheduling risk is even more severe in cases where a testing team is shared across multiple projects.

5.2 Technical

- Fixing some of the defects could lead to changes in the architecture and design. Carrying out such changes into the cycle may be expensive or even impossible. Once the developers fix the defects, the testing team would have even lesser time to complete the testing and is under even greater pressure i.e. **causing insufficient time for testing.**
- Certain defects are show stoppers may prevent the testing team to proceed further with testing, until development fixes such **show stopper defects.**

5.3 Management Risks

- Management has to take lot of efforts in hiring skilled and motivated people and they must also be available at required time.
- Management may fail in arranging the test automation tools, also manual testing might be error prone and labour intensive. Also, the test automation tools are expensive. And, the organization may face the risk of not being able to afford a test automation tool. Such risks lead to less effective and efficient testing.

5.4 Requirements Risks

When the requirements specified are not clearly documented, there is ambiguity in how to understand the results of a test. This could result in wrong defects being reported or in the real defects being missed out. This in turn, results in unnecessary and wasted cycles of communication between the development and testing teams and ultimately the loss of time.

Risk management involves

- Identifying the possible risks through use of checklists, organizational history and metrics, and informal network across the industry.
- Quantifying the risks upon probability of the risk and the impact of the risk. For example, show stopper defects have low probability but high impact.

- Planning how to mitigate the risks: It deals with identifying alternative strategies to reduce the risk events.
- Responding to risks when they become a reality.

6. ESTIMATIONS

6.1 Size Estimates quantifies the actual amount of testing that needs to be done.

Size of the product can be measured as follows :

- Lines Of Code (LOC) depends on the language, style of programming, compactness of programming, and so on.
- A function point is the application functions classified as inputs, outputs, interfaces, external data files and enquiries.
- Number of screens, reports or transactions - Each of these can be classified as simple, medium or complex.

Size of the testing can be measured as follows:

- Number of test cases
- Number of test scenarios

6.2 Effort Estimates

Factors that derive effort estimates:

- **Productivity Data :** speed at which various activities of testing can be carried out and this estimation is done based upon the historical data available in the organization.
- **Reuse opportunities :** Re-using concept ultimately reduces the efforts needed, Therefore, if the test architecture has been designed by concerning the reuse idea, then the effort required to do the same task once again obviously comes down.
- **Size estimate** also derives the estimation of efforts required. Say, for example, if some of the test cases or some part of the test cases can be reused from existing test cases, then the efforts involved in developing these would be near to zero. On the other hand, if a given test case is to be developed fully from scratch, it is obvious that the efforts must be put again which would increase the size of the test case i.e. divided by productivity.
- **Effort estimate** is measured in persons required per days, person required in number of months or person required in number of years. The effort estimate is then used to achieve the schedule estimation.

6.3 Schedule Estimates

It includes :

- Determining the external and internal dependencies among the activities
- Arranging the activities in a sequence based on the expected duration as well as on the dependencies
- Determining the time required for each of the WBS activities taking into account the above two factors.
- Monitoring and controlling the progress in terms of time and effort.
- Modifying and balancing the schedules and resources as necessary.
- Determine the time required to do perform each testing task. Also, specify the schedule for each testing task and test milestone. For each testing resource such as facilities, tools, and staff, specify its periods where it is necessary to use.

7. **TEST DELIVERABLES**

These are the documents that come out of the test cycle activity. The deliverables include :

- Test plan itself (Master test plan and various other test plans of the project)
- Test case design specifications
- Test logs produced by running the tests
- Test case coverage reports
- Test case results
- Test status reports
- Test summary reports

8. **RESPONSIBILITIES (Tells "Who will do the test")**

It includes identifying responsibilities, staffing and training them. The different role definitions should :

- Ensure there is clear accountability for a given task, so that each person knows what he or she has to do.
- Clearly list the responsibilities for various functions to various people, so that everyone knows how his or her work fits into the entire project
- Complement each other, ensuring no one steps on an others' toes
- Supplement each other so that no task is left unassigned.
- Staffing is done based on estimation of effort involved and the availability of time for release.

9. **HARDWARE AND SOFTWARE REQUIREMENTS**

Some of the hardware and software factors need to be considered are:

- Machine configuration (RAM, processor, disk and so on) needed to run the product under test.
- Compilers, test data generators, configuration management tools.
- Special requirements for running machine-intensive tests such as load or stress tests.

10. **ENVIRONMENT REQUIREMENTS**

- Client side operating system
- Client side h/w and s/w req.

11. **DEFECT RECORDING**

Document the procedures to log a defect.

12. **APPROVALS**

- Names and titles of all persons who should approve this plan. Provide space for the signatures and dates.
- Name (In Capital Letters) Signature Date Of Approval

1.

2.

3.

4.

11.11.2 Test Plan Examples

Example 1 : Write a test plan for a web based MSEB application system which provides the facilities like bill payment, customer grievance cell in support of complaints and feed back. Your plan should include the test documents and test strategies. (Make suitable assumptions if required)

Solution :

MSED provides electricity and keeps track of the usage of electricity in the whole Maharashtra. But their work is spread ona very wide area and the number of users are also in crores. Therefore, MSEB wants to automate the following functionalities. The functionality should be made available on-line (i.e. web-based)

Functionalities

(1) Maintaining the bill payment of customers
(2) Maintaining the user records.
(3) Maintaining customer grievance
(4) MSEB is using third party payments gateways of different partners like ICICI, HDFC etc.

1. **Test plan Objectives**

Test plan objectives for new web application

(i) Define the activities required to develop and test the system.
(ii) Communicate the test strategy to all responsible parties the System.
(iii) Define deliverables.
(iv) Communicate the various Dependencies and Risks to all stakeholders.

2. **Scope**

(a) **Data entry :** The new functionality of MSEB should allow the system administrator of the proposed system to enter the items of different Departments (billing dept., feedback dept., grievance dept., customer records dept. and so on), management should be able to view details of bills and address information of customers. The system should be user friendly and will provide error message to help direct the administrator through various options.

(b) **Security :** Each Customer will need a User id and password to login to the system. The system will allow the Customer to change his/her password whenever he/she wants to. Similarly as customers are paying money through third party, tight security should be provided to that interface.

3. **Test Strategy**

The test strategy consists of different types of tests that will fully exercise the online MSEB system.

(i) **System test :** The system test will focus on the behavior of the MSEB system. The system tests will test the integrated system and verifies that it meets all the customer requirements defined in the SRS.

(ii) **Performance Test :** Performance test will be conducted to ensure that the MSEB system's response time meets the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress and load.

(iii) **Security Test :** The tests will verify that unauthorized user access to confidential data is prevented.

(iv) **Configuration and Compatibility Test :** This is very essential tests required for web application. It determines how system performs in a particular hardware, software, operating system, network etc. environment.

(v) **Usability Test :** This test will determine how the system is user friendly. In MSEB system Customer and the employees of the MSEB offices are the ultimate users, so they should be able to handle all functionality of the system very easily.

(vi) **Automated Test :** A suite of automated tests will be developed to test the basic functionality of the online MSEB system and perform regression testing on areas of the system that previously had critical/major defects.

(vii) **Beta Test :** The MSEB staff will perform beta tests on the new online system and will report any defects they find to the test team and the corresponding development team. This will subject to tests that could not be performed in developer's test environment.

(viii) **User Acceptance Test :** Once the online system is ready for implementation, the MSEB authority will perform User Acceptance Testing. The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

4. Environment Requirements

(a) Workstation

1. Pentium Processor
2. 256 RAM
3. 40 GB Hard Disk
4. Windows XP/NT/2000/Linux/Mac
5. LAN Network
6. Any current trend browsers

(b) Server and Client Side Software

1. Web Server - Apache
2. Programming language – Java, JSP, Servlets.
3. Database - Oracle

5. Functions to be tested

The following is a list of functions that will be tested

(i) Add/update Customer information
(ii) Add/update bill pyment information
(iii) Add/update customer grievance and feedback
(iv) Add/update details from different departments
(v) Search for customers or areas or bill balances
(vi) Security features
(vii) Error messages
(viii) Payment through third party
(ix) Reports of MSEB

6. Resources

The test team will consists of :

(i) A project Manager
(ii) A test lead
(iii) Test team
(iv) The MSEB Manager
(v) MSEB employees
(vi) MSEB Customers

Example 2 : Peoples-Bazaar is a famous shopping mall spread globally. Shopping mall wants to automate the following functionality.

1) Maintaining user details, giving proper authentication
2) Maintaining items under different Departments, like music, clothing, etc.
3) Maintaining online customer order.
4) Providing on-line payment facility towards order.

'Peoples-Bazaar' is using third party payment gateways of different partners like ICICI, HDFC etc.

Design test plan, define scope, testing strategies, coverage, environment and other design details.

Solution :

1. Test plan Objectives

Test plan objectives for new web application

(i) Define the activities required to develop and test the system.
(ii) Communicate test strategies to all responsible parties.
(iii) Define deliverables.
(iv) Communicate various Dependencies and Risks to all responsible parties.

2. Scope

(a) **Data entry :** The new functionality of shopping mall should allow Administrator of this proposed system to enter the items of different Departments, view orders of customers. The system should be user friendly and will provide error message to help direct the administrator through various options.

(b) **Security :** Each Customer will need a User id and password to login to the system. The system will allow the Customer to change his/her password whenever he/she wants to. Similarly as customers are paying money through third party, tight security should be provided to that interface.

3. Test Strategy

The test strategy consists of different types of tests that will fully exercise the online shopping mall.

(i) **System test :** The system test will focus on the behavior of the MSEB system. The system tests will test the integrated system and verifies that it meets all the customer requirements defined in the SRS.

(ii) **Performance Test :** Performance test will be conducted to ensure that the shopping mall system's response time meets the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress and load.

(iii) **Security Test :** The tests will verify that unauthorized user access to confidential data is prevented.

(iv) **Configuration and Compatibility Test :** This is very essential tests required for web application. It determines how system performs in a particular hardware, software, operating system, network etc. environment.

(v) **Usability Test :** This test will determine how the system is user friendly. In online shopping mall system Customer is a user, so he/she should be able to handle all functionality of the system very easily.

(vi) **Automated Test :** A suite of automated tests will be developed to test the basic functionality of the online shopping mall system and perform regression testing where necessary.

(vii) **Beta Test :** The shopping mall staff will beta tests the new online system and will report any defects they find. This will subject to tests that could not be performed in our test environment.

(viii) **User Acceptance Test :** Once the online system is ready for implementation, the shopping mall authority will perform User Acceptance Testing. The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

4. **Environment Requirements**

(a) Workstation
1. Pentium Processor
2. 256 RAM
3. 40 GB Hard Disk
4. Windows XP/NT/2000/Linux/Mac
5. LAN Network
6. Any current trend browsers

(b) Server and Client Side Software
1. Web Server – Internet Information Server
2. Programming language – ASP
3. Database - Oracle

5. **Functions to be tested**

The following is a list of functions that will be tested

(i) Add/update Customer information
(ii) Add/update Items from different departments
(iii) Search/lookup Item information
(iv) Security features
(v) Error messages
(vi) Placing order
(vii) Payment through third party
(viii) Reports for shopping mall

6. **Resources**

The test team will consists of:

(i) A project Manager
(ii) A test lead
(iii) Test team
(iv) The shopping Mall Manager
(v) Shopping mall employees

Example 3 : A company has designed its computerized payroll system to handle the following activites.

1. Provide the user with payslip containing basic all allowances, all deductions
2. Store the employee information
3. Security employee information
4. Prints various reports
5. Provide compatibility with mainframe and PCS and LAN.
6. Write a test plan.

Solution :

SAN's has recruited number of employees and the company was wide spread. And it was impossible to maintain the details of all these employees manually. And, also if the manager sitting in any other department needed the details of employees of other department, he had to get it manual documents, or the softcopy in the pendrive or by mail. Therefore, SAN company wanted to automate the following functionalities. The functionality should be made available on-line (i.e. web-based)

Functionalities

1. Maintaining the employee details
2. Secure the employee records.
3. Maintain the pay slips and payments of all the employees.
4. Produce the payments by cuttings and increments or bonus.
5. Provide print reposts of employee details, their payments and information about their leaves and vacation.

1. **Test plan objectives**

Test plan objectives for new web application

(i) Define the activities required to develop and test the system.
(ii) Communicate to all responsible parties the System Test Strategy.
(iii) Define deliverables and responsible parties.
(iv) Communicate to all responsible parties the various Dependencies and Risks.

2. **Scope**

(a) **Data entry :** The new functionality of SAN company should allow the system administrator of the proposed system to enter the items of different Departments (employee personal details, payment details, leave record details, over time details and so on), management should be able to view details of payments, contact info., leave and over time information of employees. The system should be user friendly and will provide error message to help direct the administrator through various options.

(b) **Security :** Each employee will need a User id and password to login to the system. The system will allow the Customer to change his/her password whenever he/she wants to.

3. **Test strategy**

The test strategy consists of different types of tests that will fully exercise the SAN company.

(i) **System test :** The system test will focus on the behavior of the MSEB system. The system tests will test the integrated system and verifies that it meets all the customer requirements defined in the SRS.

(ii) **Performance Test :** Performance test will be conducted to ensure that the SAN system's response time meets the user expectations and does not exceed the specified performance criteria. During these tests, response times will be measured under heavy stress and load.

(iii) **Security Test :** Security tests will determine how to secure the new online SAN system is. The tests will verify that unauthorized user access to confidential data is prevented.

(iv) **Configuration and Compatibility Test :** This is very essential tests required for web application. It determines how system performs in a particular hardware, software, operating system, network etc. environment.

(v) **Usability Test :** This test will determine how the system is user friendly. In SAN system Top Management and the employees of the SAN office are the ultimate users, so they should be able to handle all functionality of the system very easily.

(vi) **Automated Test:** A suite of automated tests will be developed to test the basic functionality of the online SAN system and perform regression testing on areas of the system that previously had critical/major defects.

(vii) **Beta Test :** The SAN staff will perform beta tests on the new online system and will report any defects they find to the test team and the corresponding development team. This will subject to tests that could not be performed in developer's test environment.

(viii) **Acceptance Test :** The purpose of these tests is to confirm that the system is developed according to the specified user requirements and is ready for operational use.

4. **Environment requirements**

(a) Workstation

1. Pentium Processor
2. 256 RAM
3. 40 GB Hard Disk
4. Windows XP/NT/2000/Linux/Mac
5. LAN Network
6. Any current trend browsers

(b) Server and Client Side Software

1. Web Server – Apache
2. Programming language – Java, JSP, Servlets.
3. Database - Oracle

5. **Functions to be tested**

The following is a list of functions that will be tested

(i) Add/update the employee details

(ii) Secure the employee records.

(iii) Add/update the payment details of employees.

(iv) Add/update the employee cuttings and increments or bonus.

(v) Add/update the employee leave, vacation and over time details

(vi) Report generation of employee details, their payments and information about their leaves and vacation.

6. **Resources**

The test team will consists of :

(i) A project Manager

(ii) A test lead

(iii) Test team

(iv) The SAN top management

(v) SAN employees

(vi) SAN Customers

Syllabus Topic : Test-Case Design

11.12 Test Cases

Test case is defined as

- A set of test inputs, execution pre-conditions and expected outputs developed for a particular objective.
- The inputs will be tried to match the actual output with the expected output to verify the correctness of the software.

Contents of a test case

- Test Item : specifies the name of a particular component of the product that is to be tested
- Test case id : it is a unique identifier number given to the test cases.
- Test case - name and description : This clearly states what needs to be tested
- Test pre-condition : it indicates the stae of the system in which it is required to be before executing the test case.
- Input : these are the sequences to be entered by the user – it may be a numeric value, alphabets, symbols, just a key stroke or anything that is listed down there.
- Expected output : It states what should happen when the test case is executed
- Actual output : It indicates whether the test case is behaving in the same manner as expected output or not. It give space to the tester to record unexpected results.

Test Case Designing

1. Test Item
2. Preconditions
3. Test case

testcase id	test case name	test case desc	Test steps			test case status	test status (P/F)	test priority
			Step / Input	Expected Output	Actual Output			

11.12.1 Login Test Case

- test URL : www.yahoo.co.in/login
- Preconditions : Open Web browser and enter the given url in the address bar. Home page must be displayed. All test cases must be executed from this page.

Test case id	Test case name	Test case desc	Test steps			Test case status	Test status (P/F)	Test Priority
			Step / Input	Expected Output	Actual Output			
Login01	Validate Login	To verify that Login name on login page must be greater than 3 characters	Enter login name less than 3 chars (say a) and password and click Submit button	An error message "Login not less than 3 characters" must be displayed		Design		High
			Enter login name less than 3 chars (say ab) and password and click Submit button	An error message "Login not less than 3 characters" must be displayed		Design		High
			Enter login name 3 chars (say abc) and password and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High
Login02	Validate Login	To verify that Login name on login page should not be greater than 10 characters	Enter login name greater than 10 chars (say abcdefghijk) and password and click Submit button	An error message "Login not greater than 10 characters" must be displayed		Design		High
			Enter login name less than 10 chars (say abcdef) and password and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayeld		Design		High

Test case id	Test case name	Test case desc	Test steps			Test case status	Test status (P/F)	Test Priority
			Step / Input	Expected Output	Actual Output			
Login03	Validate Login	To verify that Login name on login page does not take special characters	Enter login name starting with specail chars (!hello) password and click Submit button	An error message "Special chars not allowed in login" must be displayed		Design		High
			Enter login name ending with specail chars (hello$) password and click Submit button	An error message "Special chars not allowed in login" must be displayed		Design		High
			Enter login name with special chars in middle(he&^llo) password and click Submit button	An error message "Special chars not allowed in login" must be displayed		Design		High
Pwd01	Validate Password	To verify that Password on login page must be greater than 6 characters	enter Password less than 6 chars (say a) and Login Name and click Submit button	An error message "Password not less than 6 characters" must be displayed		Design		High
			Enter Password 6 chars (say abcdef) and Login Name and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High
Pwd02	Validate Password	To verify that Password on login page must be less than 10 characters	Enter Password greater than 10 chars (say a) and Login Name and click Submit button	An error message "Password not greater than 10 characters" must be displayed		Design		High
			Enter Password less than 10 chars (say abcdefghi) and Login Name and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High
Pwd03	Validate Password	To verify that Password on login page must be allow special characters	Enter Password with special characters(say !@hi&*P) Login Name and click Submit button	Login success full or an error message "Invalid Login or Password" must be displayed		Design		High

Test case id	Test case name	Test case desc	Test steps			Test case status	Test status (P/F)	Test Priority
			Step / Input	Expected Output	Actual Output			
Link01	Verify Hyperlinks	To Verify the Hyper Links available at left side on login page working or not	Click Home Link	Home Page must be displayed		Design		Low
			Click Sign Up Link	Sign Up page must be displayed		Design		Low
			Click New Users Link	New Users Registration Form must be displayed		Design		Low
			Click Advertise Link	Page with Information and Tariff Plan for Advertisers must be displayed		Design		Low
			Click Contact Us Link	Contact Information page must be displayed		Design		Low
			Click Terms Link	Terms Of the service page must be displayed		Design		Low
Flink01	Verify Hyper links	To Verify the Hyper Links displayed at Footer on login page working or not	Click Home Link	Home Page must be displayed		Design		Low
			Click Sign Up Link	Contact Information page must be displayed		Design		Low
			Click Contact Us Link	Page with Information and Tariff Plan for Advertisers must be displayed		Design		Low
			Click Advertise Link	Terms Of the service page must be displayed		Design		Low
			Click Terms Of Membership Link	Privacy Policy page must be displayed		Design		Low
			Click Privacy Policy Link	Privacy Policy page must be displayed		Design		Low
Lblink01	Verify Hyper links	To Verify the Hyper Links displayed at Login Box on login page working or not	Click NEW USERS Link located in login box	New Users Registration Form must be displayed		Design		Low
			Click New Users(Blue Color) Link located in login box	New Users Registration Form must be displayed		Design		Low
			Click Forget Password Link located in login box	Password Retrieval Page must be displayed		Design		Medium

11.12.2 Email Address Test Case

Guidelines are given below; Try to design the test case template properly.

- REQUIREMENTS : consider a text box It should accept only 10 characters - 20 characters that may include @, _ , . symbols and small alphabets. So, here we should use boundary value analysis and equalence patition technique to write this test case.
- TEST CASE :There are many validation for E-mail here we are defining some of them.
- Email format: vishal.bilgaiyan @ gmail.com

(1) (2) (3) (4)

- We have divided Email ID in 4 parts.
- In first part we have to check invalid condition input spaces, Left blank, use special character, symbols etc.
- In Second part we have to check invalid condition input other specail character [!#$%^&*()_-:;"'?/|\) at the place of @. left this field as blank. input spaces.etc.
- In third part we have to check invalid domain.
- In fourth part we have to check only valid suffix(.com,.co.in,.edu,.info,.org,.in etc)

Valid Email address	Reason
email@domain.com	Valid email
firstname.lastname@domain.com	Email contains dot in the address field
email@subdomain.domain.com	Email contains dot with subdomain
firstname+lastname@domain.com	Plus sign is considered valid character
email@123.123.123.123	Domain is valid IP address
email@[123.123.123.123]	Square bracket around IP address is considered valid
"email"@domain.com	Quotes around email is considered valid
1234567890@domain.com	Digits in address are valid
email@domain-one.com	Dash in domain name is valid
a_b@domain.com	Underscore in the address field is valid
email@domain.name	.name is valid Top Level Domain name
email@domain.co.jp	Dot in Top Level Domain name also considered valid (use co.jp as example here)
firstname-lastname@domain.com	Dash in address field is valid

Invalid Email address	Reason
plainaddress	Missing @ sign and domain
#@%^%#$@#$@#.com	Garbage
@domain.com	Missing username
Joe Smith <email@domain.com>	Encoded html within email is invalid
email.domain.com	Missing @

Invalid Email address	Reason
email@domain@domain.com	Two @ sign
.email@domain.com	Leading dot in address is not allowed
email.@domain.com	Trailing dot in address is not allowed
email..email@domain.com	Multiple dots
あいうえお@domain.com	Unicode char as address
email@domain.com (Joe Smith)	Text followed email is not allowed
email@domain	Missing top level domain (.com/.net/.org/etc)
email@-domain.com	Leading dash in front of domain is invalid
email@domain.web	.web is not a valid top level domain
email@111.222.333.44444	Invalid IP format
email@domain..com	Multiple dot in the domain portion is invalid

11.12.3 Test Case for Calculator

Guidelines are given below; Try to extend the test case template properly.

Test Case Id	Test Case Name
01	Verify all the basic functionality +,-,*,/ .
02	Verify complex functionality like sqrt for both +ve and -ve Numbers
03	Verify by dividing by Zero
04	Verify the expressions like 2+-4, -4+-4, -4-4
05	Verify = button
06	Verify whether pressing AC button clears the screen
07	Verify by multiplying by Zero
08	Verify that multiplication of two negative numbers is positive. (-2*-3 = 6)

11.12.4 Test Case for ATM

Guidelines to test few of the functions that are performed while withdrawing an amount from an ATM are given below; Try to extend the test case template properly.

Test Case Id	Test Case Name	Input	Expected Output
01	Verify that system reads a customer's ATM card	Insert a readable card	Card is accepted System asks for entry of PIN
02	Verify that system rejects an unreadable card	Insert an unreadable card	Card is ejected and system displays an error screen and is ready to start a new session
03	Verify that system accepts customer's PIN	Enter PIN	System displays a menu of transaction types

Test Case Id	Test Case Name	Input	Expected Output
04	Verify that system allows customer to perform a transaction	Perform a transaction	System asks which type of transaction to be performed.
05	System allows multiple transactions in one session	Answer YES	System displays a menu of transaction types
06	Session ends when customer chooses not to do another transaction	Answer NO	System ejects the ATM card and is ready to start a new session
07	System properly handles an invalid PIN	Enter an incorrect PIN and then try a transaction	The Invalid PIN Extension is performed
08	System asks customer to choose type of account to withdraw from	Choose savings account	System displays a screen where you can enter the withdrawal amount.
09	System verifies that customer has sufficient balance to fulfill his request of withdrawal amount	Enter the amount greater than the account balance	System displays an appropriate message and gives the customer an option of choosing to do another transaction or not.
10	Three incorrect re-entries of PIN result in retaining the card, aborting the transaction and locking the account for 48 hours	Enter incorrect PIN for three times	An appropriate message is displayed, card is retained by machine, session is terminated and the account gets automatically locked for 48 hours.

Review Questions

Q. 1. Define the terms :

1. Walk-through
2. Inspection
3. Desk-checking
4. Code Review
5. Code complexity testing (Cyclomatic complexity)
6. BVA (Boundary Value Analysis)
7. ECP (Equivalence Class Partition)
8. Bug life cycle

Q. 2 Differentiate:

1. Static and dynamic testing
2. Integration and System Testing

□□□