**Manual Testing:**

* Manual Testing is a testing process in which test cases are **executed manually** without using any automation tool.
* Test cases are generated and planned and implemented manually and it is done according to the perspective of end users.
* Manual Testing is **mandatory** for every newly deployed software **before Automation Testing**. It requires the great efforts and time but it gives surety of 100% bug-free software.
* Manual Testing requires good knowledge of manual testing techniques but not of any automated testing tool.
* Manual Testing is essential because one of the basic fundamentals **of Software testing is 100% Automation is not Possible.**

**Why Manual Testing:**

* To give **stable, bug-free and good quality** product to the client.
* If the test engineer does manual testing, he/she can test the application as an **end-user perspective** and get more familiar with the product, which helps them to write the correct test cases of the application and give the quick feedback of the application.

**Types of Manual Testing:**

* White Box Testing
* Black Box Testing
* Gray Box Testing

**White Box Testing:**

* The white box testing **done by developer**. Developer will test each and every line of code and give it to the tester.
* Since the **code is visible to the developer** so it is called White box testing.

**Block Box Testing:**

* It is done by tester. The tester will check the functionalities of the software according to the client’s needs.
* Since the code is not visible to the tester so it is called black box testing.

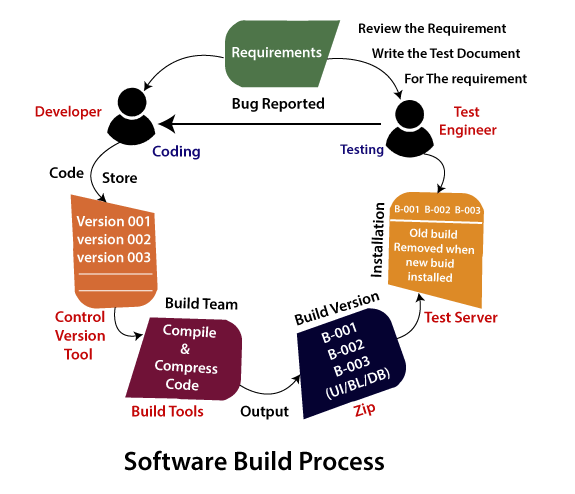
**Gray Box Testing:**

* The person who has good knowledge of **testing as well as coding** will do the code testing and functionality Testing.
* Since it is a combination of white box and black box testing so it is called Gray Box Testing

**How to Perform Manual Testing:**

* First Tester **observe** given **software related documents** to select testing areas.
* Tester analyses the requirement document to **cover all requirements** stated by **the customer**
* Tester develops **testing cases** according to requirement document
* All test cases are **executed manually** by using white box testing and block box testing.
* If bugs occurred the test team will inform to the development team
* The development team will rectify the problem and return to the testing team for retest.

**Software Build Process:**



1. Whenever the requirement is collected, it will provide to the two different team called Developer and Testing Team.
2. Then developer team will start writing the code.
3. Meantime the tester will go through the requirements and prepares the necessary documents. Up to now the developer may complete the code and store into the **control vison tool**.
4. After that, the code changes in the **UI**, and these changes handle by one separate team, which is known as the **build team**.
5. Then they start compile and compress the code with help of build tool. The output which we get will stored as zip file is called as **Build (App or Software)**
6. Then Build will be installed into the **test server**. After that the testing engineer will start do testing by accessing the test server through **Test URL**
7. If the Testing Engineer found any bug means they will inform to the concern developer.
8. Then, the developer will modify the bug by accessing the test URL. Once they fix the bug, they will again upload the new file and they will delete the old one.
9. The process will be going until the **build is getting stabled**.
10. Once the Build is stable, they deliver the Build or App to the Customer.

**Advantages of Manual Testing**

* It does **not require programming knowledge** while using the Black box method.
* It is used to test dynamically changing GUI designs.
* Tester interacts with software as a real user so that they are able to discover usability and user interface issues.
* It ensures that the software is a hundred percent bug-free.
* It is cost-effective.
* Easy to learn for new testers.

**Disadvantages of Manual Testing**

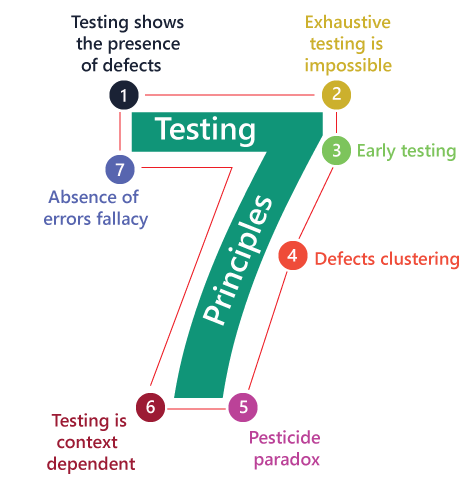
* It requires a large number of human resources.
* It is very time-consuming.
* Tester develops test cases based on their skills and experience. There is no evidence that they have covered all functions or not.
* Test cases cannot be used again. Need to develop separate test cases for each new software.

It does not provide testing on all aspects of testing.

* Since two teams work together, sometimes it is difficult to understand each other's motives, it can mislead the process.

**Software Testing Principles:**

* Testing shows the presence of defects
* Exhaustive Testing is not possible
* Early Testing
* Defect Clustering
* Pesticide Paradox
* Testing is context-dependent
* Absence of errors fallacy



**Testing Shows the presence of defects:**

Testing Engineer test the application for delivering the application bug or defect free. So, primary purpose of testing is to **address the bugs**.

And, by doing testing we can decrease the errors but it doesn’t mean that the whole application is free because we can address the error even during the deployment.

**Exhaustive Testing is not possible:**

Sometimes it seems to be very hard to test all the modules and their features with effective and non- effective combinations of the inputs data throughout the actual testing process.

Most of the time it takes more time and eventually our hard work could become unsuccessful. So better complete the testing, by testing modules according to its importance.

**Early Testing:**

The Early testing means finding the bugs in earlier stage. To achieve this, we **need requirement document** at earlier stage. So that we can find the error at earlier stage it would cause less amount of cost as compared to finding errors at final stage.

**Defect Clustering:**

Defect Clustering is the process of addressing the uncertain modules which has 80% of the bug in a whole project. Which could be because of its complication.

**Pesticide Paradox:**

* This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application.
* To get over these pesticide paradoxes, it is very significant to review all the test cases frequently.
* And the new and different tests are necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

**Testing is Context-Dependent:**

The testing is context – dependent states that we have a various field in the market and each has its own functionalities and features so, for testing these, kind of applications or software we have to get aid from various technology, method and different approaches to test the application. Therefore, testing depends on the context of the application

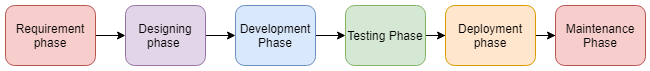
**Absence of Error Fallacy:**

The absence of error fallacy means **identifying and fixing** the bugs **would not help** if the application is **impractical (not practical)** and not able to accomplish the client's requirements and needs.

**Software Development Life Cycle (SDLC):**

SDLC is a process that creates a **structure of development** of software. There are different phases within SDLC, and each phase has its various activities. It makes the development team able to **design, create, and delive**r a high-quality product.

It simply gives the order of execution of the phases.



**Requirement Phase:**

* This is a crucial phase
* The phase in which we are collecting the requirements from the client.
* We have to get the information about the product like how will be it used and who will use to determine the load operations.

**Designing Phase:**

* This is high priority case.
* The detailed analysis of new software according to requirement phase will be happen here.
* Here logical designing converted into physical design.
* The decision like which programming language, data bases, and the combination of software and hardware to provide the platform to run the software that we have to use those, kind of decision is taken in this phase only.
* There are several techniques and tools, such as data flow diagrams, flowcharts, decision tables, and decision trees, Data dictionary, and the structured dictionary are used for describing the system design

**Development/Build Phase:**

* Once the design completed then it will be given to the development team where the work will divide into small units.
* Frontend -developer will do all the GUI work.
* Back end -developer will do all the backend operations according to the requirements.
* Since this is the coding phase, it takes the longest time and more focused approach for the developer in the software development life cycle.

**Testing Phase:**

* Testing determines whether the software is actually giving the result as per the requirements addressed in the requirement phase or not.
* Develop Team makes the test plan. This plan includes all type of essential testing such as integration testing, system testing, unit testing, acceptance testing. Non -functional testing also done in this phase.

**Deployment/Deliver Phase:**

* Whenever the testing phase is completed and it gives satisfying result, then it becomes ready to deliver to the customer.
* Once we delivered to the customer, we recommend them to do beta testing. In beta testing customer can require any changes which are not present in the software but mentioned in the document or changes in GUI to make it user friendly.
* Besides if any problem occurs it will be informed to the development team and they will fix it instantly if it is severe or otherwise they will simply said wait for the next version.

**Maintenance:**

* The maintenance phase is the last and long-lasting phase of SDLC because it is the process which **continues** until the **software's life cycle** comes to an end.
* When a customer starts using software, then actual problems start to occur, and at that time there's a need to solve these problems.
* This phase also includes making **changes in hardware and software** to **maintain** its operational **effectiveness** like to improve its **performance**, enhance security features and according to customer's requirements with upcoming time.
* This process to take care of product time to time is called maintenance

**Software Testing Life Cycle (STLC):**

The procedure of software testing is also known as STLC (Software Testing Life Cycle) which includes phases of the testing process. The testing process is executed in a well-planned and systematic manner. All activities are done to improve the quality of the software product.

**Steps**:

1. [Requirement Analysis](https://www.javatpoint.com/software-testing-life-cycle#requirement-analysis) – detail analysis of requirement document
2. [Test Plan Creation](https://www.javatpoint.com/software-testing-life-cycle#test-plan-creation) – developing test cases to execute
3. [Environment setup](https://www.javatpoint.com/software-testing-life-cycle#environment-setup) – set the testing environment. It is done by senior developers
4. [Test case Execution](https://www.javatpoint.com/software-testing-life-cycle#test-case-execution) – Then executes the test cases for each and every essential functionality.
5. [Defect Logging](https://www.javatpoint.com/software-testing-life-cycle#defect-logging) - Defect logging analysis mainly works to find out **defect distribution depending upon severity and types**. If, any defect is detected, then the software is returned to the development team to fix the defect, then the software is re-tested on all aspects of the testing
6. [Test Cycle Closure](https://www.javatpoint.com/software-testing-life-cycle#test-cycle-closure) -

* The test cycle closure report includes all the documentation related to software design, development, testing results, and defect reports.
* This phase evaluates the strategy of development, testing procedure, possible defects in order to use these practices in the future if there is a software with the same specification

**SDLC MODELS:**

* Waterfall model
* Spiral model
* Verification and validation model
* Prototype model
* Hybrid model

**Water Model:**

It is a simple model that is easy to use as well as understand. The execution happens in the **sequence order**, which means that the outcome of the one-stage is equal to the input of another stage. That's why it is also known as the **Linear-sequential life cycle model**.

The waterfall model is divided into various stages, which are as follows:

1. Requirement collection
2. Feasibility study
3. Design
4. Coding
5. Testing
6. Installation
7. Maintenance

**Spiral Models:**

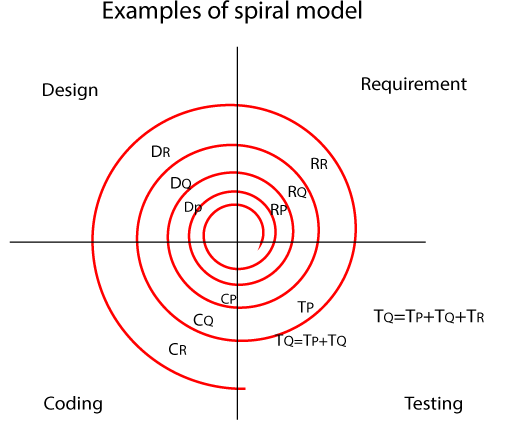
The biggest problem we face in the **waterfall model is that taking a long duration** to complete the product, and the software became outdated. To solve this problem, we have a new approach, which is known as the Spiral model. The spiral model is also known as the cyclic model.

we create the application **module by module** and handed over to the **customer** so that they can start using the application at a very **early stage**

In this model, we develop the application in the stages because sometimes the client gives the requirements in between the process.

The different phases of the spiral model are as follows:

1. Requirement analysis
2. Design
3. Coding
4. Testing and risk analysis

****

**Consider that the project that need 3 modules named P, Q, R to develop a software**

**Module P, Module Q, Module R**

**Requirement analysis = RP, RQ, RR**

**Design = DP, DQ, DR**

**Coding = CP, CQ, CR**

**Testing = TP, TQ, TR**

**Cycle 1:**

**Module 1 = RP 🡪 DP🡪TP**

**Cycle 2:**

**Once the everything is done for Module 1, P, then it will go Module 2, Q**

**Module 1 & Module 2 = RQ🡪 DQ🡪 T = TP+TQ**

**Cycle 3:**

**After the two Cycle, completed, then 3rd cycle executes,**

**Module1&Module2& Module3 = RR🡪DR🡪T = TP+TQ+TR**

**Protype Model:**

The most significant disadvantage of previous models (waterfall and spiral) is that there were lots of **customer rejection** that happens after the application was **developed**, and there was no involvement of the customers in between the project.

The prototype is just the sample or a dummy of the required software product. We should get the **product reviewed by client** then only we started develop the product.

**Protype Model Process:**

1. Requirement analysis
2. feasibility study
3. Create a prototype
4. Prototype testing
5. Customer review and approval
6. Design
7. Coding
8. Testing
9. Installation and maintenance

**V-Model (V&V-Model/Validation &Verification Model):**

* V-Model came up to overcome the drawback of the water model.
* In this model activities goes on downward direction and one point in a time it started going upward direction for testing process thus make the shape of V so, it is called V -Model.

**When We go for this model:**

🡪 Larger Application – Which means app has **n- number** of modules

🡪 Complex Application – Which means app has many dependencies modules

**Requirements:**

It is the document which is collected from the client. For, V and V Model we have two files,

🡪 CRS/BRS – Customer/Business Requirement Specification (English language)

🡪 SRS/FS – Software/Functional Requirement Specification (Coding Language)

**Characteristics of Requirement:**

🡪 Requirement should be in detail (modules, components, functional specification)

🡪 It has to be in sequential order.

🡪 Has to be written in simple language

🡪 It should be measurable and countable

**V and V Model Process:**

1.Review Process – Verification Phase (Includes Coding)

2. Validation Process – Validation Phase (whole testing process)

Here the development and testing team works parallelly

**Step1:**

The test team will review about CRS about

Review:

🡪 Incorrect requirements

🡪 Missing requirements

🡪 Conflicts in the requirements in the CRS

Write Acceptance Test documents

Once the test engineer finished the CRS review, they will give to development team, they will fix the bugs and updates the CRS and concurrently developing SRS document.

**Stage 2:**

After completing the CRS review, the test team start reviews SRS given by the development team. Meanwhile, the development team start developing the HLD (high level design).

Review SRS against the CRS,

Scenarios,

🡪 Each CRS transferred to SRS

🡪 CRS not transformed properly to SRS

Simply we have to check whether all the given CRS converted to SRS or not.

Writing Test Acceptance Documents

**Stage 3:**

Review the HLD

Here, the test engineer start reviewing about HLD that is designed by developers. Meantime the developers started creating the LLD (low level design)

Writing the Test Acceptance Documents

**Stage 4:**

Once the review of HLD is done then, the development team started writing the code and develops the application. Meantime the testing engineer reviews the LLD.

1.Review the LLD

2.Writing test acceptance documents

**Stage 5:**

After the completion of the coding part, the developers will perform one round of unit testing, which is also called white box testing, and check every line of the code and make sure that the code is correct.

The testing team, will perform multiple testing such as functional testing, integration testing, and system testing, and acceptance testing.

**Hybrid Model**

The hybrid model is the combination of two or more primary (traditional) models and modifies them as per the business requirements. This model is dependent on the other SDLC models, such as spiral, V and V, and prototype models. The hybrid model is mainly used for small, medium, and large projects. It focuses on the risk management of the product

The most commonly used combination of two models is as follows:

1. **Spiral and prototype**

**V & V and Prototype**

**White Box Testing Techniques:**

# **Data Flow Testing**

Data flow testing is used to analyze the flow of data in the program. It is the process of collecting information about how the variables flow the data in the program. It tries to obtain particular information of each particular point in the process.

Data flow testing uses the control flow graph to detect illogical things that can interrupt the flow of data. Anomalies in the flow of data are detected at the time of associations between values and variables due to:

* If the variables are used without initialization.
* If the initialized variables are not used at least once.

**Let's understand this with an example:**

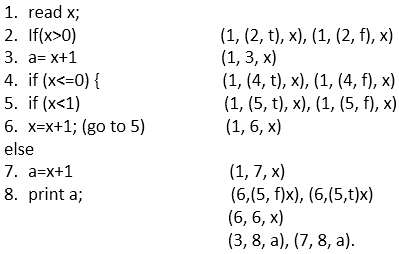
25.6M

605

OOPs Concepts in Java

**Next**

**Stay**



In this code, we have a total 8 statements, and we will choose a path which covers all the 8 statements. As it is evident in the code, we cannot cover all the statements in a single path because if statement 2 is true then statements 4, 5, 6, 7 not covered, and if statement 4 is true then statement 2 and 3 are not covered.

So, we are taking two paths to cover all the statements.

1. **x= 1**

**Path** - 1, 2, 3, 8

**Output = 2**

When we set value of x as 1 first it come on step 1 to read and assign the value of x (we took 1 in path) then come on statement 2 (x>0 (we took 2 in path)) which is true and it comes on statement 3 (a= x+1 (we took 3 in path)) at last it comes on statement 8 to print the value of x (output is 2).

For the second path, we take the value of x is 1

2. **Set x= -1**

**Path** = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8

**Output = 2**

When we set the value of x as ?1 then first, it comes on step 1 to read and assign the value of x (we took 1 in the path) then come on step number 2 which is false because x is not greater than 0 (x>0 and their x=-1). Due to false condition, it will not come on statement 3 and directly jump on statement 4 (we took 4 in path) and 4 is true (x<=0 and their x is less than 0) then come on statement 5 (x<1 (we took 5 in path)) which is also true so it will come on statement 6 (x=x+1 (we took 6 in path)) and here x is incremented by 1.

So,

x=-1+1

x=0

There is value of x become 0. Now it goes to statement 5(x<1 (we took 5 in path)) with value 0 and 0 is less than 1 so, it is true. Come on statement 6 (x=x+1 (we took 6 in path))

x=x+1

x= 0+1

x=1

There x has become 1 and again goes to statement 5 (x<1 (we took 5 in path)) and now 1 is not less than 1 so, condition is false and it will come to else part means statement 7 (a=x+1 where the value of x is 1) and assign the value to a (a=2). At last, it come on statement 8 and print the value (Output is 2).

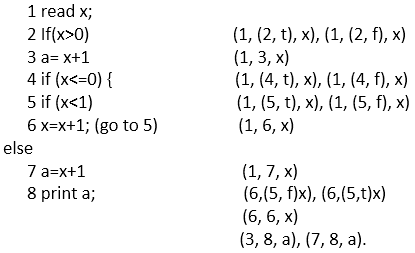
Make associations for the code:

### **Associations**

In associations we list down all the definitions with all of its uses.

(1, (2, f), x), (1, (2, t), x), (1, 3, x), (1, (4, t), x), (1, (4, f), x), (1, (5, t), x), (1, (5, f), x), (1, 6, x), (1, 7, x), (6,(5, f)x), (6,(5,t)x), (6, 6, x), (3, 8, a), (7, 8, a).

## How to make associations in data flow testing <link>

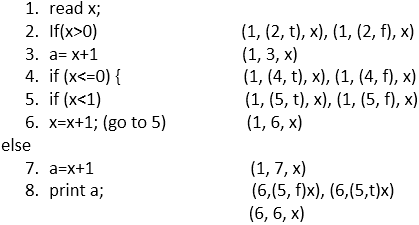


* **(1, (2, t), x), (1, (2, f), x)-** This association is made with statement 1 (read x;) and statement 2 (If(x>0)) where x is defined at line number 1, and it is used at line number 2 so, x is the variable.  
  Statement 2 is logical, and it can be true or false that's why the association is defined in two ways; one is (1, (2, t), x) for true and another is (1, (2, f), x) for false.
* **(1, 3, x)-** This association is made with statement 1 (read x;) and statement 3 (a= x+1) where x is defined in statement 1 and used in statement 3. It is a computation use.
* **(1, (4, t), x), (1, (4, f), x)-** This association is made with statement 1 (read x;) and statement 4 (If(x<=0)) where x is defined at line number 1 and it is used at line number 4 so x is the variable. Statement 4 is logical, and it can be true or false that's why the association is defined in two ways one is (1, (4, t), x) for true and another is (1, (4, f), x) for false.
* **(1, (5, t), x), (1, (5, f), x)-** This association is made with statement 1 (read x;) and statement 5 (if (x<1)) where x is defined at line number 1, and it is used at line number 5, so x is the variable.  
  Statement 5 is logical, and it can be true or false that's why the association is defined in two ways; one is (1, (5, t), x) for true and another is (1, (5, f), x) for false.
* **(1, 6, x)-** This association is made with statement 1 (read x;) and statement 6 (x=x+1). x is defined in statement 1 and used in statement 6. It is a computation use.
* **(1, 7, x)-** This association is made with statement 1 (read x) and statement 7 (a=x+1). x is defined in statement 1 and used in statement 7 when statement 5 is false. It is a computation use.
* **(6, (5, f) x), (6, (5, t) x)-** This association is made with statement 6 (x=x+1;) and statement 5 if (x<1) because x is defined in statement 6 and used in statement 5. Statement 5 is logical, and it can be true or false that's why the association is defined in two ways one is (6, (5, f) x) for true and another is (6, (5, t) x) for false. It is a predicted use.
* **(6, 6, x)-** This association is made with statement 6 which is using the value of variable x and then defining the new value of x.  
  x=x+1  
  x= (-1+1)  
  Statement 6 is using the value of variable x that is ?1 and then defining new value of x [x= (-1+1) = 0] that is 0.
* **(3, 8, a)-** This association is made with statement 3(a= x+1) and statement 8 where variable a is defined in statement 3 and used in statement 8.
* **(7, 8, a)-** This association is made with statement 7(a=x+1) and statement 8 where variable a is defined in statement 7 and used in statement 8.

### **Definition, c-use, p-use, c-use some p-use coverage, p-use some c-use coverage in data flow testing <link>**

The next task is to group all the associations in Definition, c-use, p-use, c-use some p-use coverage, p-use some c-use coverage categories.

**See the code below:**



So, these are the all association which contain definition, Predicate use (p-use), Computation use (c-use)

(1, (2, f), x), (1, (2, t), x), (1, 3, x), (1, (4, t), x), (1, (4, f), x), (1, (5, t), x), (1, (5, f), x), (1, 6, x), (1, 7, x), (6,(5, f)x), (6,(5,t)x), (6, 6, x), (3, 8, a), (7, 8, a), (3, 8, a), (7, 8, a)

### **Definition**

Definition of a variable is the occurrence of a variable when the value is bound to the variable. In the above code, the value gets bound in the first statement and then start to flow.

* If(x>0) is statement 2 in which value of x is bound with it.  
  Association of statement 2 is (1, (2, f), x), (1, (2, t.)
* a= x+1 is statement 3 bounded with the value of x  
  Association of statement 3 is (1, 3, x)

**All definitions coverage**

(1, (2, f), x), (6, (5, f) x), (3, 8, a), (7, 8, a).

### **Predicate use (p-use)**

If the value of a variable is used to decide an execution path is considered as predicate use (p-use). In control flow statements there are two

Statement 4 if (x<=0) is predicate use because it can be predicate as true or false. If it is true then if (x<1),6x=x+1; execution path will be executed otherwise, else path will be executed.

### **Computation use (c-use)**

If the value of a variable is used to compute a value for output or for defining another variable.

**Statement 3** a= x+1       (1, 3, x)  
**Statement 7** a=x+1       (1, 7, x)  
**Statement 8** print a       (3, 8, a), (7, 8, a).

These are **Computation use** because the value of x is used to compute and value of a is used for output.

**All c-use coverage**

(1, 3, x), (1, 6, x), (1, 7, x), (6, 6, x), (6, 7, x), (3, 8, a), (7, 8, a).

**All c-use some p-use coverage**

(1, 3, x), (1, 6, x), (1, 7, x), (6, 6, x), (6, 7, x), (3, 8, a), (7, 8, a).

**All p-use some c-use coverage**

(1, (2, f), x), (1, (2, t), x), (1, (4, t), x), (1, (4, f), x), (1, (5, t), x), (1, (5, f), x), (6, (5, f), x), (6, (5, t), x), (3, 8, a), (7, 8, a).

After collecting these groups, (By examining each point whether the variable is used at least once or not) tester can see all statements and variables are used. The statements and variables which are not used but exist in the code, get eliminated from the code.

# **Control Flow Testing**

Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. It is mostly used in unit testing. Test cases represented by the control graph of the program.

**Control Flow Graph** is formed from the node, edge, decision node, junction node to specify all possible execution path.

## Notations used for Control Flow Graph

1. Node
2. Edge
3. Decision Node
4. Junction node

## Node

Nodes in the control flow graph are used to create a path of procedures. Basically, it represents the sequence of procedures which procedure is next to come so, the tester can determine the sequence of occurrence of procedures.

We can see below in example the first node represent the start procedure and the next procedure is to assign the value of n after assigning the value there is decision node to decide next node of procedure as per the value of n if it is 18 or more than 18 so Eligible procedure will execute otherwise if it is less than 18 Not Eligible procedure executes. The next node is the junction node, and the last node is stop node to stop the procedure.

Skip Ad

## Edge

Edge in control flow graph is used to link the direction of nodes.

We can see below in example all arrows are used to link the nodes in an appropriate direction.

## Decision node

Decision node in the control flow graph is used to decide next node of procedure as per the value.

We can see below in example decision node decide next node of procedure as per the value of n if it is 18 or more than 18 so Eligible procedure will execute otherwise if it is less than 18, Not Eligible procedure executes.

## Junction node

Junction node in control flow graph is the point where at least three links meet.

### **Example**

1. **public** **class** VoteEligiblityAge{
3. **public** **static** **void** main(String []args){
4. **int** n=45;
5. **if**(n>=18)
6. {
7. System.out.println("You are eligible for voting");
8. }  **else**
9. {
10. System.out.println("You are not eligible for voting");
11. }
12. }
13. }

**Diagram - control flow graph**

Control Flow Testing in white box testing Link

The above example shows eligibility criteria of age for voting where if age is 18 or more than 18 so print message "You are eligible for voting" if it is less than 18 then print "You are not eligible for voting."

Program for this scenario is written above, and the control flow graph is designed for the testing purpose.

In the control flow graph, start, age, eligible, not eligible and stop are the nodes, n>=18 is a decision node to decide which part (if or else) will execute as per the given value. Connectivity of the eligible node and not eligible node is there on the stop node.

Test cases are designed through the flow graph of the programs to determine the execution path is correct or not. All nodes, junction, edges, and decision are the essential parts to design test cases.

# **Branch Coverage Testing**

Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once. Branch coverage technique is a whitebox testing technique that ensures that every branch of each decision point must be executed.

However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two. Decision coverage technique covers all branches of each decision point whereas branch testing covers all branches of every decision point of the code.

In other words, branch coverage follows decision point and branch coverage edges. Many different metrics can be used to find branch coverage and decision coverage, but some of the most basic metrics are: finding the percentage of program and paths of execution during the execution of the program.

Like decision coverage, it also uses a control flow graph to calculate the number of branches.

Branch Coverage

## How to calculate Branch coverage?

There are several methods to calculate Branch coverage, but pathfinding is the most common method.

In this method, the number of paths of executed branches is used to calculate Branch coverage. Branch coverage technique can be used as the alternative of decision coverage. Somewhere, it is not defined as an individual technique, but it is distinct from decision coverage and essential to test all branches of the control flow graph.

**Let's understand it with an example:**

1. Read X
2. Read Y
3. IF X+Y > 100 THEN
4. Print "Large"
5. ENDIF
6. If X + Y<100 THEN
7. Print "Small"
8. ENDIF

This is the basic code structure where we took two variables X and Y and two conditions. If the first condition is true, then print "Large" and if it is false, then go to the next condition. If the second condition is true, then print "Small."

## Control flow graph of code structure

Branch Coverage

In the above diagram, control flow graph of code is depicted. In the first case traversing through "Yes "decision, the path is **A1-B2-C4-D6-E8**, and the number of covered edges is 1, 2, 4, 5, 6 and 8 but edges 3 and 7 are not covered in this path. To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

**Path 1** - A1-B2-C4-D6-E8

**Path 2** - A1-B3-5-D7

Branch Coverage (BC) = Number of paths

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Covered Branches** | **Path** | **Branch coverage** |
| Yes | 1, 2, 4, 5, 6, 8 | A1-B2-C4-D6-E8 | 2 |
| No | 3,7 | A1-B3-5-D7 |

# **Statement Coverage Testing**

Statement coverage is one of the widely used software testing. It comes under white box testing.

Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code.

Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.

Statement Coverage 

In white box testing, concentration of the tester is on the working of internal source code and flow chart or flow graph of the code.

28.3M

557

Difference between JDK, JRE, and JVM

Generally, in the internal source code, there is a wide variety of elements like operators, methods, arrays, looping, control statements, exception handlers, etc. Based on the input given to the program, some code statements are executed and some may not be executed. The goal of statement coverage technique is to cover all the possible executing statements and path lines in the code.

Let's understand the process of calculating statement coverage by an example:

Here, we are taking source code to create two different scenarios according to input values to check the percentage of statement coverage for each scenario.

## Source Code Structure:

* Take input of two values like a=0 and b=1.
* Find the sum of these two values.
* If the sum is greater than 0, then print "This is the positive result."
* If the sum is less than 0, then print "This is the negative result."

1. input (**int** a, **int** b)
2. {
3. Function to print sum of these integer values (sum = a+b)
4. If (sum>0)
5. {
6. Print (This is positive result)
7. } **else**
8. {
9. Print (This is negative result)
10. }
11. }

So, this is the basic structure of the program, and that is the task it is going to do.

Now, let's see the two different scenarios and calculation of the percentage of Statement Coverage for given source code.

**Scenario 1:**  
**If a = 5, b = 4**

1. print (**int** a, **int** b) {
2. **int** sum = a+b;
3. **if** (sum>0)
4. print ("This is a positive result")
5. **else**
6. print ("This is negative result")
7. }

In scenario 1, we can see the value of sum will be 9 that is greater than 0 and as per the condition result will be "**This is a positive result.**" The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 5.

1. Total number of statements = 7
2. Number of executed statements = 5

Statement Coverage link

1. Statement coverage = 5/7\*100
2. = 500/7
3. = 71%

Statement Coverage link

Likewise, in scenario 2,

**Scenario 2:**  
**If A = -2, B = -7**

1. print (**int** a, **int** b) {
2. **int** sum = a+b;
3. **if** (sum>0)
4. print ("This is a positive result")
5. **else**
6. print ("This is negative result")
7. }

In scenario 2, we can see the value of sum will be -9 that is less than 0 and as per the condition, result will be "**This is a negative result.**" The statements highlighted in yellow color are executed statements of this scenario.

To calculate statement coverage of the first scenario, take the total number of statements that is 7 and the number of used statements that is 6.

Total number of statements = 7  
Number of executed statements = 6

Statement Coverage link

1. Statement coverage = 6/7\*100 <br>
2. = 600/7
3. = 85%

Statement Coverage link

But, we can see all the statements are covered in both scenario and we can consider that the overall statement coverage is 100%.

Statement Coverage link

So, the statement coverage technique covers dead code, unused code, and branches.

# **Decision Coverage Testing**

Decision coverage technique comes under white box testing which gives decision coverage to Boolean values. This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the statements like **do while statement, if statement and case statement** (Control flow statements), it is considered as decision point because there are two outcomes either true or false.

Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.

Generally, a decision point has two decision values one is true, and another is false that's why most of the times the total number of outcomes is two. The percent of decision coverage can be found by dividing the number of exercised outcome with the total number of outcomes and multiplied by 100.

Decision Coverage technique in whitebox testing link

In this technique, it is tough to get 100% coverage because sometimes expressions get complicated. Due to this, there are several different methods to report decision coverage. All these methods cover the most important combinations and very much similar to decision coverage. The benefit of these methods is enhancement of the sensitivity of control flow.

1.9M

Competitive questions on Structures

We can find the number of decision coverage as follows.

**Let's understand it by an example.**

Consider the code to apply on decision coverage technique:

1. Test (**int** a)
2. {
3. If(a>4)
4. a=a\*3
5. Print (a)
6. }

**Scenario 1:**  
**Value of a is 7 (a=7)**

1. Test (**int** a=7)
2. { **if** (a>4)
3. a=a\*3
4. print (a)
5. }

The code highlighted in yellow is executed code. The outcome of this code is "True" if condition (a>4) is checked.

Control flow graph when the value of a is 7.

Decision Coverage technique in whitebox testing link

Calculation of Decision Coverage percent:

Decision Coverage technique in whitebox testing link

1. Decision Coverage = ½\*100  (Only "True" is exercised)
2. =100/2
3. = 50
4. Decision Coverage is 50%

**Scenario 2:**  
**Value of a is 3 (a=3)**

1. Test (**int** a=3)
2. { **if** (a>4)
3. a=a\*3
4. print (a)
5. }

The code highlighted in yellow will be executed. The outcome of this code is ?False? if condition (a>4) is checked.

Control flow graph when the value of a is 3

Decision Coverage technique in whitebox testing link

Calculation of Decision Coverage percent:

Decision Coverage technique in whitebox testing link

1. = ½\*100  (Only "False" is exercised) <br>
2. =100/2
3. = 50
4. Decision Coverage = 50%

## Result table of Decision Coverage:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Value of A** | **Output** | **Decision Coverage** |
| 1 | 3 | 3 | 50% |
| 2 | 7 | 21 | 50% |