CHAPTER 3

Unit Testing

Boundary Value Testing

Syllabus Topic : Unit Testing

3.1 Unit Testing

Q. 3.1.1 Explain Unit Testing. (Ref. Sec. 3.1) **(5 Marks)**

Unit testing focuses verification effort on the smallest unit of software design - the software component or module.

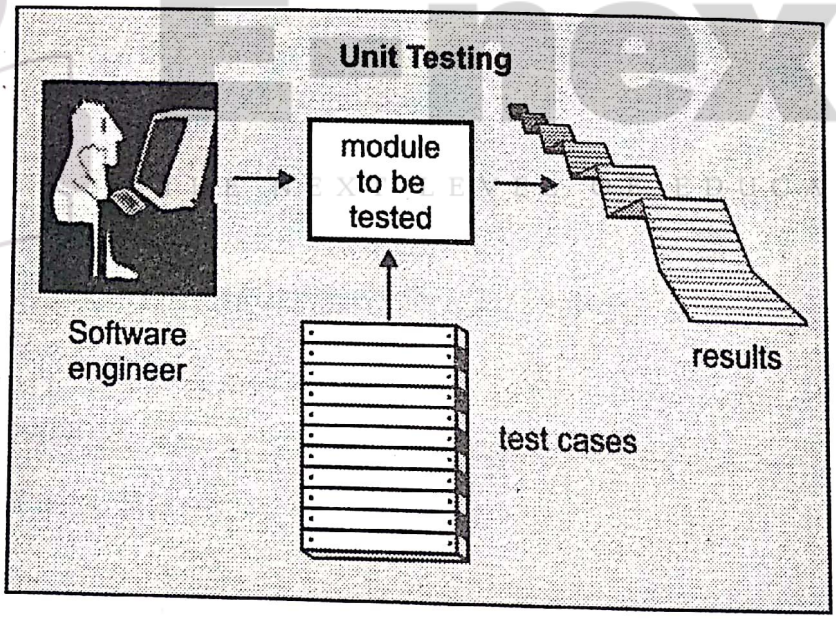

Fig. 3.1.1

3.1.1 Unit Test Considerations

The module interface is tested to ensure that information properly flows into and out of the program unit under test.

- The local data structure is checked to ensure that the data stored temporarily retains its integrity during all steps in an algorithm's execution.
- Boundary conditions are tested to make sure that the module operates properly at boundaries established to restrict processing.
- All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- And finally, all error handling paths are tested.

3.1.2 Unit Test Procedures

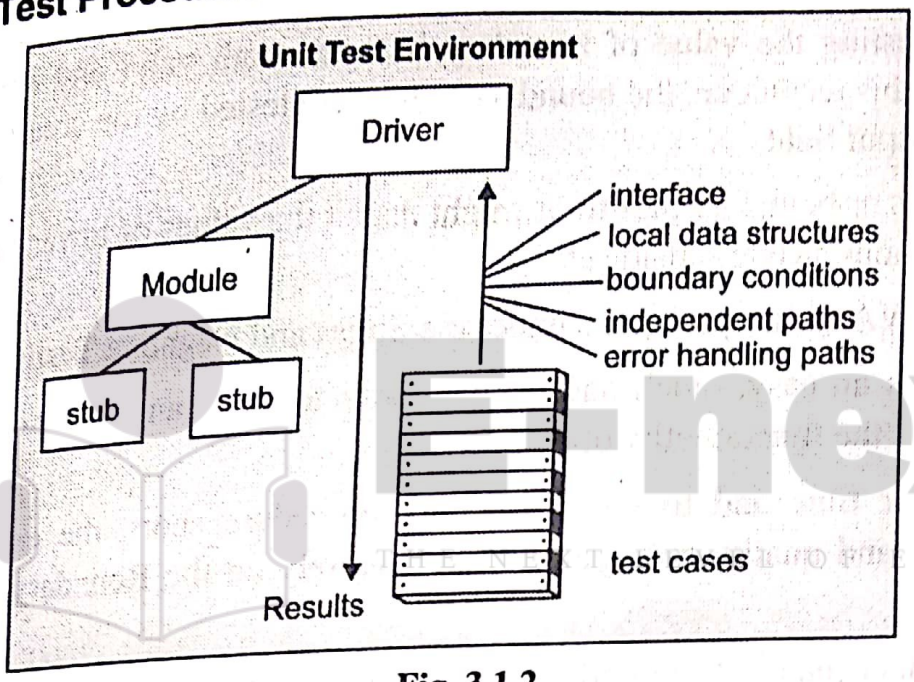

Fig. 3.1.2

- Each test case should be combined with a set of expected results. Because a component is not a stand-alone program, driver and/or stub software must be developed for each unit test.
- A driver is nothing more than a "main program" that accepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs replace modules that are subordinate (called by) the component to be tested.
- A stub or "dummy subprogram" uses the subordinate modules interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.

Syllabus Topic : Boundary Value Testing

3.2 Boundary Value Testing

Q.3.2.1

testing, the **Boundary Value Analysis (BVA) or Boundary value testing** is a **black box test** design technique based on test cases. This technique is applied to see if there are any **bugs** at the boundary of the input domain. Thus, with this method, there is no need of looking for these errors at the center of this input.

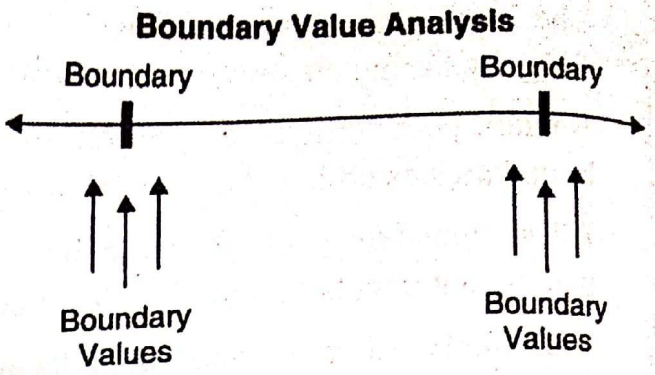

Fig. 3.2.1

- BVA helps in testing the value of boundary between both valid and invalid boundary partitions. With this technique, the boundary values are tested by the creation of test cases for a particular input field.
- The extreme ends or boundary partitions might depict the values of lower-upper, start-end, maximum-minimum, inside-outside etc.
- In general, the BVA technique comes under the Stress and Negative Testing.
- This technique is an easy, quick and brilliant way to catch any input errors that might occur to interrupt the functionality of a program.
- So, to save their time and to cut the testing procedure short, the experts delivering software testing and quality management services rely on the Boundary Value Analysis method.
- For testing of data related to boundaries and ranges, the method is considered as a very suitable one.

.2.1 An Example of Boundary Value Analysis

Consider the testing of a software program that takes the integers ranging between the values of -100 to +100. In such a case, three sets of the valid equivalent partitions are taken, which are – the negative range from -100 to -1, zero (0), and the positive range from 1 to 100.

Each of these ranges has the minimum and maximum boundary values. The Negative range has a lower value of -100 and the upper value of -1. The Positive range has a lower value of 1 and the upper value of 100

While testing these values, one must see that when the boundary values for each partition are selected, some of the values overlap. So, the overlapping values are bound to appear in the test conditions when these boundaries are checked.

These overlapping values must be dismissed so that the redundant test cases can be eliminated.

- So, the test cases for the input box that accepts the integers between -100 and +100 through BVA are :
  - Test cases with the data same as the input boundaries of input domain: -100 and +100 in our case.
  - Test data having values just below the extreme edges of input domain: -101 and 99
  - Test data having values just above the extreme edges of input domain: -99 and 101
- This is a very basic example to understand the BVA testing technique!
- With this technique, it is quite easy to test a small set of data in place of testing the whole lot of data sets. This is why, in software testing and quality management services, this method of testing is adopted more often.

3.2.2 Evaluation of Boundary Value Analysis as a Software Testing Technique

- In BVA, the software testers have found a fairly simple and correct testing method. This technique can be one of the most important testing techniques if used with care and correctness.
- However, the technique is a little limiting when there are some issues with variable dependency or when a foresight is needed for the functionality of a system.
- With its inexpensive operations and more apt ways of using it, the BVA can become one of the most used testing techniques in the times to come.
- But even today, many of the companies and experts delivering the software testing services, have adopted the BVA methods of testing.

3.2.3 Advantages of Boundary Value Analysis

Q. 3.2.2 What are the advantages and disadvantages of BVA ? **(5 Marks)**
(Ref. Secs. 3.2.3 and 3.2.4)

- The BVA technique of testing is quite easy to use and remember because of the uniformity of identified tests and the automated nature of this technique.
- One can easily control the expenses made on the testing by controlling the number of identified test cases. This can be done with respect to the demand of the software that needs to be tested.
- BVA is the best approach in cases where the functionality of a software is based on numerous variables representing physical quantities.
- The technique is best at revealing any potential UI or user input troubles in the software.
- The procedure and guidelines are crystal clear and easy when it comes to determining the test cases through BVA.
- The test cases generated through BVA are very small.

3.2.4 Disadvantages of Boundary Value Analysis

Q. 3.2.3 What are the advantages and disadvantages of BVA ? (Ref. Secs. 3.2.3 and 3.2.4) **(5 Marks)**

- This technique sometimes fails to test all the potential input values. And so, the results are unsure.
- The dependencies with BVA are not tested between two inputs.
- This technique doesn't fit well when it comes to Boolean Variables.
- It only works well with independent variables that depict quantity.

Syllabus Topic : Robust Boundary Value Testing

3.3 Robust Boundary Value Testing

Q. 3.3.1 Explain Robust Boundary Value testing. (Ref. Sec. 3.3) **(5 Marks)**

In robustness testing, software is tested by giving invalid values as inputs. Robustness testing is usually done to test exception handling. In robust boundary value testing, we make combinations in such a way that some of the invalid values are also tested as input.

☞ **Robust Boundary value testing on 3 variables**

Suppose we have 3 variables X, Y and Z to test

The range of X : 0 to 100

The range of Y : 20 to 60

The range of Z : 80 to 100

	x	y	z
Min –	– 1	19	79
Min	0	20	80
Min +	1	21	81
Nominal	50	40	90
Max –	99	59	99
Max	100	60	100
Max +	101	61	101

Fig. 3.3.1 : Testing points detected in Simple Robust Boundary Value Testing

3.3.1 Test Cases

Total Test cases = (Number of variables * Number of testing points without nominal) + (1 for Nominal)

These testing points are min-, min, min+, max- and max and max+

19 = (3*6)+1

- We can generate 19 test cases from both variables X, Y, and Z.
- There are total 3 variables X, Y and Z
- There are 6 possible values like min-, min, min+, max-, max and max+1 is for nominal

Logic

- When we make test cases, we will fix the nominal value of two variables and change the values of the third variable.
- For example, we will fix nominal values of X and Y and make a combination of these values with each value of Z variable.
- Fix nominal values of X and Y are 50,40, and we will compare these two values with 79, 80,81,90,99,100 and 101.
- We will fix nominal values of X and Z and will make a combination of these values with each value of Y variable.
- Fix nominal values of X and Z are 50, 90, and we will make a combination of these two values with 19, 20, 21, 40, 59, 60 and 61.
- We will fix nominal values of Y and Z and will make a combination of these values with each value of X variable.
- Fix nominal values of Y and Z are 40, 90, and we will make a combination of these two values with -1, 0, 1, 50, 99, 100 and 101.

Test Case#	X	Y	Z	Comment
1	50	40	79	Fix Nominal of X and Y
2	50	40	80	Fix Nominal of X and Y
3	50	40	81	Fix Nominal of X and Y
4	50	40	90	Fix Nominal of X and Y
5	50	40	99	Fix Nominal of X and Y
6	50	40	100	Fix Nominal of X and Y
7	50	40	101	Fix Nominal of X and Y
8	50	19	90	Fix Nominal of X and Z
9	50	20	90	Fix Nominal of X and Z
10	50	21	90	Fix Nominal of X and Z
11	50	59	90	Fix Nominal of X and Z
12	50	60	90	Fix Nominal of X and Z
13	50	61	90	Fix Nominal of X and Z
14	– 1	40	90	Fix Nominal of Y and Z
15	0	40	90	Fix Nominal of Y and Z
16	1	40	90	Fix Nominal of Y and Z
17	99	40	90	Fix Nominal of Y and Z
18	100	40	90	Fix Nominal of Y and Z
19	101	40	90	Fix Nominal of Y and Z

Fig. 3.3.2 : Test cases generated in Robust simple Boundary Value Testing

Syllabus Topic : Worst Case Boundary Value Testing

3.4 Worst Case Boundary Value Testing

Q. 3.4.1 Explain Worst case boundary value testing. (Ref. Sec. 3.4) **(5 Marks)**

- In Worst case boundary value testing, we make all combinations of each value of one variable with each value of another variable.

☞ Worst Boundary value testing on 2 variables

- Suppose we have two variables x1 and x2 to test.

- The range of x1 : 10 to 90.
- The range of x2 : 20 to 70.

	x1	x2
Min	10	20
Min +	11	21
Nominal	50	45
Max –	89	69
Max	90	70

Fig. 3.4.1 : Testing points detected in Worst Case Boundary Value Testing

3.4.1 Test cases

Total test cases = A*A

25 = 5*5

A = Number of testing points.

- These testing points are min, min+, nominal, max- and max.
- We can generate 25 test cases from both variables x1 and x2 by making a combination of each value of one variable with each value of another variable.

Test Case#	x1, x2	Test Case#	x1, x2	Test Case#	x1, x2
1	10, 20	2	10, 21	3	10, 45
4	10, 69	5	10, 70	6	11, 20
7	11, 21	8	11, 45	9	11, 96
10	11, 70	11	50, 20	12	50, 21
13	50, 45	14	50, 69	15	50, 70
16	89, 20	17	89, 21	18	89, 45
19	89, 69	20	89, 70	21	90, 20
22	90, 21	23	90, 45	24	90, 69
25	90, 70				

Fig. 3.4.2 : Test cases generated in Worst Case Boundary Value Testing

- There 25 test cases are enough to test the input values for these variables.

3.5 Special Value Testing

Q. 3.5.1 Explain special value testing. (Ref. Sec. 3.5) **(5 Marks)**

- Identical to any other software testing techniques, Special Value is defined and applied form of Functional Testing, which is a type of testing that verifies whether each function of the software application operates in conformance with the required specification. Each and every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.
- Special Value Testing, is probably the most extensively practiced form of functional testing which is most intuitive, and least uniform. This technique is performed by experienced professionals who are experts in this field and have profound knowledge about the test and the data required for it. They continuously participate and apply tremendous efforts to deliver appropriate test results to suit the client's requested demands.
- Furthermore, it utilizes domain knowledge and engineering judgment about program's "soft spots" to devise test cases. Even though special value testing is very subjective in the generation of test cases, it is often more effective on revealing faults in a software or a program.

3.5.1 Why Special Value Testing?

Special Value testing has several reasons which makes it the best option for testing programs, like :

- The testing executed by Special Value Testing technique is based on past experiences, which validates that no bugs or defects are left undetected.
- Moreover, the testers are extremely knowledgeable about the industry and use this information while performing Special Value testing.
- Another benefit of opting Special Value Testing technique is that it is Ad-hoc in nature. There are no guidelines used by the testers other than their "best engineering judgment".
- The most important aspect of this testing is that, it has had some very valuable inputs and success in finding bugs and errors while testing a software.

3.5.2 Disadvantages of Special Value Testing

- Special Value Testing can be very dependent on the abilities of the tester. However, finding industry experts and knowledgeable testers may be expensive as well as difficult, which is the only negative feature or drawback of this type of testing.

3.5.3 Is Special Value Testing a extension of Boundary Value Analysis ?

- Special Value Testing is an extension of Boundary Value Analysis testing (Input Domain Testing). Here, test cases are generated using the extremes of the input domain. As suggested by its name, Boundary Value testing technique is used to identify errors and bugs on the boundary rather than finding those which exists in the centre of the input data.
- With the exception of Special Value Testing, the test methods based on the boundary values of a program are mostly rudimentary.

Syllabus Topic : Example of Boundary Value Analysis

3.5.4 Example of Boundary Value Analysis

Q. 3.5.2 Give some examples of boundary value analysis (Ref. Sec. 3.5.4) (5 Marks)

- Consider a printer that has an input option of the number of copies to be made, from 1 to 99.
- To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case).

Syllabus Topic : Random Testing

3.6 Random Testing

Q. 3.6.1 Explain Random testing / Monkey testing. (Ref. Sec. 3.6) (5 Marks)

- A software testing technique, where the user tests the application by providing random inputs, is known as Monkey or Random Testing. It is a functional black box testing, which is performed when there is not enough time to write and execute the tests.
- Monkey testing, which is technically known as stochastic testing, can be performed for desktop, web and mobile applications and it is usually implemented as random, automated unit tests. This type of testing gives us an advantage of easily estimating software reliability from test outcomes.
- In this form of testing, the test inputs are randomly generated according to an operational profile and failure times are recorded. The data that is generated or obtained from random testing can then be further used to estimate products reliability.
- Other testing methods cannot be used in this way to estimate software reliability. Use of inputs of random testing/monkey testing can save some of the time and effort that more

thoughtful test input methods require. Mostly performed automatically, in monkey testing the user enters any random invalid input and checks its behavior.

- Moreover, it does not follow any predefined test cases or strategy and hence works on tester's mood and gut feeling. Monkey testing/Random testing works very well when doing load and stress testing.

3.6.1 Advantages of Monkey Testing

- With the assistance of Monkey testing one can identify some out of the box errors.
- It is testing technique that is easy to set up and execute.
- Monkey testing can be done by "not so skilled" testers.
- It is a great technique to test the reliability of the software.
- One can easily identify bugs with the assistance of this testing, which may have higher impact on the software's effectiveness and performance.
- This type of software testing is extremely cost effective.

3.6.2 Disadvantages of Monkey Testing

- The tests carried out during monkey testing are so random that it is either not possible or very difficult to recreate any bug.
- It's very difficult and time consuming to analyse the unexpected issues found during the monkey testing.
- Testers have difficulty in defining the exact test scenarios and they also cannot assure the accuracy of test cases.
- Monkey testing may consume lots of time before finding a bug because it does not have any predefined tests.

Syllabus Topic : Guidelines for Boundary Value Testing

3.7 Guidelines for Boundary Value Testing

Q. 3.7.1 What are the guidelines for boundary value testing? (Ref. Sec. 3.7) (5 Marks)

- A variation of the Boundary Value Analysis and the most widely practiced form of functional testing, Special Value testing is the best option for testing program or a software.
- This "ad-hoc" approach of testing occurs when the testers use their domain knowledge or experience with similar programs or information about "soft spots" to devise test cases.
- While subjective it can be extremely fruitful in detecting errors and defects as it also considers the past.

Syllabus Topic : Equivalence Classes

3.8 Equivalence Classes

Q. 3.8.1 What are equivalence classes? (Ref. Sec. 3.8) **(5 Marks)**

Equivalence classes assists the team in getting accurate and expected results, within the limited period of time and while covering large input scenarios.

Syllabus Topic : Traditional Equivalence Class Testing

3.8.1 Traditional Equivalence Class Testing

Q. 3.8.2 Explain Traditional Equivalence Class Testing. (Ref. Sec. 3.8.1) **(5 Marks)**

- **Programmer arrogance :**
  - in the 1960s and 1970s, programmers often had very detailed input data requirements.
  - if input data didn't comply, it was the user's fault
  - the popular phrase - Garbage In, Garbage Out (GIGO)
- **Programs from this era soon developed defenses**
  - (many of these programs are STILL legacy software)
  - as much as 75% of code verified input formats and values
- "Traditional" equivalence class testing focuses on detecting invalid input.

(almost the same as our "weak robust equivalence class testing")

Syllabus Topic : Improved Equivalence Class Testing

3.8.2 Improved Equivalence Class Testing

Q. 3.8.3 Explain Improved Equivalence Class Testing. (Ref. Sec. 3.8.2) **(5 Marks)**

- Equivalence Class Testing, which is also known as Equivalence Class Partitioning (ECP) and Equivalence Partitioning, is an important software testing technique used by the team of testers for grouping and partitioning of the test input data, which is then used for the purpose of testing the software product into a number of different classes.

- These different classes resemble the specified requirements and common behaviour or attribute(s) of the aggregated inputs. Thereafter, the test cases are designed and created based on each class attribute(s) and one element or input is used from each class for the test execution to validate the software functioning, and simultaneously validates the similar working of the software product for all the other inputs present in their respective classes.

☞ **Example**

- Let us further clear the concept of equivalence class testing with the assistance of the following example:
- Consider a software application, which takes not less than two digit number and not more than 3 digit number, for its execution. Given below is the huge amount of input to test and validate the functioning of the software application.

  1, 2, 3,.....1498, 1499, 1500.
- Now, as per the requirement specifications, these inputs are grouped together to form some classes. Now, instead of testing 1500 inputs, we have formed 4 classes and are accordingly dividing the inputs into a category of valid and invalid inputs, which reduces the work of the test case preparation.
- A single element, chosen from each class, as a test input, represents the whole class. For example, number 121 is used from the class "three digit numbers" as the test input. On using 121, it was found that software application functions properly and passes the test. Therefore, it is assumed that all the other numbers of the class "three digit number" will work well for the software application. And if the software fails the test, then it is assumed, that all the three digit numbers will generate error in the software application.
- Similarly, a number 7 is used from the class "single digit numbers", i.e. invalid input. It is expected by the software application, to generate error, on using number 7, and if does then the software is functioning appropriately, and it is also assumed that the remaining single digit number will also produces error(s), in the software application.

3.8.3 Features of Equivalence Class Testing (ECT)

ECT can be termed as a logical step in the model of **functional testing**. It improves the quality of test cases, which further enhances the quality of testing, by removing the vast amount of redundancy and gaps that appear in the boundary value testing. Other features of this testing technique are :

- It is a **black box testing** technique which restricts the testers to examine the software product, externally.
- Also known by the name of **equivalence class partitioning**, it is used to form groups of test inputs of similar behaviour or nature.

- Based on the approach, if one member works well in the family then the whole family is considered to function well and if one member fails, whole family is rejected.
- Test cases are based on classes, not on every input, thereby reduces the time and efforts required to build large number of **test cases.**
- It may be used at any level of testing i.e. unit, integration, system & acceptance.
- It is good to go for the ECT, when the input data is available in terms of intervals and sets of discrete values.
- However, there is no such specific rule to use only input from each class. Based on the experience and need, a tester may opt for more than one input.
- It may results into good amount of decrease in the redundant test cases, if implemented properly.
- It may not work well with the boolean or logical types variables.
- A mixed combination of Equivalence class testing and boundary value testing produces effective results.
- The fundamental concept of equivalence class testing/partition comes from the equivalence class, which further comes from equivalence relations.

3.8.4 Equivalence Class Testing Types

Q. 3.8.4 Explain various types of Equivalence Class Testing. (Ref. Sec. 3.8.4) **(5 Marks)**

The equivalence class testing can be categorized into four different types, which are integral part of testing and cater to different data set. These types of equivalence class testing are :

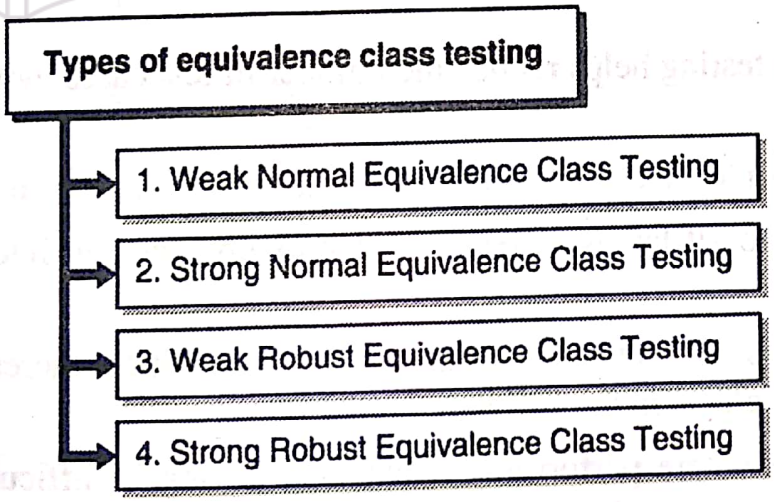

Fig. 3.8.1

1. Weak Normal Equivalence Class Testing

In this first type of equivalence class testing, one variable from each equivalence class is tested by the team. Moreover, the values are identified in a systematic manner. Weak normal equivalence class testing is also known as **single fault assumption.**

➜ 2. Strong Normal Equivalence Class Testing

Termed as **multiple fault assumption**, in strong normal equivalence class testing the team selects test cases from each element of the Cartesian product of the equivalence. This ensures the notion of completeness in testing, as it covers all equivalence classes and offers the team one of each possible combinations of inputs.

➜ 3. Weak Robust Equivalence Class Testing

Like weak normal equivalence, weak robust testing too tests one variable from each equivalence class. However, unlike the former method, it is also focused on testing test cases for invalid values.

➜ 4. Strong Robust Equivalence Class Testing

Another type of equivalence class testing, strong robust testing produces test cases for all valid and invalid elements of the product of the equivalence class. However, it is incapable of reducing the redundancy in testing.

3.8.5 Advantages and Disadvantages of Equivalence Class Testing

- Equivalence class testing or equivalence partitioning plays a potent role in reducing redundancy in testing and making the process agile and powerful.
- It is among those testing techniques that offer numerous benefits to the team and ensures compliance of the product with customer requirements.
- However, there are few drawbacks associated to this type of testing, which are listed below along with its various advantages.

☞ Advantages

- Equivalence class testing helps reduce the number of test cases, without compromising the test coverage.
- Reduces the overall test execution time as it minimizes the set of test data.
- It can be applied to all levels of testing, such as unit testing, integration testing, system testing, etc.
- Enables the testers to focus on smaller data sets, which increases the probability to uncovering more defects in the software product.
- It is used in cases where performing exhaustive testing is difficult but at the same time maintaining good coverage is required.

☞ Disadvantages

- It does not consider the conditions for boundary value.
- The identification of equivalence classes relies heavily on the expertise of testers.
- Testers might assume that the output for all input data set are correct, which can become a great hurdle in testing.

3.9 Edge Testing

Q. 3.9.1 Explain Edge testing. (Ref. Sec. 3.9) (5 Marks)

- When a programmer makes an error (mistake), which results in a defect in the software source code.
- If this defect is executed, system will produce wrong results, causing a failure. A defect can be called fault or bug.
- QA (Quality Assurance Team) person uses many test cases based on the requirement and also does create a test case for edge case or tail end case testing. This testing is known as when an user normally will not enter into this situation.

Example

- Consider simple example of inputting the employee information :

When you key in the employee information, the maximum age limit would be 70 or 80. But if you key in a year that results in employee age as more than 200, the software will not function correctly. This error should be caught while doing edge case testing. Even if we release this product to the client with out testing this case, it will continue to work until the user intentionally enters the age above 200.

Syllabus Topic : Guidelines for Equivalence Class Testing

3.10 Guidelines for Equivalence Class Testing

Q. 3.10.1 What are the guidelines for Equivalence Class Testing? (Ref. Sec. 3.10) (5 Marks)

By following a set of guidelines while implementing the process of testing, the team of testers can ensure better outputs from the tests and make sure all scenarios are being tested accurately. Therefore, listed below are some tips/guidelines for equivalence class testing:

- Use robust forms if the error conditions in the software product are of high priority.
- It can be used by the team in projects where the program function is complex.
- To ensure the accuracy and precision of equivalence class testing, define the input data in terms of intervals and sets of discrete values.
- Use of robust from is redundant of the implemented language is strongly types and when invalid values cause runtime errors in the system.

- The team needs to select one valid and one invalid input value each, if the input conditions are broken or not stated accurately.
- Establishing proper equivalence relation might require several tries and extra efforts of the team.

3.10.1 Difference between Equivalence Class Testing and Boundary Value Analysis

- Boundary value analysis and equivalence class testing are two strategies used for test case designing in black box testing, which makes it crucial for us to differentiate them from one another and define their specific relevance in software testing.
- The differences between these two are :

Equivalence Class Testing	Boundary Value Analysis
1. Equivalence Class Testing is a type of black box technique.	1. Next part of Equivalence Class Partitioning/Testing.
2. It can be applied to any level of testing, like unit, integration, system, and more.	2. Boundary value analysis is usually a part of stress & negative testing.
3. A test case design technique used to divide input data into different equivalence classes.	3. This test case design technique used to test boundary value between partitions.
4. Reduces the time of testing, while using less and effective test cases.	4. Reduces the overall time of test execution, while making defect detection faster and easy.
5. Tests only one from each partition of the equivalence classes.	5. Selects test cases from the edges of the equivalence classes.

Syllabus Topic : Observations

3.10.2 Observations

ECT is a widely used method that decreases the number of possible test cases that are required to a software product. Moreover, its ability to generate greater testing coverage, without compromising time and efforts, makes Equivalence Class Testing a popular testing technique worldwide.

Decision Table Testing

Syllabus Topic : Decision Tables

3.11 Decision Tables

Q. 3.11.1 What are decision tables? (Ref. Sec. 3.11) **(5 Marks)**

Q. 3.11.2 What is decision table testing ? (Ref. Sec. 3.11) **(5 Marks)**

Understanding the Decision Table

Also known as a Cause and effect table, a decision table is a representation of different combinations of inputs in line with their associated actions or outputs. They are used to fuel logic by embedding them with codes or program.

Role of decision table in testing

A decision table's property of dealing with large number of combinations of input with varied responses comes handy in the field of software testing.

Example to explain decision table testing

- Let us suppose as professional testers we are testing the efficacy of online ticket booking option for a railway app.

Input condition	Value 1	Value 2	Value 3
TRAINS FROM	F	T	T
TRAINS TO	T	F	T
TRAINS BUTTON	F	F	T

- In order to test the efficacy of the trains button, we will have to consider following case scenarios for enabling the trains button option which will take us to the list of all trains running between the different destinations.
- **Case 1 :** If we don't select any trains from the "TRAINS FROM" drop down value and select options only from the "TRAINS TO" drop down menu, the "TRAINS" button will remain disabled. This is represented by the false value for the Trains button in column designated as Value 1.
- **Case 2 :** If we don't select any trains from the "TRAINS TO" drop down value and select options only from the "TRAINS FROM" drop down menu, the "TRAINS" button will again remain disabled. This is represented by the false value for the Trains button in column designated as Value 2.

- **Case 3 :** If we select options from both the "TRAINS FROM" and the "TRAINS TO" menus, then only the "TRAINS" button will get enabled. This is illustrated by the truth values for all rows in column designated as Value3. Such tables are used in the form of iterations and serve as input tables for succeeding tabular cycles.
- The importance of decision table testing becomes more highlighted when we consider the fact that instead of a couple of input conditions to test, as shown in the above case, there are 100s of them with their innumerable combinations to consider, as seen with filling of online web forms.
- It goes without saying that not all the possible combinations will be utilised for testing but a select few is considered for testing the authenticity of various form filling based apps or websites.

Syllabus Topic : Decision table Technique

3.12 Decision table Technique

Q. 3.12.1 Explain Decision table techniques in detail. (Ref. Sec. 3.12) **(5 Marks)**

- It is a black box test design technique to determine the test scenarios for complex business logic.
- We can apply Equivalence Partitioning and Boundary Value Analysis techniques to only specific conditions or inputs.
- Although, if we have dissimilar inputs that result in different actions being taken or secondly, we have a business rule to test that there are different combination of inputs which result in different actions. We use decision table to test these kinds of rules or logic.

Why Decision table is important?

- Decision tables are very much helpful in test design technique - it helps testers to search the effects of combinations of different inputs and other software states that must correctly implement business rules.
- Also, provides a regular way of stating complex business rules, that's helpful for developers as well as for testers. Testing combinations can be a challenge, as the number of combinations can often be huge.
- It assists in development process with developer to do a better job. Testing with all combination might be unrealistic or unfeasible. We have to be happy with testing just a small subset of combinations but making the option of which combinations to test and which to leave out is also significant.
- If you do not have a efficient way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

- A decision table is basically an outstanding technique used in both testing and requirements management. It is **a structured exercise to prepare requirements when dealing with complex business rules.** Also, used in model complicated logic.

Syllabus Topic : Cause-and-Effect Graphing

3.13 Cause-Effect Graphing

Q. 3.13.1 Explain cause-effect graphing. (Ref. Sec. 3.13) **(5 Marks)**

- The Cause-Effect Graphing technique was invented by Bill Elmendorf of IBM in 1973. Instead of the test case designer trying to manually determine the right set of test cases, he/she models the problem using a cause-effect graph, and the software that supports the technique, BenderRBT, calculates the right set of test cases to cover 100% of the functionality.
- The cause-effect graphing technique uses the same algorithms that are used in hardware logic circuit testing. Test case design in hardware insures virtually defect free hardware.
- Cause-Effect Graphing also has the ability to detect defects that cancel each other out, and the ability to detect defects hidden by other things going right.
- The starting point for the Cause-Effect Graph is the requirements document. The requirements describe "what" the system is intended to do. The requirements can describe real time systems, events, data driven systems, state transition diagrams, object oriented systems, graphical user interface standards, etc.
- Any type of logic can be modeled using a Cause-Effect diagram. Each cause (or input) in the requirements is expressed in the cause-effect graph as a condition, which is either true or false. Each effect (or output) is expressed as a condition, which is either true or false.

A Simple Cause-Effect Graphing Example

- I have a requirement that says: "If A OR B, then C."
- The following rules hold for this requirement :
  - If A is true and B is true, then C is true.
  - If A is true and B is false, then C is true.
  - If A is false and B is true, then C is true.
  - If A is false and B is false, then C is false.
- The cause-effect graph that represents this requirement is provided in Fig. 3.13.1. The cause-effect graph shows the relationship between the causes and effects.

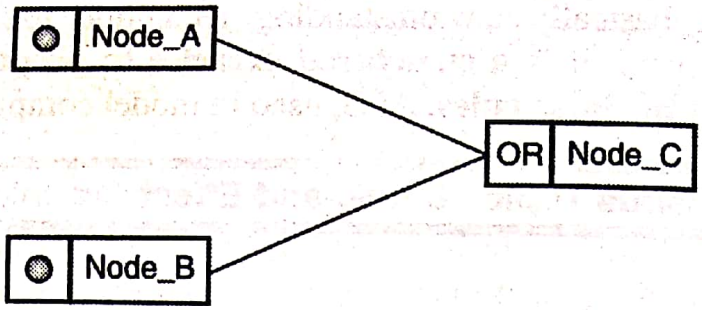

Fig. 3.13.1 : A Cause-Effect Graph

- In Fig. 3.13.1, A, B and C are called nodes. Nodes A and B are the causes, while Node C is an effect. Each node can have a true or false condition. The lines, called vectors, connect the cause nodes A and B to the effect node C.
- All requirements are translated into nodes and relationships on the cause-effect graph. There are only four possible relationships among nodes, and they are indicated by the following symbols :
  - If A always leads to C, there is a black line that connects the nodes.
  - If A or B lead to C, there is an "OR" relation.
  - If A and B lead to C, there is an "AND" relation.
  - If not A leads to C, there is a red line that connects nodes.

3.13.1 Conversion of Cause-effect graph into Decision Table

- The cause-effect graph is then converted into a decision table or "truth table" representing the logical relationships between the causes and effects. Each column of the decision table is a test case. Each test case corresponds to a unique possible combination of inputs that are either in a true state, a false state, or a masked state.
- Since there are 2 inputs to this example, there are $2 \times 2 = 4$ maximum combinations of inputs from which test cases can be selected.
- Fig. 3.13.2 represents the decision table derived for the cause-effect graph in Fig. 3.13.1.

	TEST #1	TEST #2	TEST #3
Causes :			
A	T	F	F
B	F	T	F
Effects :			
C	T	T	F

Fig. 3.13.2 : The Decision Table

3.13.2 Guidelines and Observations of Decision Table Testing

- Decision table testing makes it simple to convert multifarious business applications into simple test case scenarios.
- The tabular approach used here is simpler to understand and easy for anyone to built test cases.
- A better coverage achieved using decision table testing ensures that the test is quite exhaustive in nature.

Path Testing

3.14 Path Testing

3.14.1 Program Graphs

Q. 3.14.1 What are program graphs? (Ref. Sec. 3.14.1) **(5 Marks)**

- Given a program written in an imperative programming language, its program graph is a directed graph in which nodes are statements and statement fragments, and edges represent flow of control
- Two nodes are connected if execution can proceed from one to the other.

Triangle program text

```
1. output ("Enter 3 integers")
2. input (a, b, c)
3. output("Side a b c: ", a, b, c)
4. if (a < b) and (b <a+c) and (c <a+b)
5. then isTriangle← true
6. else isTriangle← false
7. fi
8. if isTriangle
9. then if (a = b) and (b = c)
```

10. else output ("equilateral")
11. else if (a ≠ b) and (a ≠ c) and (b ≠ c)
12. then output ("scalene")
13. else output("isosceles")
14. fi
15. fi
16. else output ("not a triangle")
17. fi

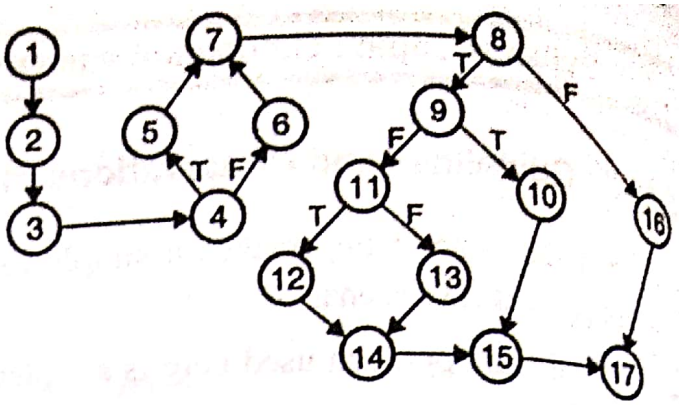

Fig. 3.14.1

Syllabus Topic : Decision to Decision (DD) Path

3.15 DD Paths

Q. 3.15.1 Explain DD Paths. (Ref. Sec. 3.15) **(5 Marks)**

☞ Informal Definition

A decision-to-decision path (DD-Path) is a path chain in a program graph such that

- Initial and terminal nodes are distinct
- Every interior node has indeg =1 and outdeg = 1
- The initial node is 2-connected to every other node in the path
- No instances of 1- or 3-connected nodes occur

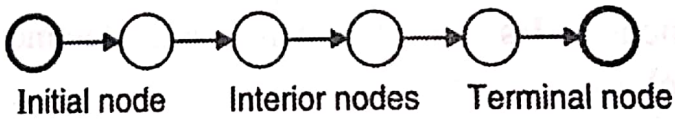

Fig. 3.15.1

☞ Formal Definition

A decision-to-decision path (DD-Path) is a chain in a program graph such that :

- **Case 1 :** consists of a single node with indeg=0
- **Case 2 :** consists of a single node with outdeg = 0
- **Case 3 :** consists of a single node with indeg ≥ 2 or outdeg ≥ 2
- **Case 4 :** consists of a single node with indeg = 1, and outdeg = 1
- **Case 5 :** it is a maximal chain of length ≥ 1

DD-Paths are also known as **segments.**

Triangle program DD-paths

Table 3.15.1

Nodes	Path	Case
1	First	1
2, 3	A	5
4	B	3
5	C	4
6	D	4
7	E	3
8	F	3
9	G	3

Nodes	Path	Case
10	H	4
11	I	3
12	J	4
13	K	4
14	L	3
15	M	3
16	N	4
17	Last	2

Syllabus Topic : Text Coverage Metrics

3.15.1 Program Text Coverage Metrics

Q. 3.15.2 What are the various program text coverage metrics? (Ref. Sec. 3.15.1) (5 Marks)

- **C0 :** Every Statement
- **C1 :** Every DD-path
- **C1p :** Every predicate to each outcome
- **C2 :** C1 coverage + loop coverage
- **Cd :** C1 coverage + every dependent pair of DD-paths
- **CMCC :** Multiple condition coverage
- **Cik :** Every program path that contains k loop repetitions
- **Cstat :** Statistically significant fraction of the paths
- **C∞ :** Every executable path

Syllabus Topic : Basis Path Testing

3.16 Basis Path Testing

Q. 3.16.1 Explain Basis Path Testing. (Ref. Sec. 3.16) (5 Marks)

- An important part of software engineering, testing ensures that no errors or issues exist in the software product that could compromise its safety, security, and even the financial investment of the client or the user.

- **Path Testing**, which is executed by the team during software testing life cycle (STLC), helps evaluate and verify various structural aspects of the product that enables the team to guarantee the effectiveness and the quality of the product.
- Therefore, to further elaborate on the importance of this testing technique, here is a detailed discussion on path testing and its various components.

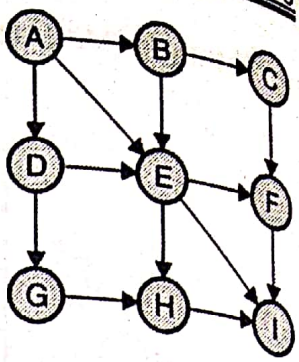

Fig. 3.16.1

What is Path Testing in Software Testing?

- Path testing refers to the checking of the paths through various processes, which are involved in the functioning of the software product or application and will later be executed. Basically, it is a type of structural testing that works on the source code.
- Path testing follows a process of control flow or workflow, rather than executing the process of testing on the basis of specifications. The main intent behind this type of testing is to ensure that every possible path has been covered and executed at least once, through the software program.

3.16.1 Features of Path Testing

An important software testing technique, path testing helps identify issues in a piece of code that can impact the functionality and performance of the product. Therefore, to get an insight into this testing, mentioned below are some of its features :

- It requires complete knowledge of the program's structure.
- Path testing is often used by software programmers to unit test the code of the software product.
- It is a white box testing technique.
- Helps in evaluating the internal flow of the software product.
- Effective for exploring large quantity of defects during unit testing.
- It calculates the number of tests to be executed based on the cyclomatic complexity and generates separate test cases for each path to be executed.
- It covers all the paths under the test. However, it does not identify all the defects, as there are many defects, which may arise due to certain reasons, such as :
  - Path followed by the process is not in order, which may result in an error in the software application. For example, a process is required to be executed, and the path to be followed by it should **A → B → C**. But it takes the wrong path i.e. **A → C → B**.
  - Mistakes introduced by the developer in the source code, such as inadvertently, forgotten to add processing in the code(s), which may lead to no path, in the direction of executing the code.

3.16.2 Path Testing Techniques

Q. 3.16.2 Which are the Path Testing techniques? (Ref. Sec. 3.16.2) (5 Marks)

Path testing is further categorized into three important techniques, each of which caters to a different aspect of the product and ensure its accuracy. From the testing workflow to its execution, these techniques play a critical role in the whole process and are hence defined in detail below :

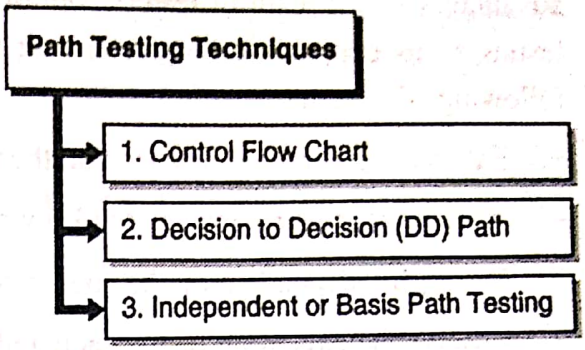

Fig. 3.16.2

1. Control Flow Chart

- Generally it involves the activity of representing the workflow of the process or program in the form of control chart or graph, which consists of nodes, links, edges, regions, etc.
- During this process, the flow graph or chart is constructed by the team by replacing the program control statements with equivalent diagrams.

2. Decision to Decision (DD) Path

This strategy involves selection of one, between the two decisions, for the purpose of path's execution. It is carried out with the help of the control flow chart and includes node, which inherits one of the following traits.

- Single node of in-degree = 0 i.e. initial node.
- Single node of out-degree = 0 i.e. terminal node
- Single node with in-degree and out-degree = 1.
- Single node with in-degree or out-degree > 2

Further, the maximum chain length in DD path > 1

3. Independent or Basis Path Testing

It is a widely used testing method that guarantees the complete branch coverage in the control flow graph.

- It involves the study and analysis of the control flow graph, which contributes towards the identification of the linearly independent paths. Thereafter, these identified paths are used to design and build test cases.
- Moreover, it includes conditional statements that are executed by the team depending on what conditions it suffices.
- Basis testing, in short, is used to maximize test coverage and to reduce the number of redundant tests.

3.16.3 Advantages of Path Testing

Similar to the miscellaneous software testing techniques, path testing also offers various advantages to the team of testers. Defining the flow of control from the starting point of the testing to its end, this testing is among the oldest types of structural testing, that offers the following advantages :

- Path testing is mainly focused on the program logic.
- Allows the team to analyze the flow of the program during the process of testing.
- It is highly dependent on the source code of the program.
- Ensures the execution of the each and every path, present in the software program, at least once.
- A structured type of testing technique that uses control flow graph and is independent of the specifications.

3.16.4 Disadvantages of Path Testing

Thought the list of benefits offered by path testing is non-exhaustive, it is crucial for us to mention the few disadvantages/drawbacks offered by this testing to help you get a thorough insight into its process and features. These disadvantages are :

- Presence of defects cannot be traced out by the path testing due to an error in the specifications.
- Path testing requires expert and skillful testers, with in-depth knowledge of programming and code.
- It is difficult to test all paths with this type of testing technique when the product becomes more complex.

Syllabus Topic : Guidelines and Observations

3.16.5 Guidelines and Observations

Q. 3.16.3 What are guidelines and observations of path testing? (Ref. Sec. 3.16.5) **(5 Marks)**

- There are several testing techniques like data flow and equivalence class testing that are performed by the team during STLC.
- However, path testing is implemented by them to identify various issues in the code, which can become a hindrance in testing.
- This testing, which is extensively effective during unit testing, helps improve the process of testing as well as the quality and functionality of the product.

Data Flow Testing

Syllabus Topic : Data Flow Testing

3.17 Data Flow Testing

Q. 3.17.1 Explain Data flow testing & its types. (Ref. Sec. 3.17) (5 Marks)

- It focuses on the data variables and their values, used in the programming logic of the software product, by making use of the control flow graph.
- Data flow testing is the form of white box testing and structural type testing, which generally keeps check at the points, where the data values are being received by the variables, and at the points, when it is called for use. It is used to fill the gap between the path testing and branch testing.
- The basic idea behind this form of testing, is to reveal the coding errors and mistakes, which may results in to improper implementation and usage of the data variables or data values in the programming code i.e. data anomalies, such as :
  - All the data variables, present in the programming code have been initialized or not,
  - Data variables which are put into use, have been, priorly initialized or not,
  - If the initialized data variables, has been used, at least once, in the programming code.

Syllabus Topic : Definition/Use of Testing

3.17.1 Definition/Use of Testing

Q. 3.17.1 How is data used in the programming code? (Ref. Sec. 3.17.1) (5 Marks)

Q. 3.17.2 Explain different types of data flow testing. (Ref. Sec. 3.17.2) (5 Marks)

How data is used in the programming code?

- Generally, data objects occurring in the programming life cycle goes through 3 phases :
  - **Definition :** Data variables are defined, created and initialized, along with the allocation of the memory to that particular data object.
  - **Usage :** Declared data variables may be used in the programming code, in two forms
  - As the part of the predicate(P), such as "If (A > B)"
  - In the computational form(C), when the data items are involved in the calculations to give some output.
  - **Deletion or Kill :** Memory allocated to the variables, gets freed and is put into for some other use.

3.17.2 Types to Perform Data Flow Testing

The process of data flow testing may be carried out through two different approach or methodology.

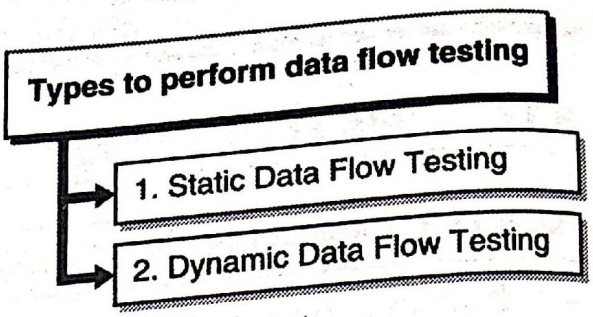

Fig. 3.17.1

➜ **1. Static Data Flow Testing**

- In static testing, study and analysis of code is done without performing the actual execution of the code such as wrong header files or library files use or syntax error.
- Generally, during this type of testing, d-u-k pattern, i.e. definition, usage and kill pattern of the data variables is monitored and observed with the help of control flow graph.

➜ **2. Dynamic Data Flow Testing**

It involves the execution of the code, to monitor and observe the intermediate results. It basically, looks after the coverage of data flow properties. This type of testing may comprise of following activities :

- *Identification of all d-u pairs, i.e. definition and usage in the code.*
- *Detecting feasible path between each definition and usage pair.*
- *Designing & creating sets of test cases for each path.*

3.17.3 Data Flow Testing Coverage

Q. 3.17.2 Explain data flow testing coverage. (Ref. Sec. 3.17.3) **(5 Marks)**

The coverage of data flow in terms of "sub-paths" and "complete path" may be categorized under following types :

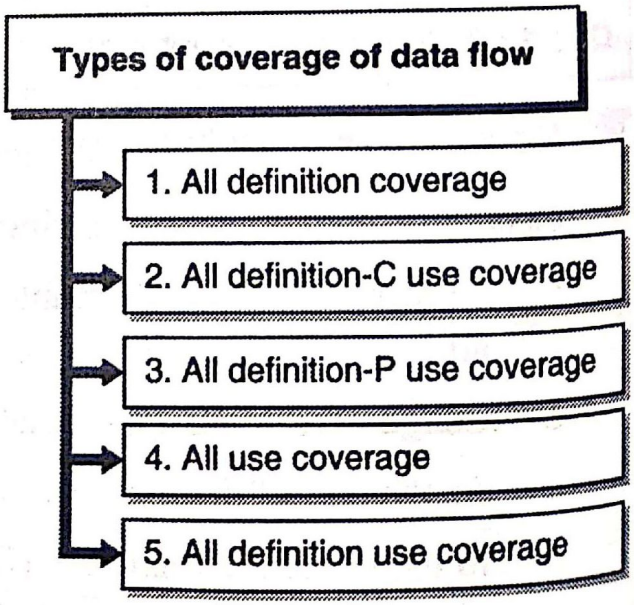

Fig. 3.17.2

➜ **1. All definition coverage**

Covers "sub-paths" from each definition to some of their respective use.

➜ **2. All definition-C use coverage**

"sub-paths" from each definition to all their respective C use.

3. **All definition-P use coverage**

"sub-paths" from each definition to all their respective P use.

4. **All use coverage**

Coverage of "sub-paths" from each definition to every respective use irrespective of types.

5 **All definition use coverage**

Coverage of "simple sub-paths" from each definition to every respective use.

Syllabus Topic : Slice-Based Testing and Program Slicing Tools

3.18 Slice Based Testing and Program Slicing Tools

Q. 3.18.1 Explain slice based testing & program slicing tools. (Ref. Sec. 3.18) (5 Marks)

- Given a program P, and a program graph G(P) in which statements and statement fragments are numbered, and a set V of variables in P, the slice on the variable set V at statement fragment n, written S(V, n), is the set node numbers of all statement fragments in P prior to n that contribute to the values of variables in V at statement fragment n.
- The idea of slices is to separate a program into components that have some useful meaning.
- We will include CONST declarations in slices.
- Five forms of usage nodes :
  - P-use (used in a predicate (decision))
  - C-use (used in computation)
  - O-use (used for output, e.g. writeln())
  - L-use (used for location, e.g. pointers)
  - I-use (iteration, e.g. internal counters)
- Two forms of definition nodes
  - I-def (defined by input, e.g. readln())
  - A-def (defined by assignment)
- For now, we presume that the slice S(V,n) is a slice on one variable, that is, the set V consists of a single variable, v
- If statement fragment n (in S(V,n)) is a defining node for v, then n is included in the slice
- If statement fragment n (in S(V,n)) is a usage node for v, then n is not included in the slice

- P-uses and C-uses of other variables are included to the extent that their execution affects the value of the variable v
- O-use, L-use, and I-use nodes are excluded from slices

☞ **What is Program Slicing?**

- Program slicing is a program analysis technique. It is mainly used during debugging and reengineering. It tries to eliminate all parts from the program that are not currently of interest to the programmer.
- Therefore, the programmer clicks on a statement (which is of interest to him, e.g. because the value of a variable is wrong) and the program slicer highlights all program statements that might have produced the erroneous value.
- Relationships between program entities like variable definitions and variable usages are visualized.

☞ **Tools**

- Code Surfer is a commercial tool for performing static slicing on C programs.
- Indus is University research tool for doing static code slicing on Java.
- Oberon Slicing Tool

3.18.1 Limitations of Data Flow Testing

With the advantages of using the exploratory testing in the agile environment, some limitations are also associated with it such as :

- Testers need to have sufficient knowledge of the programming.
- Time-consuming and costly process.

3.19 Exam Pack (Review Questions)

☞ **Syllabus Topic : Unit Testing**

Q. 1 Explain Unit Testing. *(Ans. : Refer section 3.1)* **(5 Marks)**

☞ **Syllabus Topic : Boundary Value Testing**

Q. 2 Explain Boundary Value Testing or Boundary Value Analysis. *(Ans. : Refer section 3.2)* **(5 Marks)**

Q. 3 What are the advantages and disadvantages of BVA ? *(Ans. : Refer sections 3.2.3 and 3.2.4)* **(5 Marks)**

Syllabus Topic : Robust Boundary Value Testing

Q. 4 Explain Robust Boundary Value testing. *(Ans. : Refer section 3.3)* **(5 Marks)**

Syllabus Topic : Worst Case Boundary Value Testing

Q. 5 Explain Worst case boundary value testing. *(Ans. : Refer section 3.4)* **(5 Marks)**

Syllabus Topic : Special Value Testing

Q. 6 Explain special value testing. *(Ans. : Refer section 3.5)* **(5 Marks)**

Syllabus Topic : Example of Boundary Value Analysis

Q. 7 Give some examples of boundary value analysis *(Ans. : Refer section 3.5.4)* **(5 Marks)**

Syllabus Topic : Random Testing

Q. 8 Explain Random testing / Monkey testing. *(Ans. : Refer section 3.6)* **(5 Marks)**

Syllabus Topic : Guidelines for Boundary Value Testing

Q. 9 What are the guidelines for boundary value testing? *(Ans. : Refer section 3.7)* **(5 Marks)**

Syllabus Topic : Equivalence Classes

Q. 10 What are equivalence classes? *(Ans. : Refer section 3.8)* **(5 Marks)**

Syllabus Topic : Traditional Equivalence Class Testing

Q. 11 Explain Traditional Equivalence Class Testing. *(Ans. : Refer section 3.8.1)* **(5 Marks)**

Syllabus Topic : Improved Equivalence Class Testing

Q. 12 Explain Improved Equivalence Class Testing. *(Ans. : Refer section 3.8.2)* **(5 Marks)**

Q. 13 Explain various types of Equivalence Class Testing. *(Ans. : Refer section 3.8.4)* **(5 Marks)**

Syllabus Topic : Edge Testing

Q. 14 Explain Edge testing. *(Ans. : Refer section 3.9)* **(5 Marks)**

Syllabus Topic : Guidelines for Equivalence Class Testing

Q. 15 What are the guidelines for Equivalence Class Testing? *(Ans. : Refer section 3.10)* **(5 Marks)**

☞ **Syllabus Topic : Decision Tables**

Q. 16 What are decision tables? *(Ans. : Refer section 3.11)* **(5 Marks)**

Q. 17 What is decision table testing? *(Ans. : Refer section 3.11)* **(5 Marks)**

☞ **Syllabus Topic : Decision table Technique**

Q. 18 Explain Decision table techniques in detail. *(Ans. : Refer section 3.12)* **(5 Marks)**

☞ **Syllabus Topic : Cause-and-Effect Graphing**

Q. 19 Explain cause-effect graphing. *(Ans. : Refer section 3.13)* **(5 Marks)**

☞ **Syllabus Topic : Guidelines and Observations of Decision Table Testing**

Q. 20 What are guidelines of Decision Table Testing? *(Ans. : Refer section 3.13.2)* **(5 Marks)**

☞ **Syllabus Topic : Path Testing, Program Graphs**

Q. 21 What are program graphs? *(Ans. : Refer section 3.14.1)* **(5 Marks)**

☞ **Syllabus Topic : Decision to Decision (DD) Path**

Q. 22 Explain DD Paths. *(Ans. : Refer section 3.15)* **(5 Marks)**

☞ **Syllabus Topic : Text Coverage Metrics**

Q. 23 What are the various program text coverage metrics? *(Ans. : Refer section 3.15.1)* **(5 Marks)**

☞ **Syllabus Topic : Basis Path Testing**

Q. 24 Explain Basis Path Testing. *(Ans. : Refer section 3.16)* **(5 Marks)**

Q. 25 Which are the Path Testing techniques? *(Ans. : Refer section 3.16.2)* **(5 Marks)**

☞ **Syllabus Topic : Guidelines and Observations**

Q. 26 What are guidelines and observations of path testing? *(Ans. : Refer section 3.16.5)* **(5 Marks)**

☞ **Syllabus Topic : Data Flow Testing**

Q. 27 Explain Data flow testing & its types. *(Ans. : Refer section 3.17)* **(5 Marks)**

Syllabus Topic : Definition/Use of Testing

Q. 28 How is data used in the programming code? *(Ans. : Refer section 3.17.1)* **(5 Marks)**

Q. 29 Explain different types of data flow testing. *(Ans. : Refer section 3.17.2)* **(5 Marks)**

Q. 30 Explain data flow testing coverage. *(Ans. : Refer section 3.17.3)* **(5 Marks)**

Syllabus Topic : Slice-Based Testing and Program Slicing Tools

Q. 31 Explain slice based testing & program slicing tools. *(Ans. : Refer section 3.18)* **(5 Marks)**