SOFTWARE ENGINEERING UNIT-III PART-3

Coding And Testing: Coding, Code Review, Software Documentation, Testing, Unit Testing, Black-Box Testing, White-Box Testing, Debugging, Program Analysis Tool, Integration Testing, Testing Object-Oriented Programs, System Testing, Some General Issues Associated with Testing

CODING INTRODUCTION

The purpose of coding is to create a set of instructions in a programming language so that computers execute them to perform certain operations. Coding is the software development phase that affects the testing and maintenance activities. The main goal of coding is to produce quality source codes that can reduce the cost of testing and maintenance. A clear, readable, and understandable source code will make testing, debugging, and maintenance tasks easier.

CODING FUNDAMENTALS

Although the quality of source codes depends on the skills and expertise of software engineers, the programming paradigms also play important roles in the production of quality source codes. The aim is to enhance productivity of the programmers and reduce the development time for competitive advantages. Also, this reduces the effort required in testing and maintenance tasks goals. Therefore, they should have a clear objective of their coding in the team for producing quality codes.

❖ **CODING PRINCIPLES**

Coding principles are closely related to the principles of design and modelling. Coding principles help programmers in writing an efficient and effective code, which is easier to test, maintain, and reengineer.

Some of the following coding principles can make a clear, readable and understandable are as follows

- **Information Hiding** – The information hiding hides the implementation details of data structures from the other modules. Information hiding is supported by data abstraction, which allows creating multiple instances of abstract data type. Thus, modifying a module with encapsulated data and function has minimum effect on other modules.
- **Structures Programming Feature -** Structured programming features linearize the program flow in some sequential way that the programs follow during their execution. The organization of program flow is achieved through the following three basic constructs of structured programming.
  Sequence: It provides sequential ordering of statements, i.e., S1, S2, S3, …Sn.
  Selection: It provides branching of statements using if-then-else, switch-case, etc.
  Iteration: A statement can be executed repeatedly using while-do, repeat-until, while etc

- **Maximize Cohesion And Minimize Coupling-** Writing modular programs with the help of functions, code, block, classes, etc., may increase dependency among modules in the software. Shared data should be used as little as possible. Minimizing dependencies among programs will maximize cohesion within modules. Thus, high cohesion and low coupling make a program clear, readable, and maintainable.
- **Code Reusability**- Code reusability allows the use of existing code several times. Similar to built-in library functions, reusable components can be constructed in modern programming languages and can be reused in later software developments. Minimum use of reusability enhances productivity and reliability, with reduced development time.
- **Simplicity, Extensibility, and Effortlessness** - A simple program always works better than a complicated program. Also, it can be made more reliable than a complex code. Programs should be extendable rather than being lengthy. Extendibility is different from modifiability; programs should be simple for better programming.
- **Code Verification**- The program logic and its correctness should be verified before moving toward testing. Therefore, test-driven development (TDD) environment is created for better code writing. In TDD, programming is done with testing a code. This reduces the testing and maintenance efforts.
- **Code Documentation**- Source codes are used by testers and maintainers. Therefore, programmers should add comments in source codes as and when required. A well-commented code helps to understand the code at the time of testing and maintenance.
- **Separation of Concern**- A program generally includes several functionalities in the system. Each of these functionalities is related to each other. Therefore, different functional requirements should be managed by distinct and loosely coupled modules of codes.
- **Follow Coding Standards, Guidelines, and Styles**- A source code with the standards and which is according to the programming style will have less adverse effect in the system. Therefore, programmers should focus on coding standards, guidelines, and good programming styles.

❖ CODING STYLES

Each programming language has its own pattern of programming. However, a programmer can make programs efficient and effective in his own way. Programmers can reduce the effort in the testing and maintenance tasks. There are certain programming styles that programmers can follow for writing the source code. Some of the common coding styles are discussed below.

- **Use of goto statement:** The goto statement should be used in a disciplined manner. It should be used with an “if” statement. The goto statement consumes more time in transferring control forward and backward as compared to sequential execution.
- **Use of control constructs:** If a programming language provides control constructs, then these are sufficient to use in source codes. However, preprocessor directories may be simulated as control constructs.

- **Define user-defined data types:** Traditional programming languages support limited data types such as integer, real, complex, characters, strings, logical, array, etc. Therefore, enumerated data types are defined by the user as and when required.
- **Program size:** A long program is difficult to understand and becomes unclear. Therefore, a long program should be partitioned into subprograms. A program contains a specification section, a documentation prologue, declarations, executable statement, and maybe exception handlers.
- **Information hiding:** In the importance of data encapsulation, only the accessible data should be visible to the external environment. The other data items should be hidden behind the function.
- **Commenting code:** Comments should be added to codes to explain the implementation details of the source code. Complex logics and data structures should be considered during commenting on source codes. The comments should be obvious and precise.
- **Use of temporary variable:** The use of temporary variables will consume more time in managing data rather than processing them. Also, a temporary variable will unnecessarily make the program complex. Also, any change in temporary variable will require changing other related parts of the program.
- **Replace repetitive expressions by calls to a common function:** Repetitive expressions consume memory and may take extra processing time. Therefore, such expressions can be converted into function calls because function calls return computed values within the main program.
- **Avoid patching bad code:** A bad code will always require maintenance and retesting of the program. Also, it reduces the code quality and damages the program structure. Therefore, instead of patching bad codes, a new code should be written for the purpose.

❖ CODING ERRORS

There are various categories of errors observed in programs. Errors are sometimes known as bugs. Some of the common types of errors are compilation errors (i.e., syntax errors), logical errors, and runtime errors.

- **Compilation errors:** Compilation errors prevent programs from running. A compiler checks and converts source codes into a binary language, which the computer understands. If there is any violation of the rules of the language in the code, then it is treated as a syntax error. The compiler detects and provides warning of such syntactical errors. The compiler generates syntax errors if it does not understand a command entered by the user. Most compilation errors are caused by mistakes in typing the code.
- **Logical errors:** Logical errors are the most serious errors in a program. These errors are related to the logic of program execution. Logical errors prevent the source program from performing what it is intended to do. It may happen that a code doesn’t work even it has no syntactical errors. Also, it may produce unexpected results. A

minor change may correct the result. These errors are primarily due to a poor understanding of the problem, incorrect translation of the algorithm, and a lack of clarity of the precedence of operators, etc.

- **Runtime errors:** Runtime errors are dynamic semantic and logical errors that are not covered by the compiler. Such errors are reported at the time of execution of the program. Errors such as mismatch of data types or referencing an out-of-range array element go undetected by the compiler. Runtime errors are commonly due to wrong input from the user. A program with runtime errors produces erroneous results. Runtime errors are usually more difficult to find and fix than syntax errors.

❖ **CODING STANDARDS**

Coding standards provide general guidelines that can be commonly adopted by programmers and the development organizations. Here, we will discuss some coding guidelines, irrespective of the programming language being used by the programmers.

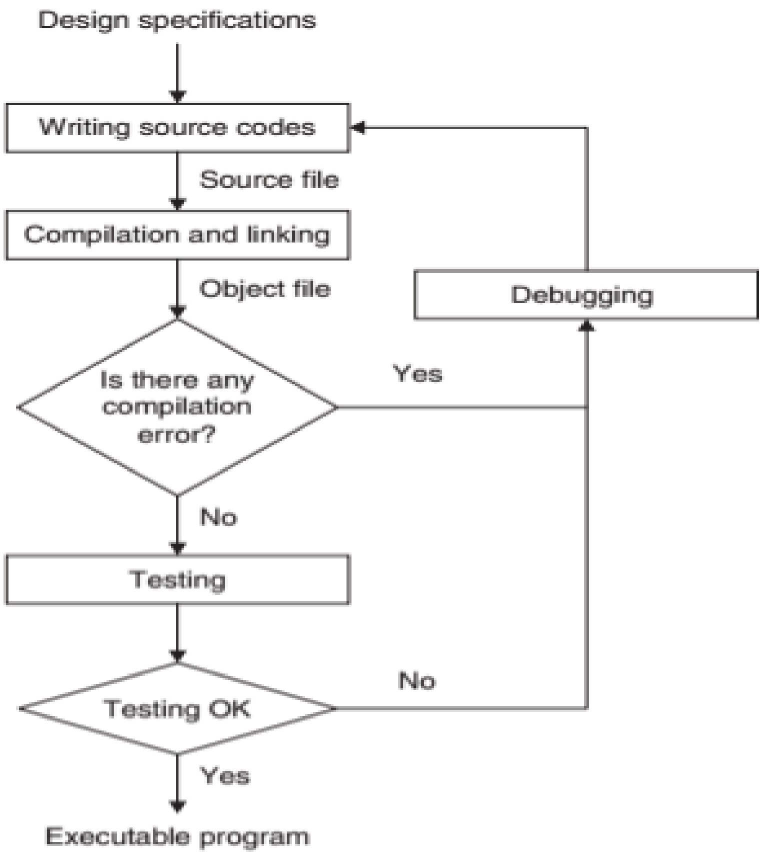
- **Naming Conventions:** There should be well-defined and meaningful names of variables, constants, programs, subprograms, files, operations, and exceptions. A meaningful name can easily be identified. It is readable and easy to understand. For example, maximum temperature can be named as maxtemp. Most of the programming languages have their own naming conventions for identifiers.
- **Code Formatting:** Code formatting is done by programmers. They should use proper indentation, references, parentheses, white spaces, line breaks, and blocks in source codes as well as in the documentation. A well-formatted code is readable and understandable. Modern programming languages provide some automated formatting of identifiers and keywords. Code formatting is to be done keeping in view the flow of a program and logical grouping of related sections of code.
- **Avoid Deep Nesting:** Too much nesting of the source code makes the code obscure. In case of nested conditions and iterations, it becomes difficult to identify which statements will be executed. Program nesting especially should not be more than one or two levels. More nested programs require much thinking to understand the code.
- **Limited Use of Global Data:** Each data item has its scope defined in the system. Before declaring a data item, its scope must be well understood. Local data are easily traceable and understandable. There are several problems with the use of global data. Global data can be used and modified by any part of the program. High use of global data makes coupling among programs stronger, which makes it too difficult to maintain the code.

CODING PROCESS

The coding process describes the steps that programmers follow for producing source codes. The coding process allows programmers to write bug-free source codes. It involves mainly coding and testing phases to generate a reliable code. The traditional coding process and test-driven development (TDD) are two widely used coding processes.

- ❖ **Traditional Coding Process-**

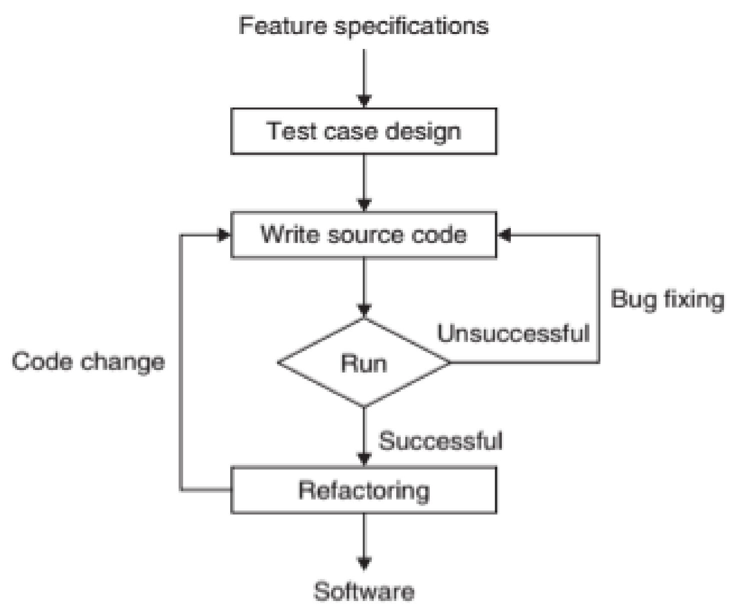
In the traditional coding process, programmers use the design specifications (algorithms, pseudo code, etc.) for writing source codes in a programming language. The source code contains executable instructions, libraries, function calls, comment lines, etc. The source files are made according to the programming language. The source files are compiled to ensure that the code is syntactically and logically correct. The compilation process proceeds once the source code is preprocessed. During compilation, source programs are translated into machine language. If there is any error in the program, it is debugged by changing the source codes. Otherwise, the same code is utilized for the testing of source codes.



Therefore, test data and test cases are designed to check that the source codes are generated as per the requirements specified in the requirements specification document.

- ❖ **Test-Driven Development-**

TDD is a disciplined programming process that helps to avoid programming errors. It is the reverse process of the traditional programming process. As the traditional development cycle is “design-code-test,” TDD has the cycle “test-code-refactor.” Here, development starts with writing test cases from the requirements rather than designing the solution. System functionality is decomposed into several small features. As the name “TDD” suggests, test cases are designed before coding.



Unit tests are written first for the feature specification and then a small source code is written according to the specification. The source code is run against the test case. It is quite possible that the small code written does not meet the requirements.

CODE VERIFICATION & REVIEW

Code verification is the process of identifying errors, failures, and faults in source codes, which cause the system to fail in performing specified tasks. Code verification ensures that functional specifications are implemented correctly using a programming language. There are several techniques in software engineering which are used for code verification.

The following methods that are widely used for code verification:

1. Code review
2. Static analysis
3. Testing

CODE REVIEW:

Code review is a traditional method for verification used in the software life cycle. It mainly aims at discovering and fixing mistakes in source codes. Code review is done after a successful compilation of source codes. Experts review codes by using their expertise in coding. The errors found during code verification are debugged.

Following methods are used for code review:

- Code walkthrough
- Code inspection
- Pair programming

Code Walkthrough-

A code walkthrough is a technical and peer review process of finding mistakes in source codes. The walkthrough team consists of a reviewee and a team of reviewers. The review team may involve a technical leader, a senior member of the project team, a person from quality assurance, a technical writer and other interested persons. The reviewee provides the review documents along with the code to be reviewed. The reviewee walks through the code or may provide the review material to the reviewers in advance. The reviewers examine the code either using a set of test cases or by changing the source code. Main focus is given on complex codes.

During the walkthrough meeting the reviewers discuss their findings to correct mistakes or improve the code. The reviewers may also suggest alternate methods for code improvement. The walkthrough session is beneficial for code verification especially when code is not properly documented. Sometimes this technique becomes time consuming and tedious. Therefore the walkthrough session is kept short.

Code Inspection -

Code Inspection is similar to code walkthrough, which aim at detecting programming defects in the source codes. The code inspection team consists of a programmer, a designer, a tester. The inspectors are provided the code and a document of checklists. The checklists focus on the important aspects in the code to be inspected. They include data referencing, memory referencing, looping conditions, conditions, conditional choices, input/output statement, comments, computational expressions, coding standards, and memory usage.

In the inspection process, definite roles are assigned to the team members, who inspect the code in a more rigorous manner. Also, the checklists help them to catch errors in a smooth manner. Code inspection takes less time as compared to code walkthrough. Most of the software companies prefer software inspection process for code review.

Pair Programming -

Pair programming is an extreme programming practice in which two programs work together at one workstation i.e. one monitor and one keyboard. In the current practice, programmers can use two keyboards. During pair programs one programmer operates the keyboard while the other is watching, learning, asking, talking and making suggestions. During pair programming, code review is done by the programmers who write the code. It is possible that they are unable to see their own mistakes. Pair programming process of code review is an implicit and a continuous process.

With the help of pair programming the pair works with better concentration. It catches simple mistakes such as ambiguous variable and method names easily. The pair shares knowledge and provides quick solution.

STATISTICAL ANALYSIS:

Static analysis is the process of automatically checking computer programs. This is performed with program analysis tools. These tools are used to detect errors in programs and they also provide information about errors. Program compilers perform static analysis to some extent. But the purpose of compilers is to ultimately generate code rather than detect defects. Static analysis tools help to identify redundancies in source codes. They identify idempotent operations, data declared but not used, dead codes, missing data, connections that lead to unreachable code segments, and redundant assignments.

TESTING:

Testing is performed before the integration of programs for system testing. Also, it is intended to ensure that the software ensures the satisfaction of customer needs. Unit testing is performed for code verification. Each module (i.e., program, function, procedure, routines, etc.) is tested using test cases. The design of a test case for testing the programs is a challenging task. Test cases are designed to observe the uncovered errors in the code. Test cases attempt to prove the code incorrect. If there is any error in the code, the programmer checks the location and the types of bugs occur in the code.

SOFTWARE DOCUMENTATION

Software development, operation, and maintenance processes include various kinds of documents. Documents act as a communication medium between the different team members of development. Documents prepared during development are problem statement, software requirement specification (SRS) document, design document, documentation in the source codes, and test document. These documents are used by the development and maintenance team members. Documents are also designed for planning, managing, and implementation of development and maintenance activities.

Thus, documentation is an important artifact in the system. There are following categories of documentation done in the system:

- Internal documentation
- System documentation
- User documentation
- Process documentation
- Daily documentation

❖ **Internal Documentation:**

Internal documentation is done in the source code. It is basically comments included in the source code, which help in understanding the source code. It mainly covers the standards prologue related to program and its compilation, comments, and self-documented lines.

❖ **System Documentation:**

System documents are supporting documents produced from requirements specification to the testing phase. These documents describe the system at different milestones. These documents are used in design, development, and even maintenance phase. System documentation is prepared to describe the technical artifacts of the system.

❖ **User Documentation:**

A software product is used by the users. Different users have different expertise and experience. The system is operated by end users and administrators. The end user uses the system to perform certain tasks and the system administrators manage the system operations performed by the various end users.

- ❖ **Daily Documentation**:

  Daily documentation helps programmers in reporting to upper levels and in preparing the phased artifacts and a plan for the next phase.

- ❖ **Process Documentation:**

  Process documentation manages the process records such as plan, schedule, process quality documents, communication documents, and standards. These are used to manage the development process. For example, the reports of various resources used are prepared during development.

TESTING INTRODUCTION

Software testing is performed once the source code has been written by software engineers. As a system is designed for the users, hence it should meet the needs and expectations of the users. Software engineers verify the source code to ensure that it meets the design specifications and it is free from errors. A quality software can be achieved through testing. Before deploying the software at the customer site, it is well tested to produce better results. Effective testing reduces the maintenance cost and provides reliable outcomes.

Testing activity takes much effort as compared to other activities of software life cycle. In testing, the behavior of software is compared with the observed and specified outcomes of the software. Software testers should be so skilled in catching the defects. Software testers always try to prove that the system is incorrect by applying test cases. To perform successful testing, testers must have a thorough understanding of the whole system and its subsystems from requirements specification to implementation.

Test Planning:

Test planning specifies the scope, approach, resources, and schedule of the testing activities. During test planning, the test team decides the features to be tested, tasks to be performed, the personnel responsible for testing, and the associated risks in the plan. Thus, the testing process follows the test plan.

Test planning includes the following activities:

- Create test plan
- Design test cases
- Design test stubs and test drivers
- Test case execution
- Defect tracking and statistics
- Prepare test summary report

A good test plan detects defects before testing begins and optimizes scarce resources. In the following subsections, we discuss the various activities of test planning.

TYPES OF TESTING TECHNIQUES

Software testing can be stated as the process of verifying and validating that a software or application is bug free, meets the technical requirements as guided by it's design and development and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.

Software Testing can be broadly classified into two types:

- Manual Testing
- Automation Testing

Software techniques can be majorly classified into two categories:

- ❖ **Black-Box Testing**
- ❖ **White-Box Testing**

BLACK BOX TESTING:

Black-box testing is performed on the basis of functions or features of the software. In black-box testing, only the input values are considered for the design of test cases. The output values that the software provides on execution of test cases are observed. The internal logic or program structures are not considered during black-box testing. Requirements specification is the basis of black-box testing. Therefore, it is also known as behavioral or functional testing

The following are different methods in black box testing techniques -

- Equivalence class partitioning
- Boundary value analysis
- Cause-effect graphing
- Error guessing

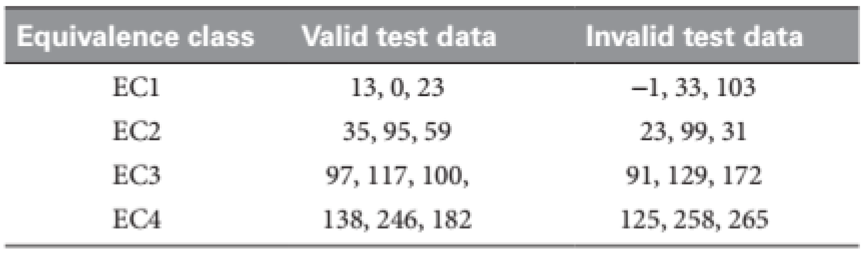
(1) Equivalence class partitioning –

Equivalence class partitioning method allows partitioning the input domain into a set of equivalence classes (i.e., sub-domains). The module under test behaves similarly for all input values of an equivalent class. Each equivalence class provides different behavior. Thus, each equivalence class is disjoint, i.e., the input values of an equivalence class will not belong to another equivalence class.

The equivalence class partitioning method has the following two aspects:

- Design of equivalence classes
- Selection of test input data

Table 9.2 Equivalence class partitioning for ASCII characters

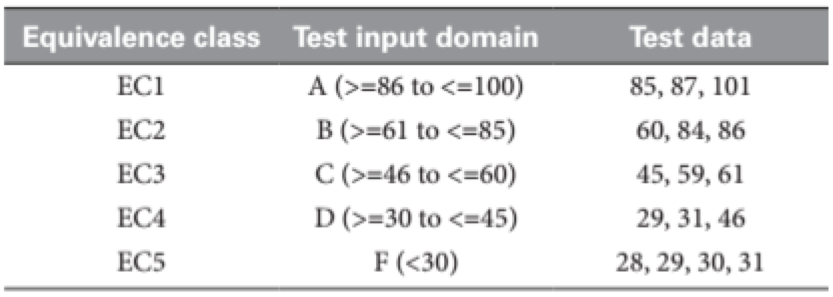


Equivalence class	Valid test data	Invalid test data
EC1	13, 0, 23	−1, 33, 103
EC2	35, 95, 59	23, 99, 31
EC3	97, 117, 100,	91, 129, 172
EC4	138, 246, 182	125, 258, 265

(2) Boundary Value Analysis -

The boundary value analysis is the special case of equivalence class partitioning method that focuses on the boundary of the equivalence classes. Here, the test input data are selected at and near the boundary of the equivalence classes whereas in equivalent classes partitioning, the test input data are selected within the equivalence class. Boundary value analysis is based on the idea of equivalence class partitioning. The input domain is partitioned into as many equivalence classes as possible. Now look at the boundary of each equivalence class. The boundary values are identified by relating the elements of the input domain. For example, relational operators such as <, <=, >, >=, ==, etc., can be applied to relate the boundary level elements.

Table 9.3 Equivalence class partitioning for ASCII characters



Equivalence class	Test input domain	Test data
EC1	A (>=86 to <=100)	85, 87, 101
EC2	B (>=61 to <=85)	60, 84, 86
EC3	C (>=46 to <=60)	45, 59, 61
EC4	D (>=30 to <=45)	29, 31, 46
EC5	F (<30)	28, 29, 30, 31

(3) Cause-Effect Graphing

Cause-effect graphing technique begins with finding the relationships among input conditions known as causes and the output conditions known as effects. A cause is any condition in the requirement that affects the program output. Similarly, an effect is the outcome of some input conditions. The logical relationships among input and output conditions are expressed in terms of a cause-effect graph.

For example, “cash withdrawal” depends upon “valid pin,” “valid amount,” and “cash availability” in the account. Using the cause-effect graphing technique, the requirements can be stated in causes and effects as follows:

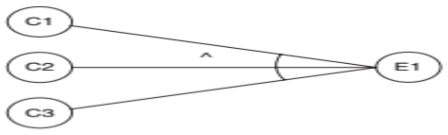
Causes: C1: Enter valid amount

C2: Enter valid pin

C3: Cash available in the account

Effects: E1: Cash withdrawal

E1 will be successful if C1, C2, and C3 are true. This can be represented in the cause-effect graph is shown below.



(4) Error Guessing

The error guessing technique is based on guessing the error-prone areas in the program. Error guessing is an intuitive and ad-hoc process of testing. The possible errors or error-prone situations are listed and then test cases are written for such errors. Software testers use their experiences and knowledge to design test cases for such error-prone situations.

Let us discuss some of such error-prone situations. Boolean variables have the values true (1) or false (0). There might be the chance of alteration of these values from 1 to 0 and vice versa.

WHITE BOX TESTING:

White-box testing is concerned with exercising the source code of a module and traversing a particular execution path. Internal logics, such as control structures, control flow, and data structures are considered during white-box testing. Unit testing is performed to test source codes. Test case designing in white-box methods also requires requirements along with source codes. The requirements are used as input data and a code is tested to check all the paths in the module at least once during testing. White-box testing is also known as glass-box testing or structural testing.

The following are different methods in white box testing techniques -

- Control-flow-based testing
- Path testing
- Data-flow-based testing
- Mutation testing

(1) Control-Flow-Based Testing

The control-flow-based testing strategy focuses on the control flows in the program. It covers those aspects of the program where control flows in the program. The goal of control-flow-based testing methods is to satisfy test adequacy criteria. Test adequacy is measured for a given test set to test a program to determine whether it satisfies the requirements.

There are various criterions used to measure test adequacy in a program.

- **Statement coverage testing** - A source program written in a programming language consists of several statements. The statements are logically grouped into program blocks. The statements can either be declarative or instructional statements. The aim of statement coverage is to design test cases so that every statement of the program can be executed at least once during testing. Similarly, block coverage concentrates on covering all the blocks in the program.

• **Branch coverage testing** - Branch coverage testing is also known as decision coverage. In this coverage criterion, test cases are designed in such a way that all the outcomes of the decision have been considered. That is, each branch or decision in the program is evaluated as true or false at least once during testing. In this context, if and while statements are evaluated to true in some cases and false in some other cases.

• Condition coverage testing - Simple conditions such as (a < 0) are covered using branch coverage criteria. But there can also exist complex conditions which are made using logical operations, such as AND, OR, and XOR. In addition, negation operator NOT (~) is used to negate the outcome of a condition. Simple conditions are easy to test because these are evaluated to true or false in a straightforward manner.

Example:

```
int a, b;
if (a>=0 && b>0)
     a=a+b;
else
    a=a-b;
```

(2) Path Testing

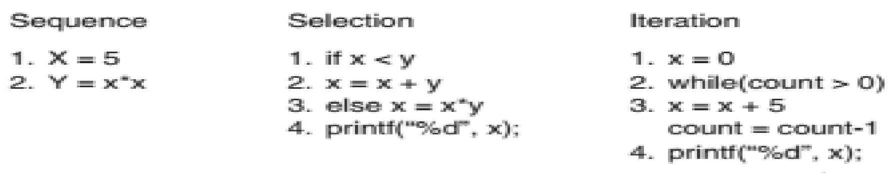
Path testing is another white-box testing method, which focuses on identifying independent paths in a program. Sometimes processing of code is missed by the programmers that may lead to unidentified paths. It is practically difficult to test all orders of processing as well as all program paths. The idea is to design test cases to exercise all independent paths at least once during testing.

The following are the important strategies in Path testing –

- **Control Flow Graph** –

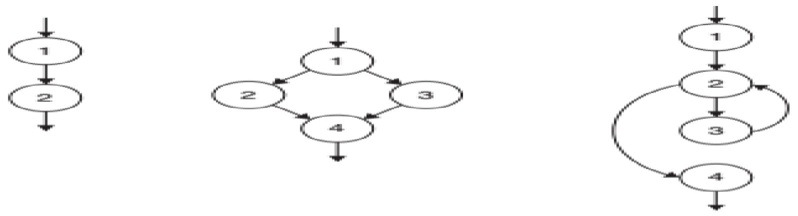
A control flow graph (CFG) is also known as flow graph or program graph. It describes the flow of control within the program. It is a finite set of nodes and a finite set of directed edges. The nodes represent the executable blocks or the lines in the program and the edges represent flow of controls. There is a start and an end node in the graph. The start node has no incoming edges and the end node has no outgoing edges. Every node in the graph is reachable from the start node and terminates at the end node. A basic block is represented by a rectangle or small circles in case of line numbers as node in the program.

Example:



```
Sequence          Selection                 Iteration

1. X = 5          1. if x < y               1. x = 0
2. Y = x*x        2. x = x + y              2. while(count > 0)
                  3. else x = x*y           3. x = x + 5
                  4. printf("%d", x);          count = count-1
                                            4. printf("%d", x);
```



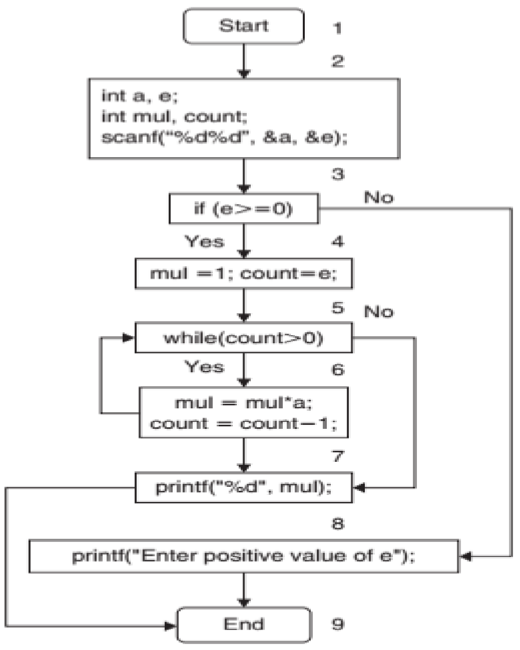
- **Independent Path** –

A path through a program is a sequence of nodes and edges from start to end node in the flow graph. An independent path is any path through the program that introduces at least one new edge that is not included in any other path before it.

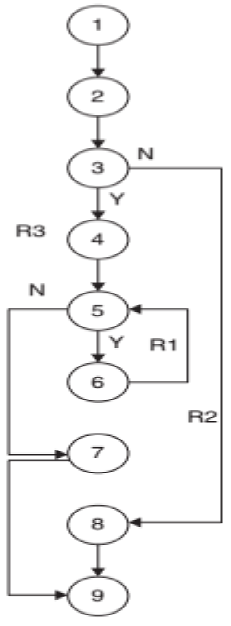
Example:

```
main()
{
      int a, e;
      int mul, count;
      scanf("%d%d", &a, &e);
      if (e>=0) {
              mul=1; count=e;
              while(count>0) {
                      mul=mul*a;
                      count =count-1;
                      }
              printf("%d", mul);
              }
      else
              printf("Enter positive value of e");
}
```

P1: $1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9$
P2: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 9$
P3: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 9$



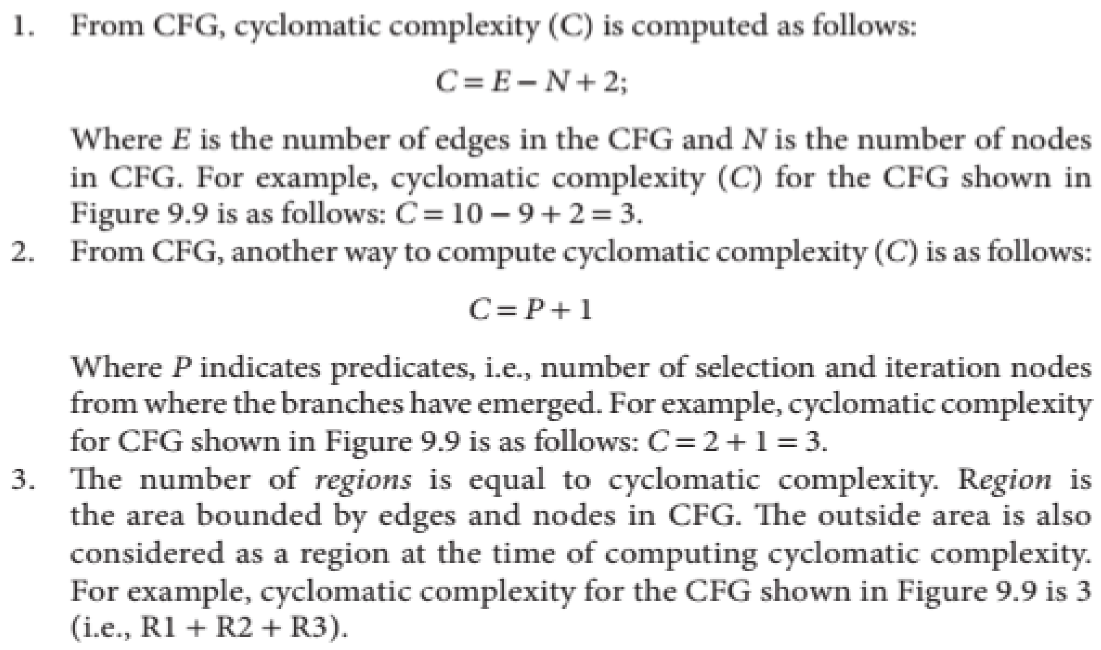
(a)



(b)

- **Cyclomatic Complexity Measure** –

Cyclomatic complexity is the measurement of logical complexity in the program. Cyclomatic complexity is a number, which is computed through certain analytical formulas. The cyclomatic complexity number defines the required number of independent paths in a given CFG.



1. From CFG, cyclomatic complexity (C) is computed as follows:

   $$C = E - N + 2;$$

   Where E is the number of edges in the CFG and N is the number of nodes in CFG. For example, cyclomatic complexity (C) for the CFG shown in Figure 9.9 is as follows: $C = 10 - 9 + 2 = 3$.
2. From CFG, another way to compute cyclomatic complexity (C) is as follows:

   $$C = P + 1$$

   Where P indicates predicates, i.e., number of selection and iteration nodes from where the branches have emerged. For example, cyclomatic complexity for CFG shown in Figure 9.9 is as follows: $C = 2 + 1 = 3$.
3. The number of *regions* is equal to cyclomatic complexity. *Region* is the area bounded by edges and nodes in CFG. The outside area is also considered as a region at the time of computing cyclomatic complexity. For example, cyclomatic complexity for the CFG shown in Figure 9.9 is 3 (i.e., R1 + R2 + R3).

- **Design of Test Cases** -

The purpose of constructing CFG and finding cyclomatic complexity is to design test cases that can execute every statement of the program at least once. The process of path testing to design test cases from CFG is as follows:

Step 1: Construct CFG.

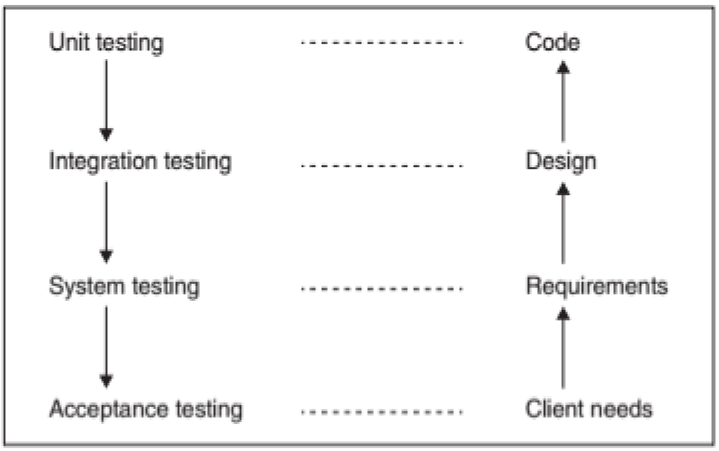
Step 2: Compute Cyclomatic complexity.

Step 3: Identify independent paths.

Step 4: Prepare test cases.

LEVELS OF TESTING/ TESTING METHODOLOGIES

Testing is a defect detection technique that is performed at various levels. Testing begins once a module is fully constructed. Although software engineers test source codes after it is written, but this is not an appealing way that can satisfy customer's needs and expectations. Software is developed through a series of activities, viz., customer needs, specification, design, and coding. Each of these activities has different aims. Therefore, testing is performed at various levels of development phases to achieve their purposes.



1) UNIT TESTING

Unit testing is the starting level of testing. Here, unit means a program unit, module, component, procedure, subroutine of a system developed by the programmer. The aim of unit testing is to find bugs by isolating an individual module using test stub and test drivers and executing test cases on it. The program units are tested before integration testing. Unit testing is performed to detect both structural and functional errors in the module. Therefore, test cases are designed using white-box and black-box testing strategies for unit testing. Most of the module errors are captured through white-box testing.

2) INTEGRATION TESTING

Integration testing is another level of testing, which is performed after unit testing of modules. It is carried out keeping in view the design issues of the system, which may consist of several subsystems. Two or more modules are integrated into subsystems and testing is performed in an integrated manner. Upon integration, the subsystems may not work properly or may not be providing the subsystems objections. The main goal of integration testing is to find interface errors between modules. Interface errors are parameter errors, are parameter errors, ordering of module under integration, and so on. The modules are together rather than stub and driver modules.

There are various approaches in which the modules are combined together for integration testing.

These are as follows;

- **Big-Bang Approach**

Big- bang is a simple and straightforward integration testing approach in this approach, all the modules are first tested individually and then these are combined together and tested as a single system. This approach works well where there is less number of modules in a system.

- **Top-Down Approach**

As we know, software design is made by approaching modules in a hierarchical order. Top-down integration testing begins with the main module and move downwards integrating and testing its lower-level modules. Again the next lower-level modules are integrated and tested. Thus, this incremental integration and testing is continued until all modules up to the concrete level have been integrated and tested.

The top down integration testing approach is as follows main system-> subsystems-> at concrete level.

- **Bottom-Up Approach**

As the name implies, the bottom-up approach begins with individual testing of the bottom-level modules in the software hierarchy. Then the lower-level modules are merged function-wise together to form a subsystem and then all subsystems are integrated to test the main module covering all modules of the system. The approach of bottom-up integration is as follows: concrete level modules-> subsystem-> main module.

- **Sandwich Approach**

Sandwich testing combines both top-down and bottom-up integration approaches. During sandwich testing, the top-down approach forces the lower-level modules to be available and the bottom-up approach requires upper-level modules. Thus, testing a module requires its top- and bottom-level modules.

3) SYSTEM TESTING

Unit and integration testing are applied to defects in the modules and the system as a whole. Once all the modules have been tested, system testing is performed to check whether the system satisfies the requirements (both functional and nonfunctional). To test the functional requirements of the system, functional or black-box testing methods, are used with appropriate test cases.

- **Performance Testing**

Performance testing is carried out to check the runtime outcomes of the system, such as efficiency, accuracy etc. Each system performs differently in different environment.

- **Volume Testing**

It deals with the system if heavy amount of date are to be processed or stored in the system.

- **Stress Testing**

In stress testing behavior of the system is checked when it is under stress. Stress may come due to load increases at park time for a short period of time.

- **Security Testing**

Security testing is conducted to ensure security checks at different levels in the system.

- **Recovery Testing**

Most of the systems now have recovery policies if there is any loss of data. Therefore, recovery testing is performed to check whether the system will recover the losses caused by data errors, software error.

- **Configuration Testing**

Configuration testing in performed to check if a system can run on different hardware and software configurations. Therefore, system is configured for each of the hardware and software.

- **Documentation Testing**

Once the system becomes operational, problems may be encountered in the system. A systematic documentation or manual can help to solve such problems. The system is checked to see whether its proper documentation is available.

4) ACCEPTANCE TESTING

Acceptance testing is a kind of system testing which is performed before the system is released into the market. It is performed with the customer to ensure that the system is acceptable for delivery. Once the entire system testing has been exercised, the system is tested from the customer's point of view. Acceptance testing is conduct because there is a difference between the actual user and the simulated user as considered by the development organization.

- **Alpha Testing**

Alpha testing is pilot testing in which customers are involved in exercising test cases. In alpha testing, the customer conducts tests in the development environment. The users perform alpha test and try to pinpoint any problem in the system. The alpha test is conducted in a controlled environment. After alpha testing-system is ready to be transported to the customer site for development.

- **Beta Testing**

Beta testing is performed by limited and friendly customers and end users. Beta testing is conducted at the customer site where the software is to be developed and used by the end users. The developer may or may not be present during beta testing. The end users operate the system under testing mode and note down any problem observed during system operation. The defects noted by the end users are corrected by the developer. If there are any major changes required, then these changes are sent to the configuration management team.

DEBUGGING

Debugging is a post testing mechanism of locating and fixing errors. A successful test case aims to prove that the program is incorrect. The behavior of a program is observed by symptoms or known errors. The system may vary from an error to system failures. Once errors are reported by testing methods, these are isolated and removed during debugging.

Debugging has two important steps:

-Identifying the location and nature of errors

- Correcting or fixing errors

The most popular debugging approaches are as follows:

- **Brute Force**

It is the simplest method of debugging but it is inefficient. It uses memory dumps or output statements for debugging. A memory dump is a machine level representation of the corresponding variables and statements. It represents the static structure of the program at a particular snapshot of execution sequence. The memory dump rarely establishes correspondence to show errors at a particular time.

- **Backtracking**

Backtracking is the refinement of the brute force method and it is one of the successful methods of debugging. In this method, debugging from where the bug is discovered and the source code is traced out backward though different paths until the exact location of the cause of bug has disappeared. This process is performed with the program logic in reverse direction of the flow of control. This method is effective for small size problems. Backtracking should be used when all other methods of debugging are not able to locate error.

- **Breakpoint**

Breakpoint debugging is a method of tracing programs with a breakpoint and stopping the programs execution at the breakpoint. A breakpoint is a kind of signal that tells the debugger to temporarily suspend execution of program at a certain point . Each breakpoint is associated with a particular instruction of the program. Program execution continues before the breakpoint statement. If any error is reported, its location is marked and then the program execution resumes till the next breakpoint. This process is continued until all errors are located in the program.

- **Debugging By Induction**

It is based on pattern matching and a thought process on some clue. The process begins with collecting information about pertinent data where the hug has been discovered. The patterns of successful test cases are observed and data items are organized. Thereafter hypothesis is derived by relating the pattern and the error to be debugged.

- **Debugging By Deduction**

This is a kind of cause elimination method. On the basis of cause hypothesis, lists of possible causes are enumerated for the observed failure. Now tests are conducted to eliminate causes to remove errors in the system. If all the causes are eliminated, then errors are fixed.

- **Debugging By Testing**

It uses test cases to locate. Test cases designed during testing are used in debugging to collect information to locate the suspected errors. The test case in testing focuses on covering many conditions and statements, whereas the test case in debugging focuses on a small number of conditions and statements. Test cases of debugging are refined cases of testing. Test cases during debugging concentrate on locating error situations in a program.

PROGRAM ANALYSIS TOOL

Program analysis tools are automated tools which provide additional information in addition to the output produced by the translators. These tools take source code or executable code as input and produce required information. These tools are useful to find program size, complexity, independency of modules, time taken in parts of the program, number of calls, adequacy of documentation, adequacy of testing etc.

Program analysis tools are classified into the following 2 categories:

- Static Analysis tools
- Dynamic analysis tools

Static Analysis Tools

Static program analysis tools look at the source code without executing it and detect possible defects before running the program. These tools compute lines of code, comment frequency, proper nesting, number of function calls, cyclomatic complexity etc. Static analysis is performed after coding and before executing unit tests.

Code walkthrough and code review techniques are considered as static program analysis techniques. Code walkthrough is an automatic analysis tool that examines source code and detects non-complying rules and standards. A compiler is an example of code walkthrough which finds lexical, syntactical and some semantical mistakes. Code review is done by humans to ensure proper coding standards and rules are adhered to while writing the program .Static analysis tools focus on reliability, reusability, efficiency, testability, portability and maintainability quality attributes.

The main advantage of static analysis is that it finds issues with the code before it is ready for integration and further testing. It finds weaknesses earlier in the development life cycle reducing the cost to fix .Also it detects problems in code at the exact location. However static code analysis tools are time consuming.

Dynamic Analysis Tools

Dynamic analysis tools look at the flow of execution of the program. It analyses the dynamic behavior of the program at run time. It requires the loading of libraries and recompilation of source code. Unit testing is the most common practice of dynamic analysis, which target to find errors in source code. Dynamic analysis is performed to point out the profiling aspects, pointer or memory error, data structure abuses etc.
Dynamic code analysis identifies vulnerabilities in a runtime environment. It can be conducted against any application. It allows for analysis of applications in which you do not have access to the actual code. It identifies false negative vulnerabilities. However dynamic code analysis tools provide a false sense of security that everything is being addressed. Sometimes, these are unable to ensure all test coverage. These tools produce false positives and false negatives. It is difficult to trace the problems at the exact location in the code and takes longer to fix it.

TESTING OBJECT-ORIENTED PROGRAMS

Object oriented programming has several benefits for developing real world problems. But it has to deal with new problems introduced by objects oriented features like

- Encapsulation
- Inheritance
- Polymorphism
- Message Passing
- Persistence
- Dynamic Binding

Object oriented programming supports reuse, suitable for development of large scale software, reliability, interoperability and extendibility. Object Oriented Testing can be performed different levels like

- **Unit Testing –** It is performed at individual classes. It looks after class attributes are implemented as per design models and methods with error-free.
- **Subsystem Testing –** It is performed at algorithmic level. Where each module f every class is tested in isolation. It is done by subsystem lead and association within subsystem to interact with outside the system.
- **System Testing -** It is job of quality assurance team for testing system as a whole. This team often uses system tests as regression test when assembling new releases.

Object Oriented Testing Techniques –

The Object oriented test case design is based upon certain model based testing known as Gray Box Testing ie., A combination of Black-box and White-box testing techniques.77

SOME GENERAL ISSUES ASSOCIATED WITH TESTING

Here we discuss about the Usability Testing, Regression Testing and Smoke Testing and its issues with Software Testing

USABILITY TESTING

Usability refers to the ease of use and comfort that users have while working with software. It is also known as user centric testing. Nowadays usability has become a wider aspect of software development and testing. Usability testing is conducted to check usability of the system, which mainly focuses on finding the differences between quality of developed software and users expectations of what it should perform. Poor usability may affect the access of the software. If the users find that the system is difficult to understand and operate then ultimately it will lead to an unsuccessful product.

There are three types of usability tests which are *scenario test*, *prototype test* and *product test*. In scenario test end users are presented with a visionary scenario of the software. The end users go through the dialogues of the scenario and developer observes how the end users interact and reacts with the system. The developers get immediate feedback by the scenario test. In a prototype test end users are provided a piece of feedback by the scenario test. In a prototype test end users provide a piece of software that implements important aspects of the system .The end user provides a realistic view of the prototype. The collected suggestions are incorporated into the system. The product test is just like the prototype test except that the functional version of the software is used for usability testing instead of prototype version.

REGRESSION TESTING

Regression testing is also known as program revalidation. Regression testing is performed whenever new functionality is added or an existing functionality is modified in the program. If the existing system is working correctly then new system should work correctly after making changes because the code may have been changed. This is required when the new version of the program is obtained by changing the existing version. Regression testing is also needed when a subsystem is modified to get the new version of the system.

There are various techniques of regression testing such as test –all, test minimization, test prioritization and random selection. In test all technique test cases for regression testing are selected from the test cases designed for the existing system .Here all test cases are executed for regression testing of the new system. This technique becomes very tedious if the new system is to be tested with added functionalities especially if it is to be tested at last. In such cases small regression tests can be used for regression testing. Therefore regression testing is supported by automated testing tools. Sometimes it is not necessary to cover all tests for the modified program. Test minimization aims to reduce the number of tests and the selected tests. It is mainly based upon the code coverage concept.

SMOKE TESTING

Smoke testing is also sometimes known as sanity testing. In smoke testing software module is tested to verify “build” activity in an informal and non exhaustive manner. It is done to check that major functionalities of the system are working properly before performing detailed black box and white box testing. Smoke test should be able to expose errors in the system. The concept of smoke testing is taken from hardware devices, in which a new hardware device was attached to the system first time and it was assumed to be successful if it did not start producing smoke. Sanity testing is also performed with build modules along with minor changes in the code .It ascertains the bugs have been fixed and no further issues are introduced to these changes .

Smoke testing is performed on various build modules when it is incorporated into the system. Each build module when it is incorporated into the system. Such build modules can be programs constructed by programmers, reusable components, library modules, data files and so on. The system is tested on a regular basis if any build module is included in the system.