**UNIT – 4 Software Implementation**

this chapter, we will study about programming methods, documentation and challenges in software implementation.



Structured Programming

In the process of coding, the lines of code keep multiplying, thus, size of the software increases. Gradually, it becomes next to impossible to remember the flow of program. If one forgets how software and its underlying programs, files, procedures are constructed, it then becomes very difficult to share, debug, and modify the program. The solution to this is structured programming. It encourages the developer to use subroutines and loops instead of using simple jumps in the code, thereby bringing clarity in the code and improving its efficiency Structured programming also helps programmer to reduce coding time and organize code properly.

Structured programming states how the program shall be coded. It uses three main concepts:

1. **Top-down analysis** - A software is always made to perform some rational work. This rational work is known as problem in the software parlance. Thus it is very important that we understand how to solve the problem. Under top-down analysis, the problem is broken down into small pieces where each one has some significance. Each problem is individually solved and steps are clearly stated about how to solve the problem.
2. **Modular Programming** - While programming, the code is broken down into smaller group of instructions. These groups are known as modules, subprograms, or subroutines. Modular programming based on the understanding of top-down analysis. It discourages jumps using ‘goto’ statements in the program, which often makes the program flow non- traceable. Jumps are prohibited and modular format is encouraged in structured programming.
3. **Structured Coding** - In reference with top-down analysis, structured coding sub-divides the modules into further smaller units of code in the

order of their execution. Structured programming uses control structure, which controls the flow of the program, whereas structured coding uses control structure to organize its instructions in definable patterns.



Functional Programming

Functional programming is style of programming language, which uses the concepts of mathematical functions. A function in mathematics should always produce the same result on receiving the same argument. In procedural languages, the flow of the program runs through procedures, i.e. the control of program is transferred to the called procedure. While control flow is transferring from one procedure to another, the program changes its state.

In procedural programming, it is possible for a procedure to produce different results when it is called with the same argument, as the program itself can be in different state while calling it. This is a property as well as a drawback of procedural programming, in which the sequence or timing of the procedure execution becomes important.

Functional programming provides means of computation as mathematical functions, which produces results irrespective of program state. This makes it possible to predict the behavior of the program.

Functional programming uses the following concepts:

**First class and High-order functions** - These functions have capability to accept another function as argument or they return other functions as results.

* **Pure functions** - These functions do not include destructive updates, that is, they do not affect any I/O or memory and if they are not in use, they can easily be removed without hampering the rest of the program.
* **Recursion** - Recursion is a programming technique where a function calls itself and repeats the program code in it unless some pre-defined condition matches. Recursion is the way of creating loops in functional programming.
* **Strict evaluation** - It is a method of evaluating the expression passed to a function as an argument. Functional programming has two types of evaluation methods, strict (eager) or non-strict (lazy). Strict evaluation always evaluates the expression before invoking the function. Non-strict evaluation does not evaluate the expression unless it is needed.
* **λ-calculus** - Most functional programming languages use λ-calculus as their type systems. λ-expressions are executed by evaluating them as they occur.

Common Lisp, Scala, Haskell, Erlang, and F# are some examples of functional programming languages.



Programming style

Programming style is set of coding rules followed by all the programmers to write the code. When multiple programmers work on the same software project, they frequently need to work with the program code written by some other developer. This becomes tedious or at times impossible, if all developers do not follow some standard programming style to code the program.

An appropriate programming style includes using function and variable names relevant to the intended task, using well-placed indentation, commenting code for the convenience of reader and overall presentation of code. This makes the program code readable and understandable by all, which in turn makes debugging and error solving easier. Also, proper coding style helps ease the documentation and updation.

## Coding Guidelines

Practice of coding style varies with organizations, operating systems and language of coding itself.

The following coding elements may be defined under coding guidelines of an organization:

* **Naming conventions** - This section defines how to name functions, variables, constants and global variables.
* **Indenting** - This is the space left at the beginning of line, usually 2-8 whitespace or single tab.
* **Whitespace** - It is generally omitted at the end of line.
* **Operators** - Defines the rules of writing mathematical, assignment and logical operators. For example, assignment operator ‘=’ should have space before and after it, as in “x = 2”.
* **Control Structures** - The rules of writing if-then-else, case-switch, while- until and for control flow statements solely and in nested fashion.
* **Line length and wrapping** - Defines how many characters should be there in one line, mostly a line is 80 characters long. Wrapping defines how a line should be wrapped, if is too long.
* **Functions** - This defines how functions should be declared and invoked, with and without parameters.
* **Variables** - This mentions how variables of different data types are declared and defined.
* **Comments** - This is one of the important coding components, as the comments included in the code describe what the code actually does and all other associated descriptions. This section also helps creating help documentations for other developers.



Software Documentation

Software documentation is an important part of software process. A well written document provides a great tool and means of information repository necessary to know about software process. Software documentation also provides information about how to use the product.

A well-maintained documentation should involve the following documents:

* **Requirement documentation** - This documentation works as key tool for software designer, developer, and the test team to carry out their respective tasks. This document contains all the functional, non-functional and behavioral description of the intended software.

Source of this document can be previously stored data about the software, already running software at the client’s end, client’s interview, questionnaires, and research. Generally it is stored in the form of spreadsheet or word processing document with the high-end software management team.

This documentation works as foundation for the software to be developed and is majorly used in verification and validation phases. Most test-cases are built directly from requirement documentation.

* **Software Design documentation** - These documentations contain all the necessary information, which are needed to build the software. It contains: **(a)** High-level software architecture, **(b)** Software design details, **(c)** Data flow diagrams, **(d)** Database design

These documents work as repository for developers to implement the software. Though these documents do not give any details on how to code the program, they give all necessary information that is required for coding and implementation.

* **Technical documentation** - These documentations are maintained by the developers and actual coders. These documents, as a whole, represent information about the code. While writing the code, the programmers also mention objective of the code, who wrote it, where will it be required, what it does and how it does, what other resources the code uses, etc.

The technical documentation increases the understanding between various programmers working on the same code. It enhances re-use capability of the code. It makes debugging easy and traceable.

There are various automated tools available and some comes with the programming language itself. For example java comes JavaDoc tool to generate technical documentation of code.

* **User documentation** - This documentation is different from all the above explained. All previous documentations are maintained to provide information about the software and its development process. But user documentation explains how the software product should work and how it should be used to get the desired results.

These documentations may include, software installation procedures, how- to guides, user-guides, uninstallation method and special references to get more information like license updation etc.



Software Implementation Challenges

There are some challenges faced by the development team while implementing the software. Some of them are mentioned below:

* **Code-reuse** - Programming interfaces of present-day languages are very sophisticated and are equipped huge library functions. Still, to bring the cost down of end product, the organization management prefers to re-use the code, which was created earlier for some other software. There are huge issues faced by programmers for compatibility checks and deciding how much code to re-use.
* **Version Management** - Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation. This documentation needs to be highly accurate and available on time.
* **Target-Host** - The software program, which is being developed in the organization, needs to be designed for host machines at the customers end.

But at times, it is impossible to design a software that works on the target machines.

**Software Testing**

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.



Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

* Validation ensures the product under development is as per the user requirements.
* Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
* Validation emphasizes on user requirements.



Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

* Verification ensures the product being developed is according to design specifications.
* Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
* Verifications concentrates on the design and system specifications. Target of the test are -
* **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
* **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
* **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.



Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

* **Manual** - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

* **Automated** This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.



Testing Approaches

Tests can be conducted based on two approaches –

1. Functionality testing
2. Implementation testing

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testing where not only functionality is tested but the way it is implemented is also analyzed.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not

possible to test each and every value in real world scenario if the range of values is large.

## Black-box testing

It is carried out to test functionality of the program and also called ‘Behavioral’ testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested ‘ok’, and problematic otherwise.

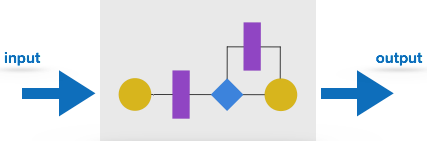
In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

* **Equivalence class** - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
* **Boundary values** - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
* **Cause-effect graphing** - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.
* **Pair-wise Testing** - The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair- wise for their different values.
* **State-based testing** - The system changes state on provision of input. These systems are tested based on their states and input.

## White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as ‘Structural’ testing.



In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

* **Control-flow testing** - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.
* **Data-flow testing** - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.



Testing Levels

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.

Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

## Unit Testing

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

## Integration Testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

## System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

* **Functionality testing** - Tests all functionalities of the software against the requirement.
* **Performance testing** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.
* **Security & Portability** - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

## Acceptance Testing

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

* **Alpha testing** - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.
* **Beta testing** - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

## Regression Testing

Whenever a software product is updated with new code, feature or functionality, it is tested thoroughly to detect if there is any negative impact of the added code. This is known as regression testing.



Testing Documentation

Testing documents are prepared at different stages -

## Before Testing

Testing starts with test cases generation. Following documents are needed for reference –

* **SRS document** - Functional Requirements document
* **Test Policy document** - This describes how far testing should take place before releasing the product.
* **Test Strategy document** - This mentions detail aspects of test team, responsibility matrix and rights/responsibility of test manager and test engineer.
* **Traceability Matrix document** - This is SDLC document, which is related to requirement gathering process. As new requirements come, they are added to this matrix. These matrices help testers know the source of requirement. They can be traced forward and backward.

## While Being Tested

The following documents may be required while testing is started and is being done:

* **Test Case document** - This document contains list of tests required to be conducted. It includes Unit test plan, Integration test plan, System test plan and Acceptance test plan.
* **Test description** - This document is a detailed description of all test cases and procedures to execute them.
* **Test case report** - This document contains test case report as a result of the test.
* **Test logs** - This document contains test logs for every test case report.

## After Testing

The following documents may be generated after testing :

* **Test summary** - This test summary is collective analysis of all test reports and logs. It summarizes and concludes if the software is ready to be launched. The software is released under version control system if it is ready to launch.



Testing vs. Quality Control & Assurance and Audit

We need to understand that software testing is different from software quality assurance, software quality control and software auditing.

* **Software quality assurance** - These are software development process monitoring means, by which it is assured that all the measures are taken as per the standards of organization. This monitoring is done to make sure that proper software development methods were followed.
* **Software quality control** - This is a system to maintain the quality of software product. It may include functional and non-functional aspects of software product, which enhance the goodwill of the organization. This system makes sure that the customer is receiving quality product for their requirement and the product certified as ‘fit for use’.

**Software audit** - This is a review of procedure used by the organization to develop the software. A team of auditors, independent of development team examines the software process, procedure, requirements and other aspects of SDLC. The purpose of software audit is to check that software and its development process, both conform standards, rules and regulations