7 SOFTWARE IMPLEMENTATIONS

Software Implementation

Structured Programming

In the process of coding, the lines of code keep multiplying, thus, the size of the software increases. Gradually, it becomes next to impossible to remember the flow of a program. If one forgets how software and its underlying programs, files, and 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 to the code and improving its efficiency Structured programming also helps programmers to reduce coding time and organize code properly.

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

- **Top-down analysis** - Software is always made to perform some rational work. This rational work is known as a problem in the software parlance. Thus we must 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.
- **Modular Programming** - While programming, the code is broken down into a smaller group of instructions. These groups are known as modules, subprograms, or subroutines. Modular programming is based on the understanding of top-down analysis. It discourages jumps using 'goto' statements in the program, which often makes the program flow nontraceable. Jumps are prohibited and modular format is encouraged in structured programming.
- **Structured Coding** - About 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 a 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 the 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, a procedure can produce different results when it is called with the same argument, as the program itself can be in a 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 produce 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 can accept another function as an argument or 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) and non-strict (lazy). Strict evaluation always evaluates the expression before invoking the function. Nonstrictevaluation 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 a 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 the 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 documentation and updation.

Coding Guidelines
The practice of coding style varies with organizations, operating systems, and the language of coding itself.

The following coding elements may be defined under the 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 a line, usually 2-8 whitespace or a single tab.
- Whitespace - It is generally omitted at the end of the line.
- Operators - Defines the rules of writing mathematical, assignment, and logical operators. For example, the 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 does and all other associated descriptions. This section also helps create help documentation for other developers.