**Chapter - 1**

# **INTRODUCTION TO SOFTWARE ENGINEERING**

## Introduction

Software is a program or set of programs containing instructions which provide desired functionality. And Engineering is the processes of designing and building something that serves a particular purpose and find a cost effective solution to problems.

**Software Engineering** is a systematic approach to the design, development, operation, and maintenance of a software system.

**Objectives of Software Engineering:**

1. **Maintainability –**
It should be feasible for the software to evolve to meet changing requirements.
2. **Correctness –**
A software product is correct, if the different requirements as specified in the SRS document have been correctly implemented.
3. **Reusability –**
A software product has good reusability, if the different modules of the product can easily be reused to develop new products.
4. **Testability –**
Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.
5. **Reliability –**
It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
6. **Portability –**
In this case, software can be transferred from one computer system or environment to another.
7. **Adaptability –**
In this case, software allows differing system constraints and user needs to be satisfied by making changes to the software.

## Concept of System

**General System Theory:** It says that we have to consider each and every part of system rather than just focus on only one component of the system. Many times, the analyst tends to focus on one component only and overlook the others, the **General System Theory tends on developing a systematic and complete structure which is further helpful in making decisions,**

**SYSTEM DEFINITION: A System can be said as a collection of components that works together to achieve some common goal.**

Input -> Processing -> Output

Types of System:

### Open and Closed Systems

An **open system** is one that interacts with its environment and thus exchanges information, material, or energy with the environment, including random and undefined inputs. Open systems are adaptive in nature as they tend to react with the environment in such a way organizing', in the sense that they change their continued existence.

Such systems are ‘self-organizing’, because they change their organization in response to changing conditions. A **closed system** is one, which doesn’t interact with its environment. Such systems, in business world, are rare. Thus the systems that are relatively isolated from the environment but not completely closed are termed **closed systems**.

### Static and Dynamic Systems

**Dynamic Systems**

A **dynamic system** is composed of many interdependent subsystems. If a subsystem fails, this fact can affect the dynamics of a whole galaxy of other subsystems and the survival of the main system.

A dynamic system can become static or unstable and destructive. Every system has a finite amount of energy with which to survive, therefore instability cannot last indefinitely.

**Static Systems**

These are the models that gives us a picture about some relationship.

## Programs v/s Software Products

**Program**

1. Program is a set of instructions written in a programming language used to execute for a specific task or particular function.
2. A program does not have further categorization.
3. A program cannot be software.
4. A program consists of a set of instructions which are coded in a programming language like c, C++, PHP, Java etc.
5. Programs do not have a user interface.
6. A program is developed and used by either a single programmer or a group of programmers.
7. A program is compiled every time when we need to generate some output from it.
8. Program has limited functionality and less features.
9. Program functionality is dependent on compiler.
10. A program takes less time to build/make.
11. Program development approach is un-procedural, un-organized and unplanned.
12. The size of a program ranges from kilobytes (Kb) to megabytes (Mb).
13. Examples of computer programs include: Operating system, office suite, video games, malware, a web browser like Mozilla Firefox and Apple Safari.

**Software Products**

1. Software is a collection of several programs and other procedures and documentation.
2. Software can be categorized into two categories: application software and system software.
3. Software can be a program.
4. Software consists of bundles of programs and data files. Programs in specific software use these data files to perform a dedicated type of tasks.
5. Every software has a dedicated user interface. The user interface of software may be in the form of a command prompt or in a graphical format.
6. Software is developed by either a single programmer or a group of programmers.
7. Whole software is compiled, tested and debugged during the development process.
8. Software has lots of functionality and features such as GUI, input/output data, process etc.
9. Software functionality is dependent on the operating system.
10. Software takes relatively more time to build/make when compared to program.
11. Software development approach is systematic, organized and very well planned.
12. The size of a software ranges from megabytes (Mb) to Gigabytes (Gb).
13. Examples of software include:  Microsoft Word, Microsoft Excel, VLC media player, Firefox, Adobe Reader, Windows, Linux, Unix, Mac etc.

## Emergence of Software Engineering

Software engineering discipline is the result of advancement in the field of technology. In this section, we will discuss various innovations and technologies that led to the emergence of software engineering discipline.

### Early Computer Programming

As we know that in the early 1950s, computers were slow and expensive. Though the programs at that time were very small in size, these computers took considerable time to process them. They relied on assembly language which was specific to computer architecture. Thus, developing a program required lot of effort. Every programmer used his own style to develop the programs.

### High Level Language Programming

With the introduction of semiconductor technology, the computers became smaller, faster, cheaper, and reliable than their predecessors. One of the major developments includes the progress from assembly language to high-level languages. Early high level programming languages such as COBOL and FORTRAN came into existence. As a result, the programming became easier and thus, increased the productivity of the programmers. However, still the programs were limited in size and the programmers developed programs using their own style and experience.

### Control Flow Based Design

With the advent of powerful machines and high level languages, the usage of computers grew rapidly: In addition, the nature of programs also changed from simple to complex. The increased size and the complexity could not be managed by individual style. It was analyzed that clarity of control flow (the sequence in which the program's instructions are executed) is of great importance. To help the programmer to design programs having good control flow structure, **flowchartingtechnique** was developed. In flowcharting technique, the algorithm is represented using flowcharts. A **flowchart** is a graphical representation that depicts the sequence of operations to be carried out to solve a given problem.

Note that having more GOTO constructs in the flowchart makes the control flow messy, which makes it difficult to understand and debug. In order to provide clarity of control flow, the use of GOTO constructs in flowcharts should be avoided and **structured constructs-decision,** sequence, and loop-should be used to develop **structured flowcharts.** The decision structures are used for conditional execution of statements (for example, if statement). The sequence structures are used for the sequentially executed statements. The loop structures are used for performing some repetitive tasks in the program. The use of structured constructs formed the basis of the **structured programming** methodology.

Structured programming became a powerful tool that allowed programmers to write moderately complex programs easily. It forces a logical structure in the program to be written in an efficient and understandable manner. The purpose of structured programming is to make the software code easy to modify when required. Some languages such as Ada, Pascal, and dBase are designed with features that implement the logical program structure in the software code.

### Data-Flow Oriented Design

With the introduction of very Large Scale Integrated circuits (VLSI), the computers became more powerful and faster. As a result, various significant developments like networking and GUIs came into being. Clearly, the complexity of software could not be dealt using control flow based design. Thus, a new technique, namely, **data-flow-oriented** technique came into existence. In this technique, the flow of data through business functions or processes is represented using **Data-flowDiagram (DFD). IEEE** defines a data-flow diagram (also known as **bubble chart** and **work-flow diagram)** as 'a diagram that depicts data sources, data sinks, data storage, and processes performed on data as nodes, and logical flow of data as links between the nodes.'

### Object Oriented Design

Object-oriented design technique has revolutionized the process of software development. It not only includes the best features of structured programming but also some new and powerful features such as encapsulation, abstraction, inheritance, and polymorphism. These new features have tremendously helped in the development of well-designed and high-quality software. Object-oriented techniques are widely used these days as they allow reusability of the code. They lead to faster software development and high-quality programs. Moreover, they are easier to adapt and scale, that is, large systems can be created by assembling reusable subsystems.