E-Content for SEMESTER-VI
CMS-A-CC-6-13-TH
(Software Engineering)

by

Mitra Tithi Dey,

Assistant Professor, Department of Computer Science,

Netaji Nagar Day College

The term software engineering is composed of two words, software and engineering. Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product. Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods. So, we can define software engineering as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as: The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

- NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

Cost- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

Quality Management- Better process of software development provides better and quality software product.

- CHARACTERESTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- ✓ Operational
- ✓ Transitional
- ✓ Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- ✓ Budget
- ✓ Usability
- ✓ Efficiency
- ✓ Correctness

- ✓ Functionality
- ✓ Dependability
- ✓ Security
- ✓ Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- ✓ Portability
- ✓ Interoperability
- ✓ Reusability
- ✓ Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the everchanging environment:

- ✓ Modularity
- ✓ Maintainability
- ✓ Flexibility
- ✓ Scalability

● SOFTWARE DEVELOPMENT LIFE CYCLE LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out.

● THE NEED FOR A SOFTWARE LIFE CYCLE MODEL

The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure.

A few important and commonly used life cycle models are as follows:

- ✓ Classical Waterfall Model
- ✓ Iterative Waterfall Model
- ✓ Prototyping Model
- ✓ Evolutionary Model
- ✓ Spiral Model

● REQUIREMENTS ANALYSIS AND SPECIFICATION

Before we start to develop our software, it becomes quite essential for us to understand and document the exact requirement of the customer. Experienced members of the development team carry out this job. They are called as system analysts. The analyst starts requirements gathering and analysis activity by collecting all information from the customer which could be used to develop the requirements of the system. He/She then analyzes the collected information to obtain a clear and thorough understanding of the product to be developed, with a view to remove all ambiguities and inconsistencies from the initial customer perception of the problem.

The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?

• What exactly are the data input to the system and what exactly are the data output by the system?
• What are the likely complexities that might arise while solving the problem?
• If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

After the analyst has understood the exact customer requirements, he proceeds to identify and resolve the various requirements problems. The most important requirements problems that the analyst has to identify and eliminate are the problems of anomalies, inconsistencies, and incompleteness. When the analyst detects any inconsistencies, anomalies or incompleteness in the gathered requirements, he resolves them by carrying out further discussions with the endusers and the customers.

Parts of a Software Requirement Specification (SRS) document

The important parts of SRS document are:

- ✓ Functional requirements of the system
- ✓ Non-functional requirements of the system
- ✓ Goals of implementation

● SOFTWARE DESIGN

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently. Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software

Advantage of modularization:

- ✓ Smaller components are easier to maintain
- ✓ Program can be divided based on functional aspects
- ✓ Desired level of abstraction can be brought in the program
- ✓ Components with high cohesion can be re-used again.
- ✓ Concurrent execution can be made possible
- ✓ Desired from security aspect

Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

1) Co-incidental cohesion - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
2) Logical cohesion - When logically categorized elements are put together into a module, it is called logical cohesion.

3) Temporal Cohesion - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
4) Procedural cohesion - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
5) Communicational cohesion - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
6) Sequential cohesion - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
7) Functional cohesion - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -
1) Content coupling - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
2) Common coupling- When multiple modules have read and write access to some global data, it is called common or global coupling.
3) Control coupling- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
4) Stamp coupling- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

5) Data coupling- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

DFD

Data Flow Diagram Data flow diagram is a graphical representation of data flow in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data.

Difference between DFD and Flowchart

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

Data Flow Diagrams are either Logical or Physical.

a) Logical DFD - This type of DFD concentrates on the system process and flow of data in the system. For example in a Banking software system, how data is moved between different entities.

b) Physical DFD - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

- ## TESTING

Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.

Different steps for testing are:
1) Unit testing
2) Integration testing
3) System testing
4) Acceptance testing

● SOFTWARE MAINTENANCE

Software maintenance is becoming an important activity of a large number of software organizations. Given the rate of hardware obsolescence, the immortality of a software product, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary. Also, whenever the support environment of a software product changes, the software product requires rework to cope up with the newer interface.
For instance, a software product may need to be maintained when the operating system changes. Thus, every software product continues to evolve after its development through maintenance efforts. Therefore it can be stated that software maintenance is needed to correct errors, enhance features, port the software to new platforms, etc.

Types of software maintenance

There are basically three types of software maintenance. These are:
1) Corrective: Corrective maintenance of a software product is necessary to rectify the bugs observed while the system is in use.
2) Adaptive: A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
3) Perfective: A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customer demands, or to enhance the performance of the system.