SOFTWARE ENGINEERING

UNIT – 1

CHAPTER – 1

INTRODUCTION TO SOFTWARE ENGINEERING

I. Definition

Software is program/ set of programs containing instructions which provide desired functionality. Also comprises of data structures that enable the program to manipulate information

Engineering is the process of designing and building something that serves a particular purpose

Software Engineering is a systematic approach to the development, operation and maintenance of desired software

II. Evolving Role of Software

Software serves as a:

- Product: It delivers the computing potential of a H/W i.e., enables the h/w to deliver the expected functionality. Acts as information transformer
- Vehicle to deliver the product: Helps in creation and control of other programs i.e., it helps other software to do functions and helps as platform. E.g., Operating System

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information. Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway

to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms. The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.

III. Why Software Engineering is important?

i. Imposes discipline to work that can become quite chaotic - Lot of steps are involved in the development of a s/w, so if a systematic approach is not taken, it becomes difficult/clumsy

ii. Ensures high quality of software - If a s/w could deliver the features and functionalities required then it is a high-quality software.

iii. Enables us to build complex systems in a timely manner - Whenever we have a huge/complex project, then we need to set proper deadlines and milestones of the time taken in each step, so that in time we can deliver the s/w to the customer.

IV. Difference between Software and Hardware

Software:

- S/W is logical unit
- No spare parts for s/w
- Problem statement may not be complete and clear initially
- Requirements may change with time
- Multiple copies (less cost)
- Idealized and actual graph

Hardware:

- H/W is physical unit
- Spare parts for h/w exist

- Problem statement is clearly mentioned at the beginning of development
- Requirements are fixed before manufacturing
- Multiple copies (more cost)
- Bath tub graph

Bath Tub Curve:

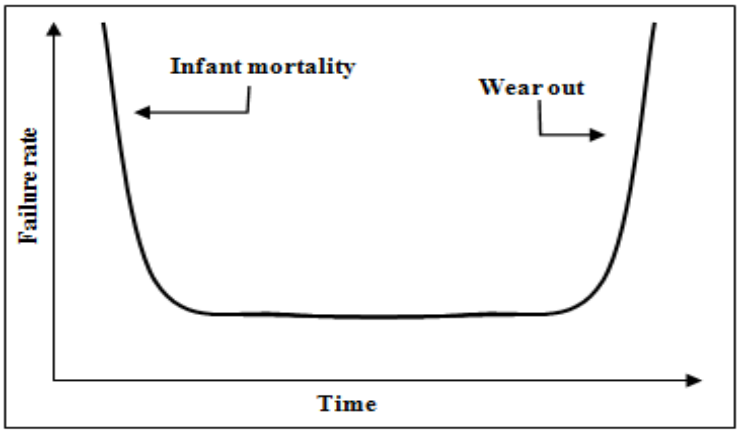


Figure 1: Bath tub Curve- Failure curve for Hardware

- Given the name bath tub because its shape is same as bath tub.
- It explains the reliability of the product i.e. until how many days / time the product works
- We have 3 stages in bath tub curve:
  - Decreasing failure rate or Infant mortality
  - Constant failure rate
  - Increasing failure rate or wear out (H/W effected by the environment factors like dust, temperature, pollution, etc). S/W does not have wear out.
- Decreasing failure rate:
  - As the product in this stage is new, there are very less chances of it to fail. The product still fails because of
    - Manufacturing defect
    - We haven't assembled it properly

- It is a weak part or product.

- Constant failure rate:
  - It is named as constant failure rate because the graph remains in straight line according to the time.
  - Most of the products fail in this stage
  - It is the service life of the product.
- Increasing failure rate:
  - This is the stage where we use the product more than its service period. The suddenly, the product may fail.
  - It is named increasing failure rate as the curve moves upwards as shown according to the time.

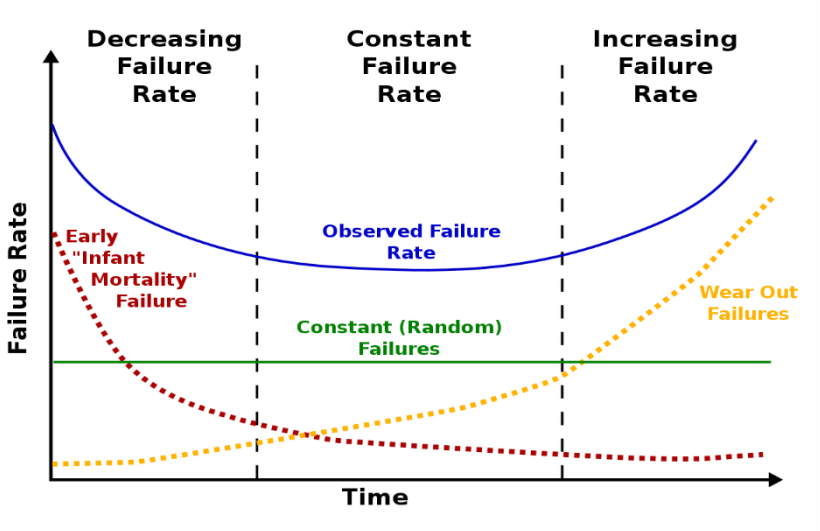


Figure 2: Failure rates in Bath tub Curve

Idealized Curve:

- Since there is no wear out in s/w the curve must undergo on 2 phases i.e., decreasing failure rate and constant failure rate which is called as Idealized curve.
- But in reality, idealized curve is not possible.
- Initial failure happens just like h/w due to undetected defects.

- After correcting the defects, the curve reaches a steady state, but if any change occurs in the s/w then there is spike in the graph.
- Most of the times the failure rate increases when a change effect is requested. The actual curve is higher than the idealized curve.

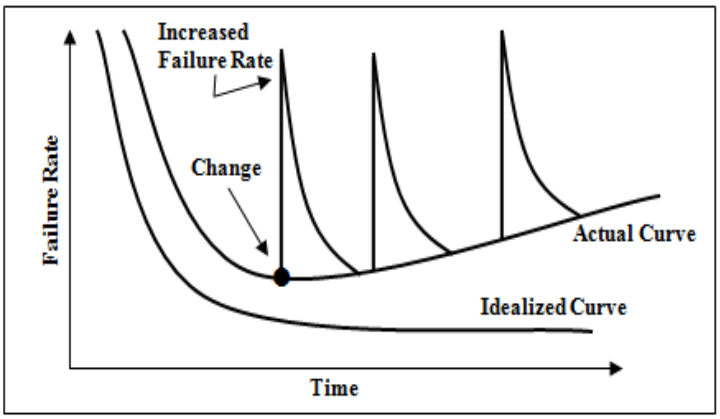


Figure 3: Idealized Curve- Failure curve for Software

V. Work Product:

Work product is the result or outcome of Software Engineering

The outcome is in 2 perspectives:

1. Software Engineer: The set of programs, the content (data) along with documentation that is a part of s/w.

2. User/Customer: The functionality delivered by the s/w that improves user experience

VI. Software Engineering focuses on:

- Quality (While Software development)

– Functional: To what extent that we have delivered the correct s/w and is it reaching the expectations. Degree to which correct software is produced.

– Non functional: Also called structural attributes. Features other than functions of s/w like robustness, security, etc.

- Maintainability (After the software has been developed and delivered)

- Should be easily enhanced and adapted to changing requirements whenever required.

VII. Changing Nature of Software:

The nature of software has changed a lot over the years.

1. **System software:** Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically, system software is a collection of programs to provide service to other programs.
2. **Real time software:** This software is used to monitor, control and analyse real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forecast the weather.
3. **Embedded software:** This type of software is placed in "Read-Only- Memory (ROM)" of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. The embedded software handles hardware components and is also termed as intelligent software.
4. **Business software:** This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.
5. **Personal computer software:** The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.
6. **Artificial intelligence software:** Artificial Intelligence software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network, signal processing software etc.

7. **Web based software:** The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

VIII. Legacy Software:

Legacy:

- Generally, something which u get in inheritance
- Example: Money / Status
- In terms of s/w, something which is old but still in use and difficult to replace.
- Example: Legacy car

Legacy System:

- The combination of legacy software and hardware overall is called legacy system.
- Ex: a s/w working in win XP is not compatible with windows 11.

Legacy Software:

- Outdated s/w still in use as it is fulfilling the needs of the organization
- Ex: Bank s/w
- Customize s/w: s/w once developed will not have any changes or new requirements then we use it for decade
- Ex: Games

Need to be replaced for the following reasons:

Due to change in business model

Due to change in architecture

Inadequate security, performance, flexibility, etc.

IX. Software Myths:

The development of software requires dedication and understanding on the developers' part. Many software problems arise due to myths that are formed during the initial stages of software development. Software myths propagate false beliefs and confusion in the minds of

management, users and developers. Blind belief that management, customers and practitioners have on s/w and the process to use it. Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations. Common management myths are:

- Management myths
- Customer myths
- Practitioner's myths

i. Management Myths:

Myth 1:

We have all the standards and procedures available for software development.

Fact:

- Software experts do not know all the requirements for the software development.
- And all existing processes are incomplete as new software development is based on new and different problem.

Myth 2:

The addition of the latest hardware programs will improve the software development.

Fact:

- The role of the latest hardware is not very high on standard software development; instead (CASE) Engineering tools help the computer, they are more important than hardware to produce quality and productivity.
- Hence, the hardware resources are misused.

Myth 3:

- With the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

Fact:

- If software is late, adding more people will merely make the problem worse. This is because the people already working on the project now need to spend time educating the newcomers, and are thus taken away from their work. The newcomers are also far less productive than the existing software engineers, and so the work put into training them to work on the software does not immediately meet with an appropriate reduction in work.

ii. Customer Myths:

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

Myth 1:

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

Fact:

- Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.
- Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer.

Myth 2:

Software requirements continually change, but change can be easily accommodated because software is flexible

Fact:

- It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small **(Refer Figure 4)**. However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

iii. Practitioner's Myths:

Myths 1:

They believe that their work has been completed with the writing of the plan.

Fact:

- It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

Myths 2:

There is no other way to achieve system quality, until it is "running".

Fact:

- Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test.

Myth 3:

An operating system is the only product that can be successfully exported project.

Fact:

- A working system is not enough, the right document brochures and booklets are also required to provide guidance & software support.

Myth 4:

Engineering software will enable us to build powerful and unnecessary document & always delay us.

Fact:

- Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

X. SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY

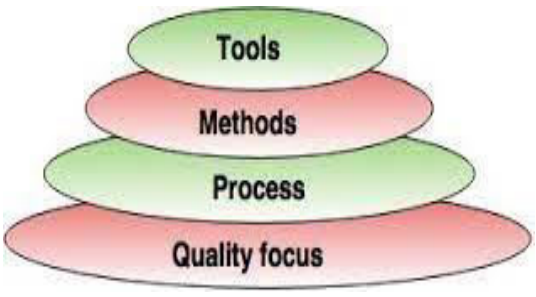
IEEE has developed a more comprehensive definition of software engineering:

"Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."

- **Software Engineering is a layered technology.**

Software Engineering encompasses a **Process, Methods** for managing and engineering software and **tools**.

The following Figure represents **Software engineering Layers**



Software Engineering – A Layered Technology

Referring to the above Figure, any engineering approach must rest on an organizational commitment to **quality**. The bedrock that supports software engineering is a **quality focus**. The foundation for software engineering is the ***process* layer**. The software engineering process is the glue that holds the technology layers together and enables the rational and timely development of computer software. **Process** defines a **framework** that must be established for the effective delivery of software engineering technology.

Software engineering ***methods*** provide the technical **how-to** for building software. **Methods** encompass a broad array of tasks that include communication, requirements analysis, design modelling, program construction, testing, and support. Software engineering ***tools*** provide **automated or semi-automated** support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

XI. THE SOFTWARE PROCESS FRAMEWORK

A ***process*** is a collection of **activities, actions, and tasks** that are performed when some work product is to be created.

An ***activity*** strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.

An ***action*** encompasses a set of tasks that produce a major work product (e.g., an architectural design model).

A ***task*** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

A ***process framework*** establishes the foundation for a complete software engineering process by identifying a small number of ***framework activities*** that apply to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of ***umbrella activities*** that are applicable across the entire software process.

A generic process framework for software engineering encompasses **five** activities:

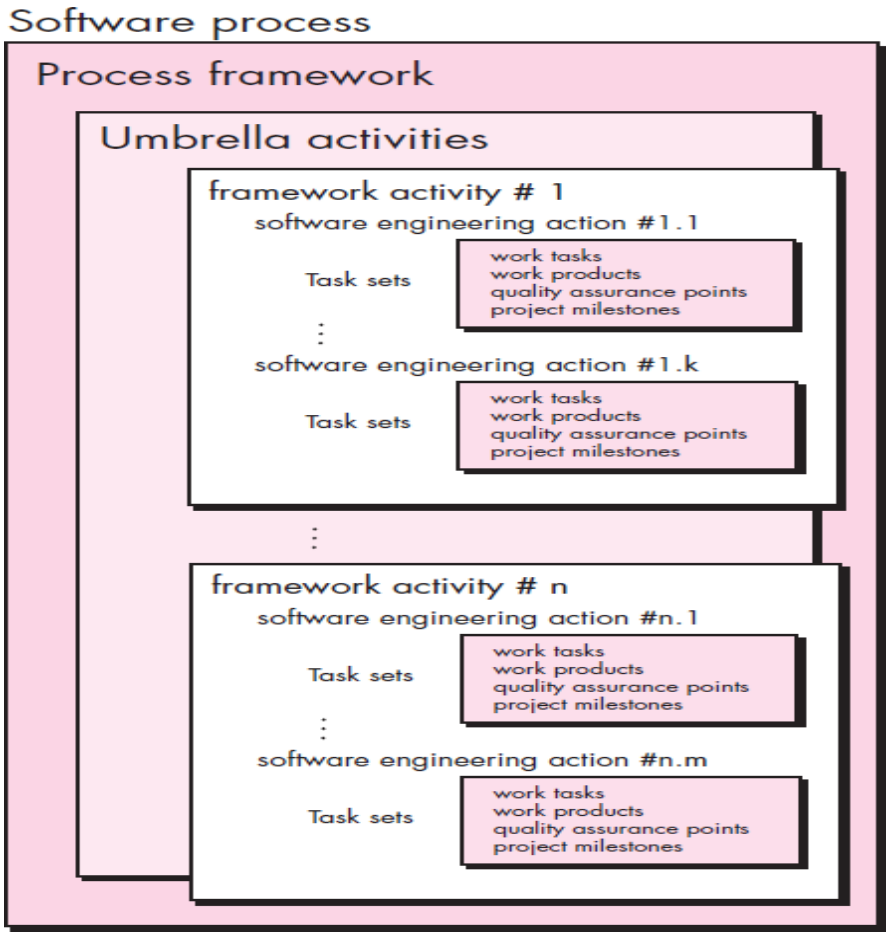
- **Communication**
- **Planning**
- **Modelling**
- **Construction**
- **Deployment**

These **five** generic framework activities can be used during the development of small, simple programs, the creation of large Web applications, and for the engineering of large, complex computer-based systems.

Software engineering process framework activities are complemented by several U**mbrella Activities**. In general, **umbrella activities** are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
- Software configuration management
- Reusability management
- Work product preparation and production

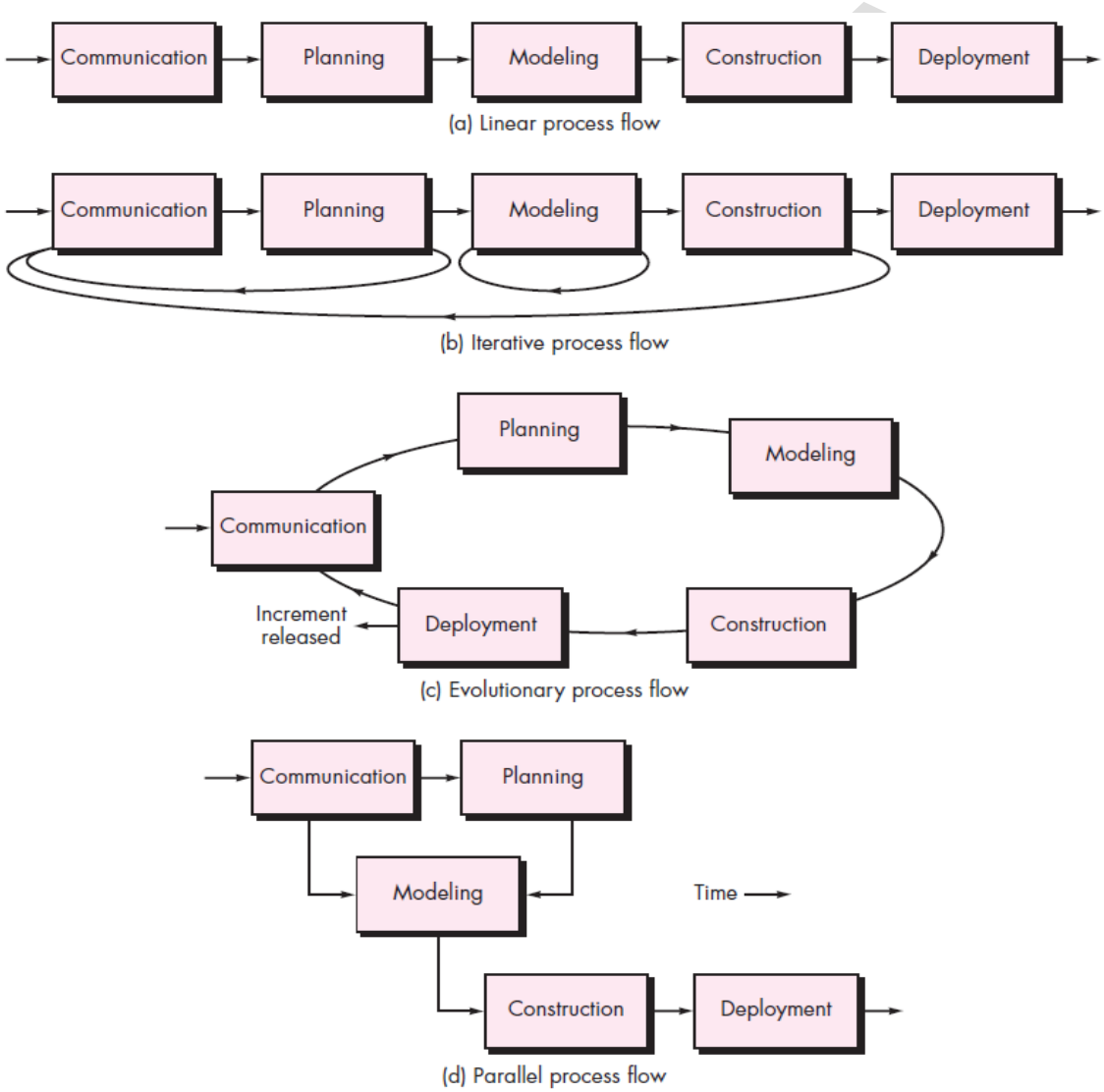
XII. A GENERIC PROCESS MODEL



The software process is represented schematically in the following figure. Each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.

A generic process framework defines five framework activities—communication, planning, modeling, construction, and deployment. In addition, a set of umbrella activities like project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

This aspect is called process flow. It describes how the framework activities and the actions and tasks that occur within each framework activity are organized concerning sequence and time and is illustrated in the following figure



A generic process framework for software engineering A linear process flow executes each of the five framework activities in sequence. An iterative process flow repeats one or more of the activities before proceeding to the next. An evolutionary process flow executes the activities in a "circular" manner. A parallel process flow executes one or more activities in parallel with other activities.