SOFTWARE ENGINEERING

Software and Software Engineering: The Nature of Software, The Unique Nature of Web Apps, Software Engineering, Software Process, Software Engineering Practice, Software Myths.

Process Models: A Generic Process Model like Waterfall Models, Agile Model etc. Process Assessment and Improvement, Prescriptive Process Models, Specialized Process Models, The Unified Process, Personal and Team Process Models, Process Terminology, Product and Process.

PART-1: SOFTWARE AND SOFTWARE ENGINEERING

INTRODUCTION

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

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

Software is a program or set of programs containing instructions which provide desired functionality.

Engineering is the application of scientific and practical knowledge and processes of designing a system that serves a particular purpose and find a cost effective solution to problems.

THE NATURE OF SOFTWARE

The key role of Software if to perform task foe the user by activity and controlling the computer hardware. Software can have number of attributes which together can decide whether it is good or bad.

- **Correctness:** Software is functionally correct if it behaves accordingly to the functional requirements specifications. Correctness is a mathematical property requires that a specification be available must be possible to determine whether software meets the specifications.
- **Reliability:** Software is said to be reliable, if the user can depend on its statistical property that the software will operate as expected over a specified period of time.

- **Robustness:** Software is robust, if it behaves reasonably even in all circumstances that were not anticipated in the requirements specifications.
- **Performance:** A Software performance is equated with efficiency, where resources are available economically. It also affects usability & scalability.
- **Usability:** Software is usable if its end users find it easy to use. Which is an extremely subjective property.
- **Understandability:** Software is said to be understable if it is easy for developers to understand the produced artifacts. It also effects an internal product quality.
- **Verifiability:** Software is verifiable if it satisfaction of desired properties can be easily determined. It also depends on internal quality.
- **Maintainability:** Software is maintainable if it can be modified easily after a version release improvements rather than upkeep as in other engineered products evidence shows that maintenance cost exceed 60% of total software costs.
- **Repairability:** Software is repairableif it allows defect correction with limited effort in other disciplines. It also addresses corrective maintenance.
- **Evolability:** Software is evolvable if it facilities additiom of functionality or modification of existing function. Resolvability addresses adaptive and perfective maintenance.
- **Reusability:** Software is reusable, if it can be used perhaps with minor modification to construct another product.
- **Portability:** Software is known to be portable, if it can run in different environments with little. To enhance by assuming with minimal environment capabilities.

THE UNIQUE NATURE OF WEB APPS

In the early days of the World Wide Web (1990 to 1995), websites consisted of little more than a set of linked hypertext files that presented information using text and limited graphics.

Today, WebApps have evolved into sophisticated computing tools that not only provide stand-alone function to the end user, but also have been integrated with corporate databases and business applications due to the development of HTML, JAVA, xml etc.

Attributes of WebApps :

Network intensiveness: A WebApp resides on a network and must serve the needs of a diverse community of clients. The network may enable worldwide access and communication (i.e., the

Internet) or more limited access and communication**.** E.g: A corporate Intranet Network Intensiveness)

Concurrency: A large number of users may access the WebApp at one time. In many cases, the patterns of usage among end users will vary greatly.

Unpredictable load: The number of users of the WebApp, may vary by orders of magnitude from day to day. One hundred users may show up on Monday; 10,000 may use the system on Thursday.

Performance: If a WebApp user must wait too long (for access, for server side processing, for client-side formatting and display), he or she may decide to go elsewhere.

Availability: Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a 24/7/365 basis

Data driven: The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end user.

In addition, WebApps are commonly used to access information that exists on databases that are not an integral part of the Web-based environment (e.g., e-commerce or financial applications).

Content sensitive: The quality and artistic nature of content remains an important. Determinant of the quality of a WebApp.

Continuous evolution: Unlike conventional application software that evolves over a series of planned, chronologically spaced releases, Web applications evolve continuously. It is not unusual for some WebApps (specifically, their content) to be updated on a minute-by-minute schedule or for content to be independently computed for each request.

Immediacy: Although immediacy—the compelling (forceful) need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time-to-market that can be a matter of a few days or weeks.

Security: Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end users who may access the application. In order to protect sensitive content and provide secure mode of data transmission, strong security measures must be implemented.

Aesthetics : [Artistic / Visual] An undeniable part of the appeal of a WebApp is its look and feel. When an application has been designed to market or sell products or ideas, aesthetic may have as much to do with success as technical design.

SOFTWARE CONCEPTS

Defining Software:

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information, and

(3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

Software characteristics:

1. **Software is developed or engineered: it is not manufactured in the classical sense.** Although some similarities exist between software development and hardware manufacturing, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software. Software projects cannot be managed as if they were manufacturing projects.
2. **Software doesn't "wear out.":** Figure below depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life; defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.

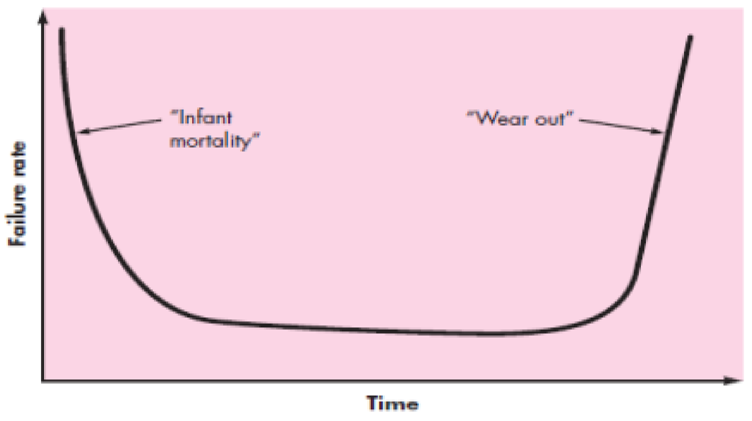


Fig: Failure curve of Hardware

3. **Although the industry is moving toward component-based construction, most software continues to be custom built:** As an engineering discipline evolves, a collection of standard design components is created. The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new.

Software Application Domains:

Today, seven broad categories of computer software present continuing challenges for software engineers:

- **System software**—a collection of programs written to service other programs.
- **Application software**—stand-alone programs that solve a specific business need.
- **Engineering/scientific software**—has been characterized by "number crunching" algorithms.
- **Embedded software**—resides within a product or system and is used to implement and control features and functions
- **Product-line software**—designed to provide a specific capability for use by many different customers.
- **Web applications**—called "WebApps," this network-centric software category spans a wide array of applications.
- **Artificial intelligence software**—makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.

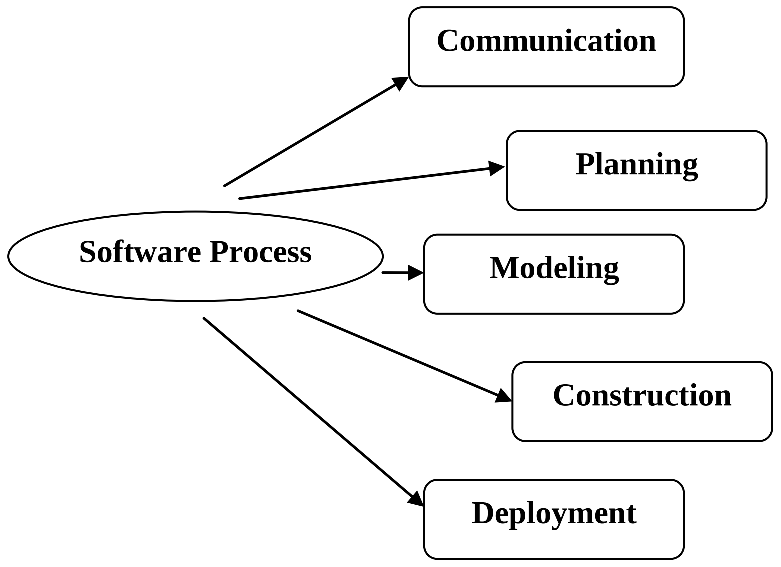
SOFTWARE PROCESS

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

NOTE: An activity strives to achieve a broad objective with which software engineering is to be applied. **An action** encompasses a set of tasks that produce a major work product. **A task** focuses on a small, but well-defined objective 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 are applicable to all software projects, regardless of their size or complexity.

A generic process framework for software engineering encompasses five activities:



- **Communication:** Before any technical work can commence, it is critically important to communicate and collaborate with the customer. The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
- **Planning:** The planning activity creates a "map" called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

- **Modeling:** You create a “sketch” of the thing so that you’ll understand the big picture. A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.
- **Construction:** This activity combines code generation and the testing that is required to uncover errors in the code.
- **Deployment:** The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

SOFTWARE ENGINEERING PRACTICE

It consists of a collection of concepts, principles, methods and tools that a software engineer calls upon on a daily basis. Provides necessary technical and management skills, how to get the job to be done.

Essence of Practices are as follows

1. **Understand the problem (communication and analysis):** It’s worth spending a little time to understand, answering a few simple questions:

   • Who has a stake in the solution to the problem? That is, who are the stakeholders?

   • What are the unknowns? What data, functions, and features are required to properly solve the problem?

   • Can the problem be compartmentalized? Is it possible to represent smaller problems that may be easier to understand?
2. **Plan a solution (modeling and software design):** Now you understand the problem and you can’t wait to begin coding. Before you do, slow down just a bit and do a little design:

   • Have you seen similar problems before? Are there patterns that are recognizable in a potential solution?

   • Is there existing software that implements the data, functions, and features that are required?

   • Has a similar problem been solved? If so, are elements of the solution reusable?
3. **Carry out the plan (code generation):** Now you understand the problem (or so you think) and you can’t wait to begin coding. Before you do, slow down just a bit and do a little design:

   • Have you seen similar problems before? Are there patterns that are recognizable in a potential solution?

   •Is there existing software that implements the data, functions, and features that are required?

   • Has a similar problem been solved? If so, are elements of the solution reusable?

   • Can sub problems be defined? If so, are solutions readily apparent for the sub problems?

4. **Examine the result for accuracy (testing and quality assurance):** You can't be sure that your solution is perfect, but you can be sure that you've designed a sufficient number of tests to uncover as many errors as possible.

   • Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented?

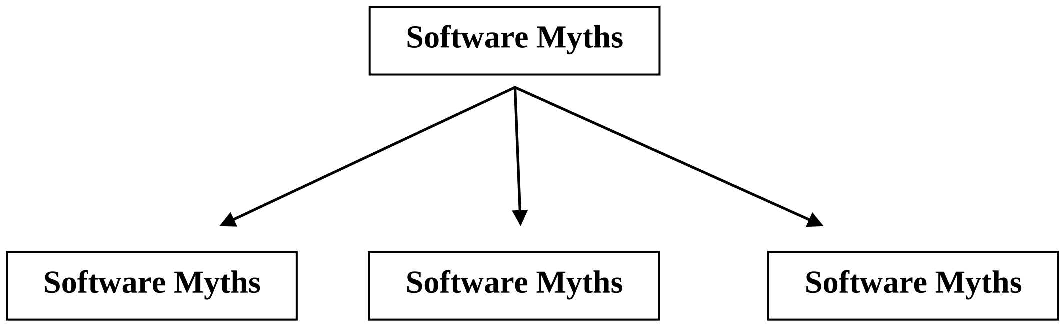
   • Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?

SOFTWARE MYTHS

Myth: It's a wrong belief and mis information. Myths have number attributes that causes serious problem on software.

Software Myth: These are the beliefs about software and process used to build it.

Types of Software Myths:



- **Management myths:** Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.
  (1) Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?
  (2) Myth: If we get behind schedule, we can add more programmers and catch up
  (3) Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.
  (4) **Reality:** Are software practitioners aware of its existence? Software development is not a mechanistic process like manufacturing. In the words of Brooks "adding people to a late software project makes it later."

- **Customer myths:** A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract.
  1. Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.
  2. Myth: Software requirements continually change, but change can be easily accommodated because software is flexible.
  3. **Reality:** Unambiguous requirements are developed only through effective and continuous communication between customer and developer. When requirements changes are requested early the cost impact is relatively small.
- **Practitioner's myths:** Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture
  1. Myth: Once we write the program and get it to work, our job is done.
  2. Myth: Until I get the program "running" I have no way of assessing its quality.
  3. Myth: The only deliverable work product for a successful project is the working program
  4. **Reality:** Software are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects. A variety of work products provide a foundation for successful engineering and, more important, guidance for software support.