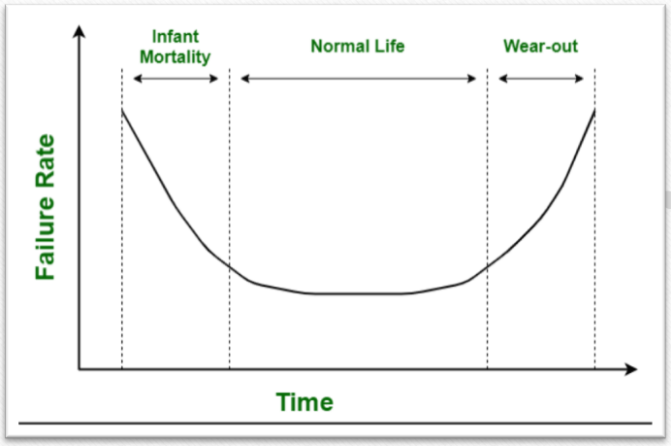
What is a software?

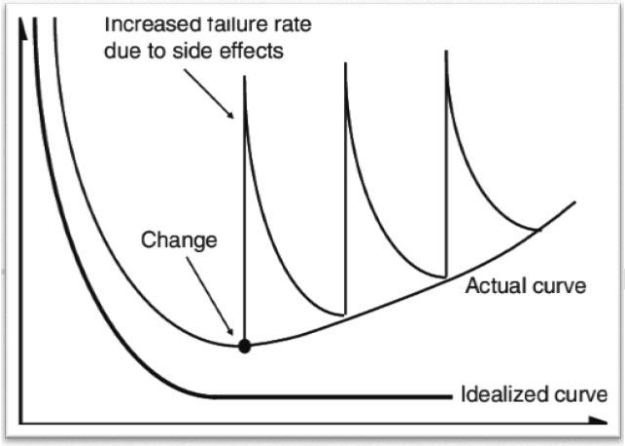
- Software engineering defines software as computer programs, procedures, rules and possibly associated documentation and data planning to the operation of a computer system.
- Software can also be defined as:
  - **Computer programs** when executed provide desired results and performance.
  - **Data structure** that enable the programs to follow manipulation.
  - **Documents** that describe the operations and use of the program.

Characteristics of a software:

1. **Software is developed or engineered, it is not manufactured:**
   - Software is logical and not physical.
   - There are many automated software development tools but it is the skill of individuals, creativity of developers and proper management by project manager that count for a good software product.
   - In both software and hardware, high quality is achieved by good design but the manufacturing phase for hardware can introduce quality problems that are not existing for software.
2. **Software does not "wear out":**
   - Hardware Bathtub Curve: This relationship often called the "bathtub curve" indicates that hardware exhibits relatively high failure rate early in its life, defects are corrected and the failure rate drops to a steady state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibrations, abuse, temperature extremes, etc. Stated simply, hardware begins to wear out.



Hardware Bathtub Curve



Software Bathtub Curve

- Software Bathtub Curve: The failure rate curve for software shows that, undiscovered defects will cause high failure rates early in the life of a program. . These are corrected the curve flattens as shown in figure (B) Software doesn't wear out, but it does deteriorate. During the software life, it will undergo change (maintenance). As changes are made, it is likely that some new defects will introduced, causing the failure rate cure to spike. Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise the software is deteriorating due to change.

3. **Most software is custom-built, rather than being assembled from existing components:**
   - Most of the engineered products are first designed before they are manufactured, Designing includes identifying various components for the product before they are actually assembled. Here several people can work independently on these components thus making the manufacturing system highly flexible. In software, breading a program into modules is difficult task, since each module is highly interlinked with other modules. Further, it requires lot of skill to integrate different modules into one.

Characteristics of well-engineered/ideal software are:

- Efficiency
- Maintainability
- On time
- Within budget
- Functionality
- Adaptability
- Dependability
- Usability

Software Engineering:

- The systematic approach to the development, operations, maintenance and refinement of software.
- It is the application of a systematic, disciplined and quantifiable approach to the development, operation and maintenance of software that is the approach of engineering to the software.
- The main motive of software engineering is to improve the quality of software product and to increase the productivity and job satisfaction of software engineers.

Types of Software:

1. **System Software:**
   - The system software is a collection of programs designed to operate, control, and extend the processing capabilities of the computer itself.
   - The system software area is characterized by heavy interaction with computer hardware, heavy usage by multiple users, parallel operation that requires scheduling, resource sharing, complex data structures, and multiple external interfaces.

- Ex: Windows, IOS, Linux, Android, etc.

2. **Application Software:**
   - Application software products are designed to satisfy a particular need of a particular environment.
   - Application software may consist of a single program, such as Microsoft's notepad for writing and editing a simple text. It may also consist of a collection of programs, often called a software package, which work together to accomplish a task, such as a spreadsheet package.
3. **Embedded Software:**
   - Embedded software is specialized programming in a chip or on firmware in an embedded device to controls its functions. Hardware makers use embedded software to control the functions of various hardware devices and systems.
   - Ex: dedicated GPS devices, factory robots, some calculators and even modern smart watches.
4. **Web based Software:**
   - Web-Based Software is software you access with just an internet connection and a web browser.
   - There is no software or hardware to purchase, no need to download software, or ever worry about costly product upgrades.
   - Ex: Google docs, Canva, Gmail, Google sheets, etc.
5. **Artificial Intelligence Software:**
   - Artificial intelligence software definition: "Software that is capable of intelligent behaviour." In creating intelligent software, this involves simulating a number of capabilities, including reasoning, learning, problem solving, perception, knowledge representation.
6. **Business Software:**
   - Business software (or a business application) is any software or set of computer programs used by business users to perform various business functions. These business applications are used to

increase productivity, measure productivity, and perform other business functions accurately.

- Ex: Accounting software, billing software, payroll software, etc.

7. Real Time Software:

- Real time is a guaranteed level of computer responsiveness within a specified time constraint, usually milliseconds or microseconds, between an event and its response deadline. Real time describes a human sense of time (rather than machine time) that seems immediate.
- Ex: aircraft navigation programs, multimedia broadcasts, multi-player video games, data analysis programs and stock-trading applications.
- Types of Real time software:
  - Hard real time: Medical critical care system, Aircraft systems, etc.
  - Firm real time: Various types of Multimedia applications.
  - Soft real time: Online Transaction system and Livestock price quotation system.

Software Myths:

1. Management Myths:

a. 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?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no".

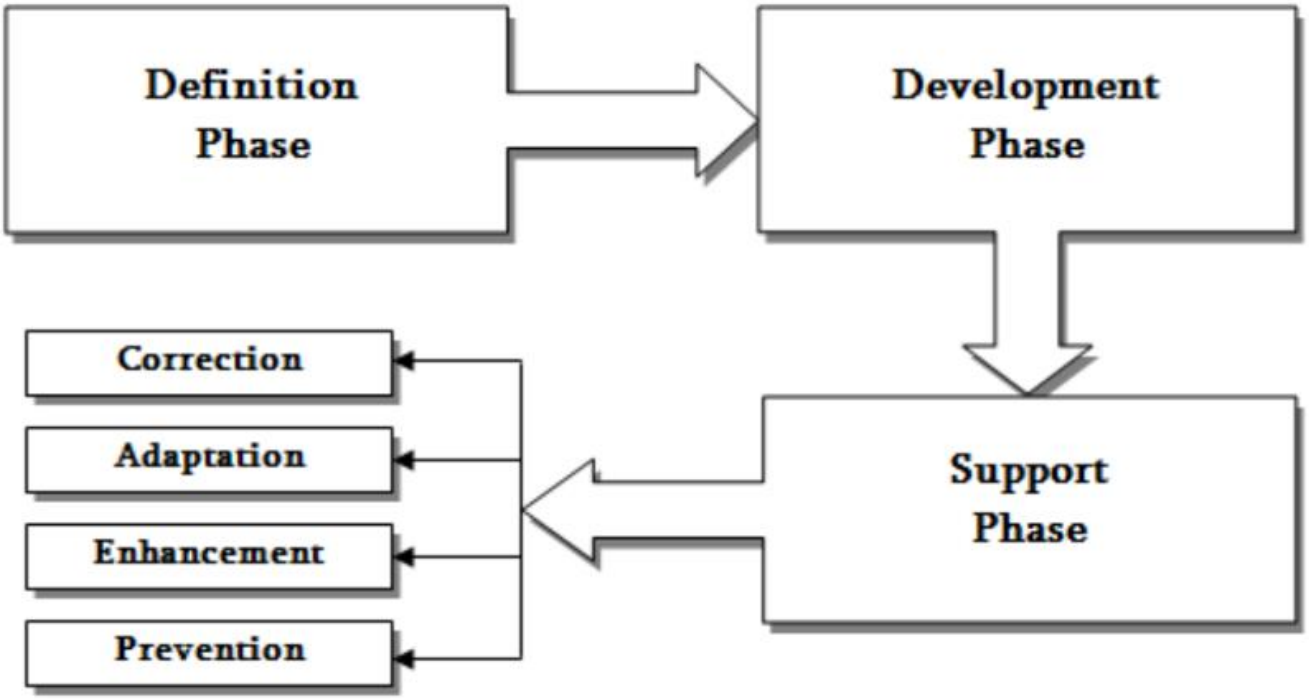
b. **Myth:** If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept).
   Reality: It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development. Computer-aided software engineering (CASE) tools are more important than hardware for achieving good quality and productivity, yet the majority of software developers still do not use them effectively.
c. **Myth:** My people have state-of-art software development tools, after all, we buy them the newest computers.
   Reality: People who were working must spend time in educating the new comers.
d. **Myth:** If I decide to outsource the software project to a third party, I can just relax and let that firm build it.
   Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

2. **Customer's Myths:**
   a. **Myth:** A general statement of objective is sufficient to begin writing programs we can fill in the details later.
      Reality: poor up-front definition is the major cause of failed software efforts.
   b. **Myth:** Project requirements continually change, but change can be easily accommodated because software is flexible.
      Reality: It is true that software requirements change, but the impact of change project schedule and planning will disturb.
3. **Practitioner's Myths:**
   a. **Myth:** Once we write the program and get it to work, our job is done.
      Reality: Software can be expended after it is delivered to the customer for the first time.

b. **Myth:** Until I get the program running, I have no way of assessing its quality.
   Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project — the formal technical review. Software reviews are a 'quality filter' that have been found to be more effective than testing for finding certain classes of software defects.
c. **Myth:** The only deliverable work product for a successful project is the working program.
   Reality: A working program is only one part of a software configuration that includes any elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.
d. **Myth:** Software Engineering will make us to create voluminous and unnecessary documentation and will invariably slow us down.
   Reality: Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Software Engineering – A Generic View:

- Engineering is the analysis, design, construction, verification and management of technical entities.
- There might be some questions regardless of the entity to be engineered. They are as follow:
  - What is the problem to be solved?
  - What characteristics of the entity are used to solve the problem?
  - How will the entity (and the solution) be realized?



1. **Definition Phase:**
   - The definition phase focuses on —what‖. That is, during definition, the software engineer attempts to identify:
     - What information is to be processed?
     - What function and performance are desired?
     - What system behavior can be expected?
     - What interfaces are to be established?
     - What design constraints exist?
     - What validation criteria are required to define a successful system?
   - During this, three major tasks will occur in some form: system or information engineering, software project planning and requirements analysis.
2. **Development Phase:**
   - The development phase focuses on "How". That is, during development a software engineer attempts to define:

- How data are to be structured?
- How function is to be implemented within a software architecture?
- How interfaces are to be characterized?
- How the design will be translated into a programming language?
- How testing will be performed?

- During this, three specific technical tasks should always occur: software design, code generation, and software testing.

3. Support Phase:

- The support phase focuses on "Change" associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements.
- Four types of change are encountered during the support phase:

a. Correction: Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.

b. Adaptation: Over time, the original environment, that is, CPU, operating system, business rules etc. for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.

c. Enhancement: As software is used, the customer/user will recognize additional functions that will provide benefit. Perfectible maintenance extends the software beyond its original functional requirements.

d. Prevention: Computer software deteriorates due to change, and because of this, preventive maintenance, often called software re-engineering, and must be conducted to enable the software to serve the needs of its end users. The phases and related steps

described in generic view of SE are complemented by a number of Umbrella Activities (which are the non SDLC activities that span across the entire software development life cycle) as under:

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software Configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management