Introduction to Software Engineering

Computer software is the product that software professionals build and then support over the long term.
Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high quality computer software.

Nature of software system

- Ubiquitous (Omni-present)
  - It touches all aspects of human lives.
  - It is used in variety of applications. (Transportation, medical, telecommunications, military, industrial, entertainment, office machines… list is endless)
- simple to complex
  - Calculator is a simple software system while an operating system is a complex software system.
- internal to public
  - They may be meant for internal use within an organization (Like egov website) or they may be meant for public use such as the railway reservation system that we are all familiar with.
- single function to enterprise wide
  - The software solution could be something for a very single well defined task such as a payroll, or it could be an application which covers all functions of a business organization, that means it could be enterprise wide software.
- One location to distributed
  - Text editor runs at one location but banking software is a distributed over a network.
- batch or real-time
  - A batch job is a program that is assigned to the computer to run without further user interaction.
  - A real time software requires continues interaction with the user. E.g. online billing system.

Roles of software

- **Product** that performs computation and also works as information manager
- **Vehicle** for delivering a product i.e. as vehicle it provides medium to reach the destination, software like IDE can be used to develop a website.
- We are talking about software solutions which are of long duration.
- They give important service to people or organizations and this is what would be the focus of software in this course

We must realize that developing software is quite different from developing one time program we typically develop as a student or for ourselves.

Major challenges in developing software

- effort intensive (we have to organize the work very meticulously)
- high cost
- long development time
- changing needs for user
- high risk of failure
  - user acceptance (user may not accept the solution)
  - performance (solution may be give expected performance)
  - maintainability (solution cannot be maintained up to date)

Software systems are not always been very successful. Success rate is much lower compared to the success rate in other engineering domains.

When do we call a software system to be successful?

- ✓ development completed
- ✓ useful
- ✓ usable
- ✓ used
- ✓ cost effective
- ✓ maintainable

Reasons for failure

- schedule slippage
- cost over-runs
- does not solve user's problem
- poor quality
- poor maintainability

What are the reasons for such problems?

- Ad hoc software development results in such problems
- No planning of development work i.e. milestones
- Deliverable to user not identified (what exactly is to be given (delivered) to user is not identified) this is because of next point.
- Poor understanding of user requirements
- No control/ review
  For any large project it is very important that we control its development very systematically, we review the progress systematically. Because we are committing lot of resources, we are committing lot of cost and time. Therefore it is very important that we control and review the progress continuously.
- Technical incompetence of developers
- Poor understanding of cost and effort by both developer and user

Let us look at **other engineering disciplines**

- large projects are common: successful
  Examples: buildings, bridges, dams, aircrafts, missiles, etcetera.
- Engineering a solution
  - to design, develop and artifact that meets specifications efficiently, cost effectively and ensuring quality using scientific principles
- require well defined approach
  - repeatable
  - predictable

Software VS Other engineering domains

- the cost of production in software is concentrated in development
- maintenance: making corrections, enhancements
- progress is difficult to measure

Apply engineering approach

- planning and control are more important
- estimate cost and effort
- plan and schedule work
- involve user in defining requirements (ultimately user must be able to use the software)
- identify stages in development
- clear milestones for progress
- review for control and quality (very important to review during the development)
- define deliverables (define what exactly is to be delivered to the end-user)
- plan extensive testing (testing must be planned)

Defining Software:

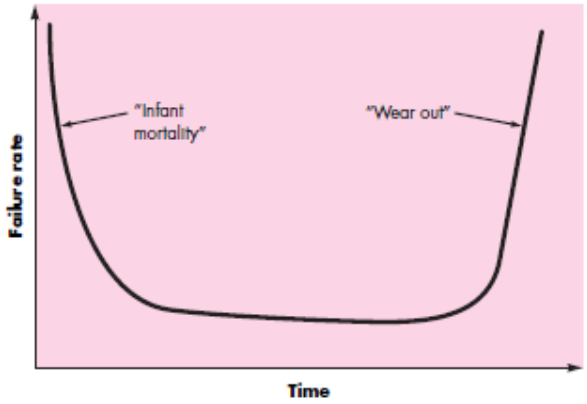
Software is:

(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.

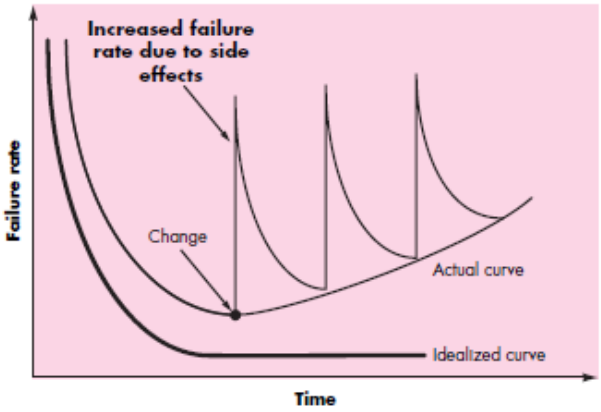
Characteristics of software as compared to hardware

1. **software is developed or engineered; it is not manufactured in classical sense**
   Quality: in software if some part fails then we can rework and make it right. While in hardware if some part fails then we cannot make it right. We need to replace the parts.
   People: in hardware, work is directly proportionate to no. of people involved while in software it is not the case.

2. **software doesn't wear out**



Failure curve for hardware



Failure curve for software

The failure curve for hardware is also known as bathtub curve.
Due to design or manufacturing defects, there is high failure. Once the defects are corrected the failure drops and becomes steady. Then with time, hardware wears out, rising the curve.

Idealize curve for software. Changes made in the software causes the failure rate to spike in actual curve.

As the software deteriorates, the software maintenance tasks that accommodate requests for change involve considerably more complexity then hardware maintenance.

When hardware completely fails, it is out of use and it might be possible that some of its components are reusable. If software fails, it will become obsolete and of no use.

3. although industry is moving toward component-based construction, most software continues to be custom built

These simple realities lead to one conclusion: software in all of its forms and across all of its application domains should be engineered.

IEEE definition:
Software Engineering: (1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

Software Application Domains

1. **System Software**: collection of programs written to service other programs
   E.g. operating system, compilers, linkers, etc.
2. **Application Software**: stand-alone programs that solve a specific business need
   e.g. IDE etc.
3. **Engineering/scientific software**: design based on some algorithm(numerical algorithms)
   e.g. astronomy, automated manufacturing, simulators etc.
4. **Embedded software**: e.g. robot (software made to run a hardware), driverless car, keypad of microwave, etc
5. **Product-line software**: generic software (it is designed for many number of customers or users) e.g. word processing, spreadsheets, entertainment, database management, etc.
6. **Web applications**
7. **AI software**: uses non-numerical algorithms to solve complex problems.
   E.g. gaming software, symptom checker, etc.

Legacy Software:

In computing, a legacy system is an old method, technology, computer system, or application program, "of, relating to, or being a previous or outdated computer system," yet still in use.

E.g. NASA was using space shuttle software developed in 1970's for about 30-35 years.

Mainframes + COBOL (Common business oriented language) can be considered legacy systems.

Windows XP: its support was ended in 2014 but still till today many countries are using it without upgrading because they have XP applications that are difficult or impossible to migrate to a newer version of Windows.

Many legacy systems remain supportive to core business functions and are indispensable (so good or important that you could not manage without it).

But, they may have **poor quality** in terms of

- Inextensible design
- Convoluted code (complicated and complex)
- Poor or nonexistent documentation
- Test cases and results that were never archived
- Poorly managed change history

So, now what to do??

A. Do nothing
B. Evolve i.e. re-engineer

Remember: Change is Constant