INTRODUCTION TO SOFTWARE ENGINEERING

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.

There are two types of software:

- Generic – developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
- Bespoke (Custom) – developed for a single customer according to their specifications.

New software can be created be developing new programs, configuring generic software systems or re-using existing software.

Software Engineering

*An engineering discipline that is **concerned with all aspects of software production** from the early stages of system specification to maintaining the system after it has gone into use.*

Two key phrases in the above definition:-

1. ***Engineering Discipline***:
   - The creative application of scientific principles to design and develop useful things.
   - Involves selectively applying ***theory***, ***methods*** and ***tools*** where these are appropriate.
   - Engineers always try to discover solutions to problems even when no applicable theories and methods.
   - They must work to ***organizational and financial constraints***.
2. ***All aspects of software production***:
   - SE is not just concerned with the technical processes of software development
     - but also with activities such as ***software project management*** and
     - with the ***development*** of tools, methods and theories to support software production.

Software Engineering VS Computer Science

- Computer Science is concerned with the basic principles and methods underlying computers and software systems.
- Software Engineering is concerned with the practical problems of producing software.
- Ideally, all of software engineering should be underpinned by theories of computer science, but in reality this is not the case.
- Software engineers must often use *ad hoc* approaches to developing the software.
- Elegant theories of computer science cannot always be applied to real, complex problems that require a software solution.

Software Process

Set of activities that lead to the production of a software product.

- Some fundamental activities are common to all software processes:
  - *Software specification*: The functionality of the software and constraints on its operation must be defined.
  - *Software design and implementation*: The software to meet the specification must be produced.
  - *Software validation*: The software must be validated to ensure that it does what the customer wants.
  - *Software evolution*: The software must evolve to meet changing customer needs.
- Software processes can be improved by process standardization where the diversity in software processes across an organization is reduced.
  - This leads to improved communication and a reduction in training time and makes automated process support more economical.

Software Development Life Cycle:
The activities through which software passes during its development.

Phase 0: Feasibility Study
Cost/benefit Analysis (Is it worth developing software?)

Phase 1: Requirement Analysis and Specification
Concept Exploration: Requirements are identified in detail and a clear understanding is made of what is desired from the software.
Client's requirements are produced in the form of a document called SRS (Software Requirement Specification). SRS sometimes be treated as a legal contract between client and developer.
This phase is performed by a senior position holder called System Analyst.

Phase 2: Planning
The Project Plan is created that describes cost estimation, schedule, personnel allocation etc. This phase is accomplished by Project Manager.

Phase 3: Design
This phase brings from Problem domain to solution phase.

Phase 4: Coding and Module (Unit) Testing
Translate the design into PL and test for the existence of errors.

Phase 5: Integration & System Testing
Overall system is tested after integration. Acceptance testing by the client follows system testing. Testers are usually part of SQA team not SE team.

Phase 6: Maintenance
Longest Phase.

Phase 7: Retirement

Costs of Software Engineering:
Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs. Costs vary depending upon the type of system being developed.

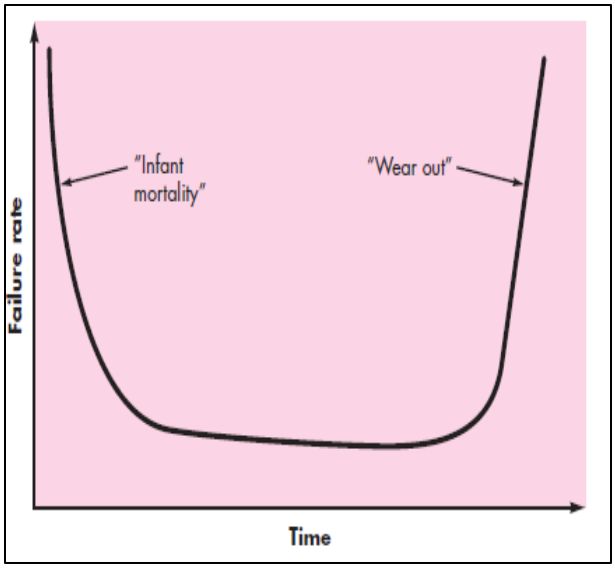
CASE (Computer Aided Software Engineering):
Software systems that are intended to provide automated support for software process activities.

- Upper-CASE: Tools to support the early process activities of requirements and design.
- Lower-CASE: Tools to support later activities such as programming, debugging and testing.

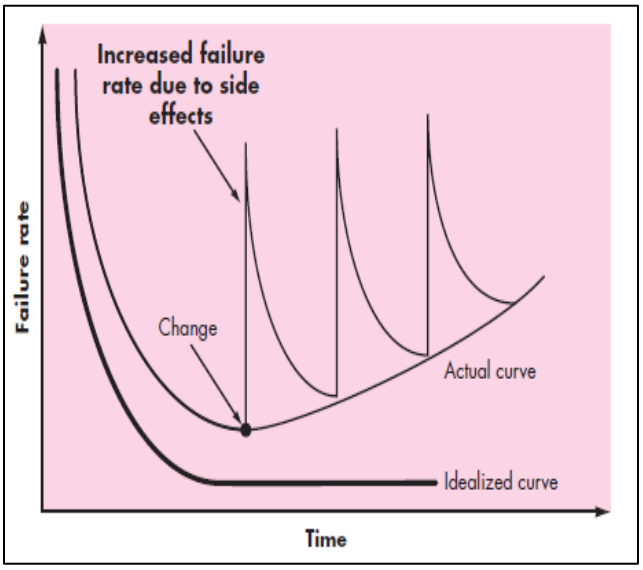
Software Characteristics:

Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware:

1. Software is developed or engineered; it is not manufactured in the classical sense.
2. Software doesn't "wear out."
3. Although the industry is moving toward component-based construction, most software continues to be custom built.
4. Software is intangible.



Failure Curve for Hardware



Idealized and actual failure curves for software

Attributes of Good Software

- Includes *Services* provided by software
- Other associated attributes that reflect the *quality* of that software.
  - They reflect its *behavior* while it is executing and
  - the structure and organization of the source program and the associated documentation.
  - Examples of these attributes (sometimes called non-functional attributes) are the *software's response time* to a user query and the *understandability* of the program code.

Product Characteristic	Description
Maintainability	o Software may evolve to meet the changing needs of customers. o Critical attribute • software change is an inevitable consequence of a changing business environment.
Dependability	o Range of characteristics, including reliability, security and safety. o Dependable software should not cause physical or economic damage in the event of system failure.
Efficiency	o Avoiding wasteful use of system resources such as memory and processor cycles. o Includes responsiveness, processing time, memory utilization etc.
Acceptability	o Software must be acceptable to the type of users for which it is designed. o This means that it must be understandable, usable, and compatible with other systems that they use. o Requirement: appropriate user interface and adequate documentation.

Key challenges facing Software Engineering

1. **The heterogeneity challenge**
   - Increasingly, systems are required to operate as distributed systems.
   - It is often necessary to integrate new software with older legacy systems.
   - It is the challenge of developing techniques for building dependable software to cope with this heterogeneity.
2. **The delivery challenge**
   - Businesses today must be responsive and change very rapidly.

- Their supporting software must change equally rapidly.
- It is the challenge of shortening delivery times for large and complex systems without compromising system quality.

3. The trust challenge

- It is essential that we can trust our software.
- Especially true for remote software systems accessed through a web page or web service interface.
- The trust challenge is to develop techniques that demonstrate that software can be trusted by its users.

Software Application Domains

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

1. **System software** — a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) are common. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data. In either case, the systems software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces.

2. **Application software**—stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making. In addition to conventional data processing applications, application software is used to control business functions in real time (e.g., point-of-sale transaction processing, real-time manufacturing process control).

3. **Engineering/scientific software**—has been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

4. **Embedded software**—resides within a product or system and is used to implement and control features and functions for the end user and for the system it-self. Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

5. **Product-line software**—designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

6. **Web applications**—called "WebApps," this network-centric software category span a wide array of applications. In their simplest form, WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as Web 2.0 emerges, WebApps are evolving into sophisticated computing environments that not only provide stand-alone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

7. **Artificial intelligence software**—makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

8. **Open-World Computing**—the rapid growth of wireless networking may soon lead to true pervasive, distributed computing. The challenge for software engineers will be to develop systems and application software that will allow mobile devices, personal computers, and enterprise systems to communicate across vast networks.
    a. *Netsourcing*—the World Wide Web is rapidly becoming a computing engine as well as a content provider. The challenge for software engineers is to architect simple (e.g., personal financial planning) and sophisticated applications that provide a benefit to targeted end-user markets worldwide.
    b. *Open source*—a growing trend that results in distribution of source code for systems applications (e.g., operating systems, database, and development environments) so that many people can contribute to its development. The challenge for software engineers is to build source code that is self-descriptive, but more importantly, to develop techniques that will enable both customers and developers to know what changes have been made and how those changes manifest themselves within the software.

*****