Software Crisis

It was in late 1960’s
• Many software projects failed.
  • Many software projects late, over budget, providing unreliable software that is expensive to maintain.
  • Many software projects produced software which did not satisfy the requirements of the customer.
• Complexities of software projects increased as hardware capability increased.
• Larger software system is more difficult and expensive to maintain.
• Demand of new software increased faster than ability to generate new software.

All the above attributes of what was called a ‘Software Crisis’. So the term ‘Software Engineering’ first introduced at a conference in late 1960’s to discuss the software crisis.
Why software engineering?

Once the need for software engineering was identified and software engineering recognized as a discipline--

• The late 1970’s saw the widespread evolution of software engineering principles.
• The 1980’s saw the automation of software engineering and growth of CASE (Computer Aided Software Engineering).
• The 1990’s have seen increased emphasis on the ‘management’ aspects of projects and the use of standard quality and ‘process’ models like ISO 9001 and the Software Engineering Institute’s Software Capability Maturity Model (CMM).

These models help organizations put their software development and management processes in place
What is software Engineering?

• In 1969 Fritz Bauer defined software eng. as, ‘the establishment and use of sound engineering principles in order to obtain, economically, software that is reliable and works efficiently on real machines’.

• According to Boehm, software engineering involves, ‘the practical application of scientific knowledge to the design and construction of computer programs and the associated documentation required developing, operating and maintaining them’

• IEEE, in its standard 610.12-1990, defines software engineering as:
  (i) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
  (ii) The study of approaches as in (i).

• By combining all the above definition we can define software engineering as, ‘Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates.’
Goal of software engineering

The primary goals of software engineering are:

• To improve the quality of the software products.
• To increase the productivity &
• To give job satisfaction to the software engineers.
Foundation of Software engineering

Software engineering is a technological discipline distinct from, but based on the foundation of the following disciplines:

- Computer Science
- Management Science
- Economics
- System Engineering &
- Communication Skills
The relationship of software engineering with other disciplines

- **Computer Science** gives the scientific foundation to the software as electrical engineering relies on physics.
- **Management Science** provides the foundation for software project management. Since software engineering is labor intensive activity, it requires both technical and managerial control.
- **Economics** provides the foundation for resource estimation and cost control. Since, computing system must be developed and maintained on time and within cost estimates; thus economics plays an important role.
- **System Engineering** is the field concerned with studying complex systems. Software is often a component of a much larger system. For example, the software in a factory monitoring system or the flight software on an airplane; is just the component of more complex system. System engineering techniques can be applied to study of such systems.
- Good oral, written and interpersonal **communication skills** are crucial for the software engineers, because software engineering activities occur within an organizational context, and a high degree of communication is required among customers, managers, software engineers, hardware engineers and other technical workers.
Difference of s/w eng. with traditional engineering

• Software is intangible. It has no mass, no volume, no color, no odor--- no physical properties. Source code is merely a static image of computer program, and while the effects produced by a program are often observable, the program itself not.

• Software doesn’t degrade with time as hardware does. Software failures are caused by design and implementation error, not by degradation over time.

• There is always an obscurity in the interface between software modules. It is difficult to design a software system so that all the control and interfaces among modules are explicit, and so that the modules do not interact to produce unexpected side effects when they invoked one another.
In classical engineering disciplines, the engineer is equipped with tools and the mathematical maturity to specify the properties of the product separately from those of design. The typical software engineering relies much more on experience and judgment rather than mathematical formula. While experience and judgment are necessary, formal analysis are also essential in the practice of engineering.
The role of software engineer

The evolution of software engineering field has defined the role the software engineer. A software engineer should have the following qualities:

• Should be a good programmer, be well-versed in data structures and algorithms, and be fluent in one or more programming languages.

• Should be familiar with several design approaches, be able to translate vague requirements and desires into precise specifications and be able to converse with the use of a system in terms of applications.

• Needs the ability to move among several levels of abstraction at different stages of the project, from specific application procedures and requirements, to abstraction for software systems, to a specific design for system and finally to the detailed coding level.
The characteristics of software engineer

• Should be able to build and use a model of the application to guide choices of the many trade-offs that he or she will face. The model is used to answer questions about both the behavior of the system and its performance.

• Needs communication skills and interpersonal skills. He also needs the ability to schedule work both of his own and that of others.
What is well engineered software?

If the software system does what the user wants, and can be made to continue to do what the user wants, it is well engineered.

• Any well engineered software system should have the following attributes:
  • Be easy to maintain
  • Be reliable
  • Be efficient
  • Provides an appropriate user interface

The development of software must make trade-offs between these attributes.
Distribution of software effort

The typical life-span for a typical software product is 1 to 3 years in development and 5 to 15 years in use. The distribution of effort between development and maintenance has been variously reported depending on the type of software as 40/60, 30/70 and 10/90.

Maintenance:

• **Corrective**: Even with the best quality of software, it is likely that customer will uncover defect in software. Corrective maintenance changes the software to correct the defects.

• **Adaptive**: Over time, the original environment (CPU, OS, business rules, external product character etc.) for which the software was developed may change. Adaptive maintenance results in modification to the software to accommodate the change to its environment.

• **Perfective**: As the software is used, the customer / user will recognize additional function that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.
The software product

The objective of software engineering is to produce software products. Computer software is the product that software engineers design and built. software products are software systems delivered to a customer with the documentation which describes how to install and use the system.

- Software products fall into two broad classes:
  - **Generic products**: These are stand alone systems which are produced by a software development organizations/firms and sold on the open market to any customer who is able to buy them.
  - **Customized products**: These are systems which are commissioned by a particular customer. The software is developed specially for that customer by some developer.
.... Software products

- Until the 1980’s, the vast majority of software systems which were sold that were customized and specially designed systems which run on large computers. They are expensive because all the development cost had to be met by a single client.

- After the development of PCs, this situation has completely changed. The PC market is totally dominated by software products produces by companies such as Microsoft. These account for the vast majority of software sales. These are usually relatively cheap because their development cost is spread across hundred or thousands of different customers.
Difference between generic and customized software

• The generic software product specifications are produced internally by the marketing department of the product company. They reflect what they think will sell. They are usually flexible and non-prescriptive.

• For customized systems are often the basis for the contract between customer and developer. They are usually defined in detail and changes have to be negotiated and carefully costed.
Software product attributes

The attributes of a software product are the characteristics displayed by the product, once it is installed and put in use. They are not the services provided by the product. Rather, they are concerned with the products dynamic behavior and the use made of the product. Examples of these attributes are therefore, efficiency, reliability, maintainability, robustness, portability and so on. The relative importance of these characteristics obviously varies from system to system.
... product attributes

<table>
<thead>
<tr>
<th>Product characteristics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintainability</td>
<td>It should be possible to evolve software to meet the changing needs of the customer.</td>
</tr>
<tr>
<td>Dependability</td>
<td>Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure.</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Software should not make wasteful use of system resources such as memory and processor cycle</td>
</tr>
<tr>
<td>Usability</td>
<td>Software should have an appropriate user interface and documentation</td>
</tr>
</tbody>
</table>
..product attributes

Optimizing the above attribute is difficult as some are exclusive. For example, providing a better user interface may reduce system efficiency. The relationship between cost and improvement in each of the attribute is not linear one. Small improvement in any of these attributes may be devoted to optimizing particular attribute.