Unit 1 - Chapter 1
Introduction to Software Engineering

Course Outcomes

After completion of the course, students shall be able to

- Understand Software Development Life Cycle (SDLC) models and the importance of *engineering software*.
- Creation of efficient Software Requirement Specification (SRS) documents to obtain the desired software product.
- Compare various software development methodologies and their applicability in developing a specific type of software product.
- Construct an efficient design specification document to obtain the desired product per the user requirements.
- Be able to develop software applications using the concepts applicable to various phases of the software development life cycle.
- Study various software testing techniques and identify their relevance in developing quality software.
- Recognize software maintenance as a major activity in SDLC
- Examine the various tools used in all phases of Software construction
- Understand concepts of software project management and quality assurance

Pre-requisites

- Knowledge of one or more programming language
- Experience in the implementation of at least one software project/mini project

Outcomes

- Examine concepts required to understand Software Engineering
- Origin of and the need for Software Engineering

1.0 Introduction

The modern world is unimaginable without software. Software has its presence across various industries. Software is pervasive in manufacturing, utilities, distribution, financial, education, and entertainment among numerous sectors. Software systems are abstract and lack physical constraints. Consequently, software systems can become extremely complex to create, difficult to understand, and expensive to change. The diversity of software applications is extremely diverse. It is still an extremely challenging and complex endeavour to develop and maintain software applications.

Software systems are getting larger, more complex, the demands change. Systems have to be built and delivered more quickly; systems have to have new capabilities that were previously thought to be impossible. In many places ad-hoc approach to software development is prevalent. Therefore software tends to be unreliable, expensive to maintain and use. As a consequence, an Engineering approach to develop and manage software is required. This entails education and training in software engineering methods. Thanks to software engineering achievements like space travel and the Internet, eCommerce, banking and other services we take for granted today are possible.

In the subsequent sections, we shall try to explore the following topics

- Meaning of Software and Engineering
- What led to the Software Crisis and the birth of Software Engineering?
- Why Study Software Engineering?
- Essential attributes of good software
- Software Myths

1.1 Terminology

Software comprises of

- Instructions (computer programs) that when executed provide desired features, function, and performance;
- Data structures that enable the programs to adequately store and manipulate information and
- Documentation that describes the operation and use of the programs.

Thus, Software may be defined as a collection of computer programs, data structures and associated documentation. Software products may be developed for a particular customer (*custom/bespoke*) or may be developed for a general market (*off-the-shelf/generic*).

Engineering

Let us examine definitions of Engineering as applicable to the disciplines of civil or mechanical engineering.

- the study of using scientific principles to design and build machines, structures, and other things, including bridges, roads, vehicles, and buildings: (*American Dictionary*)
- the art or science of making practical application of the knowledge of pure sciences, as physics or chemistry, as in the construction of engines, bridges, buildings, mines, ships, and chemical plants. (*dictionary.com*)

IEEE fully defines software engineering as:

- *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*

Engineering definitions as applied to:

Mechanical engineering involves the design, manufacturing, inspection and maintenance of machinery, equipment and components as well as control systems and instruments for monitoring their status and performance. This includes vehicles, construction and farm machinery, industrial installations and a wide variety of tools and devices.

Civil engineering involves the design, construction, maintenance and inspection of large infrastructure projects such as highways, railroads, bridges, tunnels, dams and airports.

Software Engineering

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically, software that is reliable and works efficiently on real machines.

The 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).

Engineering is about getting results of the required quality within the schedule and budget. In general, software engineers adopt a systematic and organized approach to their work. It involves an Engineering approach to developing software. Thereby Software Engineering is a disciplined and systematic approach whose aim is to develop quality software, software that is delivered on time, within budget, and that satisfies its requirements. It relies on past experiences for techniques, methodologies, guidelines.

Software engineering is an engineering discipline that is concerned with all aspects of software production/construction from the early stages of system specification through to maintaining the system after it has gone into use. Engineers apply theories, methods, and tools where these are appropriate. However, they always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints so they look for solutions within these constraints. Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software production &maintenance.

1.2 Why Study Software Engineering?

During your under-graduation academic journey, you are unable to appreciate the importance of software engineering primarily due to the size of projects/code being undertaken by you. Once you move on to implement large projects you shall appreciate the importance of software engineering. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are indispensable to achieve good quality software cost-effectively.

Try to answer the questions below

- How would you construct a small bridge to cross a small open drain flowing in front of your home?
- How would you go about building a bridge across river Ganga in Haridwar?
- How would you build a shelter for your pet dog?
- How would you build a Home for a large human Family of 20 members?
- Program to calculate the area of a triangle versus the problem of automating your College Operations

There are numerous reasons for the study of Software Engineering as a discipline. A few important ones are listed below.

- The ad hoc approach breaks down when the size of software increases.
- To acquire skills to develop large programs and work in large real-life projects.
- The exponential growth in complexity and difficulty level increases with code and project size.
- Develop the ability to solve and manage complex programming problems

- Understand how to break large projects into smaller and manageable parts

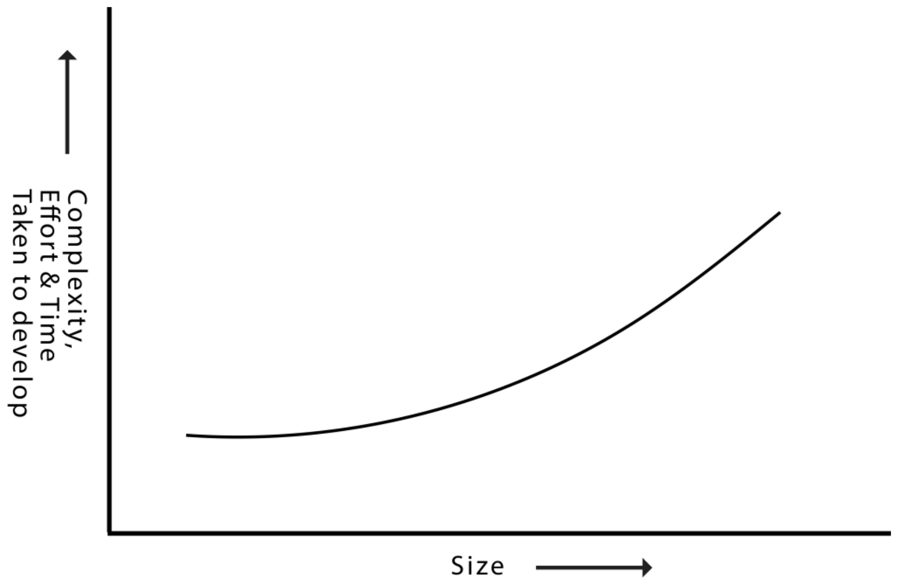

Fig 1.1.1: Increase in development time and effort with problem size

Without using software engineering principles it would be difficult to develop large programs. In industry, it is usually needed to develop large programs to accommodate multiple functions. A problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes as shown in fig. 1.1.1. For example, a program of size 1,000 lines of code has some complexity. But a program with 10,000 LOC is not just 10 times more difficult to develop, but may well turn out to be 100 times more difficult unless software engineering principles are used. In such situations, software engineering techniques come to the rescue. Software engineering helps to reduce programming complexity.

1.3 Origin of Software Engineering (courtesy the Software Crisis)

The term 'software engineering' was first proposed in 1968 at a conference held to discuss what was then called the software crisis. It became clear that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late. Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding and object-oriented development. Tools and standard notations were developed and are now extensively used.

1.3.1 Causes of and solutions for software crisis

Software engineering appears to be among the few options available to tackle the present software crisis. To explain the present software crisis in simple words, consider the following. The expenses that organizations all around the world are incurring on software purchases compared to those on hardware purchases have been showing a worrying trend over the years (as shown in fig. 1.1.2).

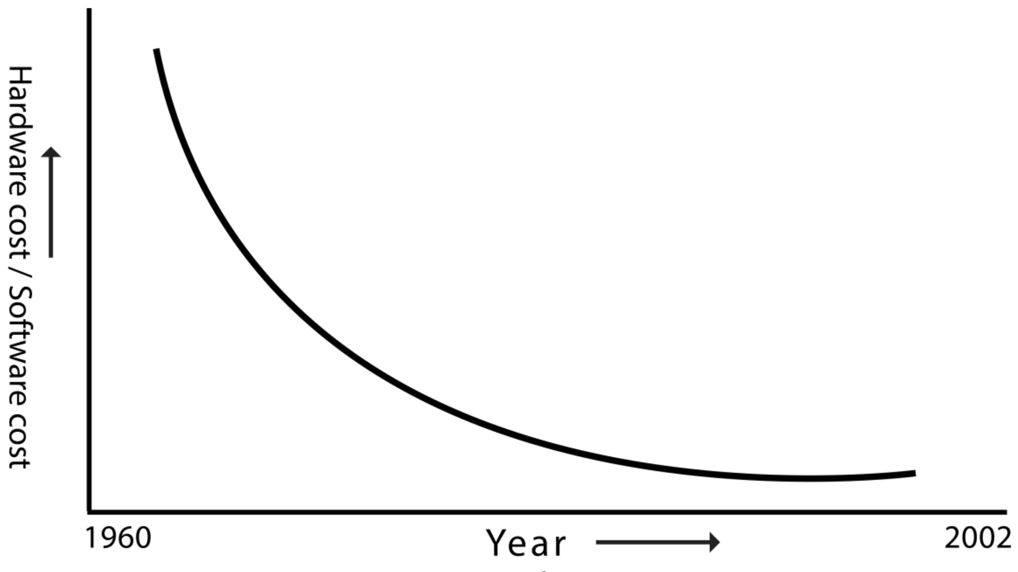

Fig. 1.1.2: Change in the relative cost of hardware and software over time

Organizations are spending larger and larger portions of their budget on software. Not only are the software products turning out to be more expensive than hardware, but they also present a host of other problems to the customers: software products are difficult to alter, debug, and enhance; use resources non-optimally; often fail to meet the user requirements; are far from being reliable; frequently crash, and are often delivered late. Among these, the trend of increasing software costs is probably the most important symptom of the present software crisis. Remember that the cost we are talking of here is not on account of increased features, but due to ineffective development of the product characterized by inefficient resource usage, and time and cost over-runs. Many factors have contributed to the making of the present software crisis. Factors are larger problem sizes, lack of adequate training in software engineering, increasing skill shortage, and low productivity improvements.

It is believed that the only satisfactory solution to the present software crisis can come from a spread of software engineering practices among the engineers, coupled with further advancements to the software engineering discipline itself.

1.3.2 Technology Development Pattern (Evolution)

During the early stages of software development, there were not many past experiences to fall back on. As a consequence, the implementation of software projects was based on intuition, gut feel and was more art. As several software projects began to be implemented certain past experiences were used but in an unorganized manner. Once a substantial amount of projects were implemented a codified body of knowledge emerged. This led to the use of a systematic approach to software development using scientific principles and past experiences in an organized manner.

1.4 Essential attributes of good software

Attribute	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of

	a changing business environment.
Dependability and security	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. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency, therefore, includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

1.5 Frequently Asked Questions (FAQ)

Question	Answer
What is software?	Computer programs plus associated documentation.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software	Software engineering is an engineering

engineering?	discipline that is concerned with all aspects of software production.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What are the key challenges facing software engineering?	Coping with increasing diversity demands, reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.

1.6 Software Myths in Software Engineering

Roger Pressman (1997) describes several common beliefs or myths that software managers, customers, and developers believe falsely. He describes these myths as “misleading attitudes that have caused serious problems.” The types of Software myths are

- Managers Myth
- Developers Myth
- Customers Myth

MANAGEMENTS MYTH

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”.

MYTH: My people have state-of-the-art software development tools; after all, we buy them the newest computers.
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

MYTH: If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept).
REALITY: Software development is not a mechanical process like manufacturing, adding people to a late software project makes it later

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.

DEVELOPER’S MYTH

MYTH: Once we write the program and get it to work, our job is done.
REALITY: Someone once said that “the sooner you begin ‘writing code’, the longer it’ll take you to get done. Industry data indicate that between 60 and 80 per cent of all effort expended on software will be expended after it is delivered to the customer for the first time.

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 is more effective than testing for finding certain classes of software defects.

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 many elements. Documentation provides a foundation for successful engineering and, more importantly, guidance for software support.

MYTH: Software engineering will make us 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 aster delivery times.

CUSTOMER’S MYTH

MYTH: A general statement of objectives is sufficient to begin writing programs— we can fill in the details later.
REALITY: A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customers and developers.

MYTH: Project requirements continually change, but change can be easily accommodated because the software is flexible.
REALITY: Software requirements indeed change, but the impact of change varies with the time at which it is introduced.

1.7 Conclusion

Progress in software engineering has been remarkable over time. The modern world cannot function without large, professional software systems that support the development and deployment of large enterprise applications. It is hard to think of areas not requiring software applications.

More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly. It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.

Software engineering is, therefore, a critically important technology for the future of mankind. We must continue to educate software engineers and develop the discipline so that we can create more complex software systems.

Additional Information and MCQ

1 It is strongly recommended to read the paper by Prof. Nicklaus Wirth hosted online. You learn a lot from history. It is easy to take things for granted today. You shall develop an appreciation for the evolution of computers from the 1950s till date.

2 Wikipedia Reference (Brief History of Software Engineering)
https://en.wikipedia.org/wiki/History_of_software_engineering

3 Additional Information on evolution of computing and software
https://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering

***