Topics:

- Software Engineering Importance
- Software Engineering Process
- Software Engineering Pertinence
- Software Engineering Crisis
- Quality of professional Software
- Software engineering Diversity

Software engineering is important for two reasons:

1. Needing to produce reliable and trustworthy systems. Thus, individuals and society rely on advanced software systems.
2. 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.

Software process

The systematic approach that is used in software engineering is sometimes called a **software process. A software process** is a sequence of activities that leads to the production of a software product. There are four fundamental activities that are common to all software processes. **These activities are:**

1. **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. **Software design and implementation**, where the software is designed and programmed.
3. **Software validation**, where the software is checked to ensure that it is what the customer requires.
4. **Software evolution**, where the software is modified to reflect changing customer and market requirements.

Different types of systems need different development processes. For example, real-time software in an aircraft has to be completely specified before development begins. In e-commerce systems, the specification and the program are usually developed together. Consequently, these generic activities may be organized in different ways and described at different levels of detail depending on the type of software being developed.

- **Software Engineering Pertinence**

  Software engineering is related to both computer science and systems engineering:

  - **Computer science** is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. However, computer science theory is often most applicable to relatively small programs. Elegant (refined and perfect) theories of computer science cannot always be applied to large, complex problems that require a software solution.
  - **System engineering** is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design and system deployment, as well as software engineering. System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system.

There are many different types of software. There is no universal software engineering method or technique that is applicable for all of these. However, **there are three general issues that affect many different types of software:**

1. **Heterogeneity increasingly**: systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. The developer often has to integrate new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to deal with this heterogeneity.
2. **Business and social change**: Business and society are changing quickly. They need to be able to change their existing software and to rapidly develop new software. Many traditional software engineering techniques are time consuming and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.
3. **Security and trust**: The remote software systems accessed through a web page or web service interface must be protected. This lead to make sure that malicious user cannot attack our software and that information security is maintained.

Software: A crisis on the horizon

Whether we call it a software crisis or affliction, the term alludes to a set of problems that are encountered in the development of computer software. The problems are not limited to software that "doesn't function properly". Rather, the affliction encompasses problems associated with how we develop software,

how we support a growing volume of existing software, and how we can expect to keep pace with a growing demand for more software.

- **Quality of professional software**

The software is used and changed by people apart from (in addition to) its developers. Quality is therefore not just concerned with what the software does. Rather, it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation. This is reflected in so called quality or non-functional software attributes. Examples of these attributes are the software's response time to a user query and the understandability of the program code.

The specific set of attributes that software engineer/developer might expect from a software system obviously depends on its application. Therefore, a banking system must be secure, an interactive game must be responsive, and a telephone switching system must be reliable, and so on.

The essential attributes of good software are:

1. **Maintainability:** Software should be written in such a way so that it can evolve to meet the changing needs of customers.
2. **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.
3. **Efficiency:** Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness,

processing time, memory utilization, etc.

4. **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.

   Engineering is about getting results of the required quality within the schedule and budget. This often involves making compromises; engineers cannot be perfectionists. In general, software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software. People writing programs for themselves, however, can spend as much time as they wish on the program development.

- **Software engineering diversity**

Software engineering is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.

The most significant factor in determining which software engineering methods and techniques are most important is the type of application that is being developed. **There are many different types of application including:**

1. **Stand-alone applications**: These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network. Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.
2. **Interactive transaction-based applications**: These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals. Obviously, these include web applications such as e-commerce applications where the person can interact with a remote system to buy goods and services. This class of application also includes business systems, where a business provides access to its systems through a web

browser or special-purpose client program, such as mail and photo sharing. Interactive applications often incorporate a large data store that is accessed and updated in each transaction.

3. **Embedded control systems**: These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system. Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car, and software in a microwave oven to control the cooking process.
4. **Batch processing systems**: These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs. Examples of batch systems include periodic billing systems, such as phone billing systems, and salary payment systems.
5. **Entertainment systems**: These are systems that are primarily (in the first place/mainly) for personal use and which are intended to entertain the user. Most of these systems are games. The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.
6. **Systems for modeling and simulation**: These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects. These are often computationally intensive and require high-performance parallel systems for execution.
7. **Data collection systems**: These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing. The software has to interact with sensors and often is installed in a hostile (unfriendly) environment such as inside an engine or in a remote location.
8. **Systems of systems**: These are systems that are composed of a number of other

software systems. Some of these may be generic software products, such as a spreadsheet program. Other systems in the assembly may be specially written for that environment.

There are software engineering fundamentals that apply to all types of software system:

1. The software should be developed using a managed and understood development process.
2. Dependability and performance are important for all types of systems.
3. Understanding and managing the software specification and requirements (what the software should do) are important.
4. The software engineer should make as effective use as possible of existing resources.

The software engineer should notice that these fundamentals do not cover implementation and programming.