The Joint Task Force on Computing Curricula
IEEE Computer Society
Association for Computing Machinery

The Discipline of Software Engineering

Since the dawn of computing in the 1940s, the applications and uses of computers have grown at a staggering rate. Software plays a central role in almost all aspects of daily life: in government, banking and finance, education, transportation, entertainment, medicine, agriculture, and law.

The number, size, and application domains of computer programs have grown dramatically; as a result, hundreds of billions are being spent on software development, and the livelihood and lives of most people depend on the effectiveness of this development. Software products have helped us to be more efficient and productive. They make us more effective problem solvers, and they provide us with an environment for work and play that is often safer, more flexible, and less confining. Despite these successes, there are serious problems in the cost, timeliness, and quality of many software products. The reasons for these problems are many and include the following:

- Software products are among the most complex of man-made systems, and software by its very nature has intrinsic, essential properties (e.g., complexity, invisibility, and changeability) that are not easily addressed [Brooks 95].

- Programming techniques and processes that worked effectively for an individual or a small team to develop modest-sized programs do not scale-up well to the development of large, complex systems (i.e., systems with millions of lines of code, requiring years of work, by hundreds of software developers).

- The pace of change in computer and software technology drives the demand for new and evolved software products. This situation has created customer expectations and competitive forces that strain our ability to produce quality of software within acceptable development schedules.

It has been over thirty-five years since the first organized, formal discussion of software engineering as a discipline took place at the 1968 NATO Conference on Software Engineering [Naur 1969]. The term “software engineering” is now widely used in industry, government, and academia: hundreds of thousands of computing professionals go by the title “software engineer”; numerous publications, groups and organizations, and professional conferences use the term software engineering in their names; and there are many educational courses and programs on software engineering. However, there are still disagreements and differences of opinion about the meaning of the term. The following definitions provide several views of the meaning and nature of software engineering. Nevertheless, they all possess a common thread, which states, or strongly implies that software engineering is more than just coding - it includes quality, schedule and economics, and the knowledge and application of principles and discipline.

Definitions of Software Engineering

Over the years, numerous definitions of the discipline of Software Engineering have been presented. For the purpose of this document, we highlight the following definitions:
• "The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines" [Bauer 1972].
• "Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems." [CMU/SEI-90-TR-003]

• "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [IEEE 1990].

There are aspects of each of these definitions that contribute to the perspective of software engineering used in the construction of this volume. One particularly important aspect is that software engineering builds on computer science and mathematics. But, in the engineering tradition, it goes beyond this technical basis to draw upon a broader range of disciplines.

These definitions clearly state that software engineering is about creating high-quality software in a systematic, controlled, and efficient manner. Consequently, there are important emphases on analysis and evaluation, specification, design, and evolution of software. In addition, there are issues related to management and quality, to novelty and creativity, to standards, to individual skills, and to teamwork and professional practice that play a vital role in software engineering.

**Software Engineering as a Computing Discipline**

A common misconception about software engineering is that it is primarily about process-oriented activities (i.e., requirements, design, quality assurance, process improvement, and project management). In this view, competency in software engineering can be achieved by acquiring a strong engineering background, a familiarity with a software development process and a minimal computing background, including experience using one or more programming languages. Such a background is, in fact, quite insufficient; the misconception that leads to Duch thinking is based on an incomplete view of the nature and challenges of software engineering.

In the historical development of computing, computer scientists produced software and electrical engineers produced the hardware on which the software runs. As the size, complexity, and critical importance of software grew, so did the need to ensure that software performs as intended. By the early 1970’s, it was apparent that proper software development practices required more than just the underlying principles of computer science; they need both the analytical and descriptive tools developed within computer science and the rigor that the engineering disciplines bring to the reliability and trustworthiness of the artifacts they engineer.

Software engineering thus is different in character from other engineering disciplines, due to both the intangible nature of software and to the discrete nature of software operation. It seeks to integrate the principles of mathematics and computer science with the engineering practices developed to produce tangible, physical artifacts. Drawing on computing and mathematics as foundations, software engineering seeks to develop systematic models and reliable techniques for producing high-quality software; and these concerns extend all the way from theory and principles to the development practices that are most visible to those outside of the discipline. While it is not expected that every software engineer will have deep expertise in all of aspects of computing, a general understanding of their relevance and some expertise in particular aspects are a necessity.
Software Engineering as an Engineering Discipline

The study and practice of software engineering is influenced both by its roots in computer science and its emergence as an engineering discipline. A significant amount of current software engineering research is conducted within the context of computer science and computing departments or colleges. Similarly, software engineering degree programs are being developed by such academic units as well as within engineering colleges. Thus, the discipline of software engineering can be seen as an engineering field with a stronger connection to its underlying computer science discipline than the more traditional engineering fields. In the process of constructing this volume, particular attention has been paid to incorporating the practices of engineering into the development of software, so as to distinguish this curriculum from computer science curricula. To prepare for the more detailed development of these ideas, this section examines the engineering methodology and how it applies to software development.

We must also point out that although there are strong similarities between software engineering and more traditional engineering, there are also some differences (not necessarily to the detriment of software engineering):

- Foundations are primarily in computer science, not in natural sciences.
- The focus is on discrete rather than continuous mathematics.
- The concentration is on abstract/logical entities instead of concrete/physical artifacts.
- There is no “manufacturing” phase in the traditional sense.
- Software “maintenance” primarily refers to continued development, or evolution, and not to conventional wear and tear.

Characteristics of Engineering

There is a set of characteristics that is not only common to every engineering discipline, but is so predominant and critical that they can be used to describe the underpinnings of engineering. It is these underpinnings that should be viewed as desirable characteristics of software engineers. Thus they have influenced the development of software engineering.

[1] Engineers proceed by making a series of decisions, carefully evaluating options, and choosing an approach at each decision-point that is appropriate for the current task in the current context. Appropriateness can be judged by tradeoff analysis, which balances costs against benefits.

[2] Engineers measure things, and when appropriate, work quantitatively; they calibrate and validate their measurements; and they use approximations based on experience and empirical data.

[3] Engineers emphasize the use of a disciplined process when creating a design and can operate effectively as part of a team in doing so.

[4] Engineers can have multiple roles: research, development, design, production, testing, construction, operations, management, and others such as sales, consulting, and teaching.
[5] Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is key to engineering.

[6] Engineers, via their professional societies, advance by the development and validation of principles, standards, and best practices.

[7] Engineers reuse designs and design artifacts.

It should be noted that while the term engineer and engineering will be used extensively in the following sections, this document is about the design, development and implementation of undergraduate software engineering curricula. It must be acknowledged that much of the work in this document is based on the work of numerous individuals and groups that have advanced the state of computer science and information technology, and have developed programs that help prepare graduates to practice software development in a professional manner.

**Engineering design**

Design is central to any engineering activity, and it plays a critical role in regard to software. In general, engineering design activities refer to the definition of a new artifact by finding technical solutions to specific practical issues, while taking into account economic, legal, and social considerations. As such, engineering design provides the prerequisites for the "physical" realization of a solution, by following a systematic process, that best satisfies a set of requirements within potentially conflicting constraints.

Software engineering differs from traditional engineering because of the special nature of software, which places a greater emphasis on abstraction, modeling, information organization and representation, and the management of change. Software engineering also includes implementation and quality control activities normally considered in the manufacturing process design and manufacturing steps of the product cycle. Furthermore, continued evolution (i.e., “maintenance”) is also of more critical importance for software. Even with this broader scope, however, a central challenge of software engineering is still the kind of decision-making known as engineering design. An important aspect of this challenge is that the supporting process must be applied at multiple levels of abstraction. An increasing emphasis on reuse and component-based development hold hope for new, improved practices in this area.

**Domain-specific software engineering**

Within a specific domain, an engineer relies on specific education and experience to evaluate possible solutions, keeping in mind various factors relating to function, cost, performance and manufacturability. Engineers have to determine which standard parts can be used and which parts have to be developed from scratch. To make the necessary decisions, they must have a fundamental knowledge of the domain and related specialty subjects.

Domain-specific techniques, tools, and components typically provide the most compelling software engineering success stories. Great leverage has been achieved in well-understood domains where standard implementation approaches have been widely adopted. To be well prepared for professional practice, graduates of software engineering programs should come to terms with the fundamentals of at least one application domain. That is, they should understand the problem space that defines the domain as well as common approaches, including Standard components (if any), used in producing software to solve problems in that domain.