What Is Software Engineering?

http://www.sei.cmu.edu/about/overview/whatis.html

The establishment and use of sound engineering principles (methods) in order to obtain economically software that is reliable and works on real machines (Bauer, F. L. *Software Engineering.* Information Processing 71., 1972).

> Mr. Bauer was a principal organizer of the 1968 NATO conference that led to the widespread use of the term "software engineering."

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) IEEE Std 610-1990.
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 (Fairley, R. Software Engineering Concepts. New York: McGraw-Hill, 1985).
Software engineering is the computer science discipline concerned with developing large applications. Software engineering covers not only the technical aspects of building software systems, but also management issues, such as directing programming teams, scheduling, and budgeting (WebReference Webopaedia).
SEI software engineering definition from 1990 *SEI Report on Undergraduate Software Engineering Education* (CMU/SEI-90-TR-003):

- Engineering is the systematic application of scientific knowledge in creating and building cost-effective solutions to practical problems in the service of mankind.
- Software engineering is that form of engineering that applies the principles of computer science and mathematics to achieving cost-effective solutions to software problems.

For additional information about software engineering, see the *SEI Report about Undergraduate Software Engineering Education* CMU/SEI-90-TR-003 , IEEE Std 610-1990 or WebReference Webopaedia.

Software Engineering:
The computer science discipline concerned with developing large applications. Software engineering covers not only the technical aspects of building software systems, but also management issues, such as directing programming teams, scheduling, and budgeting.

Software engineering

From Wikipedia, the free encyclopedia.

Software engineering (SE) is the profession concerned with specifying, designing, developing and maintaining software applications by applying technologies and practices from computer science, project management, and other fields.

SE applications are used in a wide range of activities, from industry to entertainment. Software applications improve user productivity and quality of life. Application software examples: office suites, video games, and the world wide web. System software examples: embedded systems and operating systems.

SE technologies and practices improve the productivity of developers and the quality of the applications they create. Software engineering examples: databases, languages, libraries, patterns, and tools. Computer science examples: algorithms and data structures. Project management examples: processes.

The SE community includes 630,000 practitioners and educators in the U.S. and an estimated 1,400,000 practitioners in the E.U., Asia, and elsewhere; it is about 60% the size of traditional engineering. SE pioneers include Barry Boehm, Fred Brooks, C. A. R. Hoare, and David Parnas.

Related terms: software engineer.

See also List of software engineering topics.

Contents [hide]

[edit]

Terminology

[edit]

Origins

The term *software engineering* was used occasionally in the late 1950s and early 1960s. Software engineering was popularized by the 1968 NATO Software Engineering Conference held in Garmisch, Germany and has been in widespread use since.

[edit]

Meanings

As of 2004, in common parlance the term *software engineering* is used with at least three distinct meanings:

- As the usual contemporary term for the broad range of activities that was formerly called programming or systems analysis;
- As the broad term for the technical analysis of all aspects of the *practice,* as opposed to the *theory* of computer programming;
- As the term embodying the *advocacy* of a specific approach to computer programming, one that urges that it be treated as an engineering profession rather than an art or a craft, and advocates the codification of recommended practices in the form of *software engineering methodologies*.

[edit]

Levels

There are currently no widely accepted criteria for distinguishing someone who is a software engineer from someone who is not a software engineer. In addition, the industry is in the midst of a complex debate on the licensing of practicing software engineers. For the localities that do not license software engineers, some hiring classifications are made based on education and experience. Classification levels may include: entry-level, mid-level, and senior.
Typical entry-level software engineers have a bachelors degree and zero to five years of experience. Typical mid-level software engineers have a bachelors or masters degree and have five to ten years of experience. Typical senior-level software engineers have an advanced degree and have ten or more years of

experience. Note that these are only guidelines that are trends seen in hiring practices and that many exceptions exist.

[edit]

Software engineer

Software engineering is the practice of creating software.

Members of this profession are called software engineers, programmers, developers, or practitioners.

People who write code and do not follow the doctrines of software engineering are more accurately called programmers, developers, or software artists.

[edit]

Software engineering today

[edit]

Impact of software engineering

Software engineering affects economies and societies in many ways.

Economic

In the U.S., software drove about 1/4 of all increase in GDP during the 1990s (about $90 billion per year), and 1/6 of all productivity growth (efficiency within GDP) during the late 1990s (about $33 billion per year). Software engineering drove $1 trillion of economic and productivity growth over the last decade.

Social

Software engineering changes world culture, wherever people use computers. Email, the world-wide web, and instant messaging enable people to interact in new ways. Software lowers the cost and improves the quality of health-care, fire departments, and other important social services.

Successful projects where software engineering methods have been applied include Linux, the space shuttle software, and automatic teller machines. When it is cheaper to run a business or agency with software applications than without, businesses and agencies often invest in computers, software, and personnel. See also software engineering economics.

[edit]

Room for improvement

In spite of the enormous economic growth and productivity gains enabled by software there remains a persistent complaint about the quality of software. Since its beginning as a field software engineering has provided great increases in the efficiency of software production, greatly increased complexity and functionality of programs and the implementation of previously impractical operations. There remain great problems in stability, security and utility. On going research in software engineering practices and techniques seeks to address these open concerns.
See also Debates within software engineering and Criticism of software engineering
[edit]

Education

People from many different educational backgrounds make important contributions to SE. The fraction of practitioners who earn computer science or software engineering degrees has been slowly rising. Today, about 1/2 of all software engineers earn computer science or software engineering degrees. For comparison, about 3/4 of all traditional engineers earn engineering degrees.

Software degrees
: About half of all practitioners today have computer science degrees, which are the most relevant degrees that are widely available. A small, but growing, number of practitioners have **software engineering** degrees. As of 2004, in the U.S., about 2,000 universities offer computer science degrees and about 50 universities offer software engineering degrees. Most SE practitioners will earn computer science degrees for decades to come, though someday, this may change.

Domain degrees
: Some practitioners have degrees in application domains, bringing important domain knowledge and experience to projects. In MIS, some practitioners have business degrees. In embedded systems, some practitioners have electrical or computer engineering degrees, because embedded software often requires a detailed understanding of hardware. In medical software, some practitioners have medical informatics, general medical, or biology degrees.

Other degrees

Some practitioners have mathematics, science, engineering, or other technical degrees. Some have philosophy, or other non-technical degrees. And, some have no degrees. Note that Barry Boehm earned degrees in mathematics and Edsger Dijkstra earned degrees in physics.

[edit]

Graduate

Graduate computer science degrees have been available from hundreds of universities for several decades.

Graduate software engineering degrees have been available from dozens of universities for a decade or so.

[edit]

Undergraduate

Undergraduate computer science degrees are available from most universities. In 1996, Rochester Institute of Technology established the first BSSE degree program (*http://www.se.rit.edu*) in the United States. The program received ABET accredidation in 2003. Since then, software engineering undergraduate degrees have been established at many universities. A standard international curriculum for undergraduate software engineering degrees was recently defined by the CCSE.

[edit]

Secondary

Programming and coding are being taught to students at an increasingly earlier stage in secondary schools. However, software engineering is not always included in the curriculum. Many have the impression that students are adequately capable of managing projects. Development techniques beyond learning a programming syntax is required.

[edit]

Tasks

Software engineering requires performing many tasks, some of which do not seem to directly produce software. While some organizations may have specialist to perform some of these tasks there are other cases when a software engineer may be required to do them all.

- **Specification** Extracting the requirements and specifications of a desired software product is the first task in creating it. While customers probably believes they know what the software is to do, it may require skill and experience in software engineering to recognise incomplete, ambiguous or contradictory requirements.
- **Design** Design and architecture refer to determine how software is to function in a general way without being involved in details.
- **Coding** Reducing a design to code may be the most obvious part of the software engineeing job but it is not necessarily the largest portion.
- **Testing** Testing of parts of software, especially where code by two different engineers must work together, falls to the software engineer.
- **Documentation** An important (and too often overlooked) task is documenting the internal design of software for the purpose of future maintenance and enhancement.
- **Maintenance** maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer.

[edit]

Technologies and practices

What is the best way to make more and better software? SEs advocate many different technologies and practices, with much disagreement. This debate has gone on for 60 years and may continue forever. Software engineers use a wide variety of technologies and practices.
[edit]

Technologies

Practitioners use a wide variety of technologies, from compilers to word processors to code repositories.

[edit]

Processes and methodologies

A decades-long goal has been to find processes or methodologies that improve productivity and quality. Some want to systematize or formalize the seemingly-unruly task of writing applications. Others want to apply project management techniques to writing applications. Without project management, software projects can easily be delivered late or over budget. With large numbers of software projects not meeting their expectations in terms of functionality, cost, or delivery schedule, effective project management is proving difficult.

The best-known and oldest process is the waterfall model, where (roughly) developers analyze the problem, design a solution approach, architect a software framework to that solution, develop code, test, deploy, and maintain. The problem is that adequate experience to analyze and specify large systems is almost never available and errors discovered late in the process are expensive to fix. Recent approaches (such as agile) aim to be more flexible and more incremental. These models are more complex and subtler and more realistic. Advocates urge both discipline and pragmatism.

Iterative development prescribes the construction of an initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster.

'[SE advocates] have climbed a social ladder for a few decades and are now fighting against a tide of open source software that seems to be bringing bazaar anarchy and taking the well-deserved control out of their hands. Part of this is their utopia of "software engineering" by some magic cathedral approach which has never worked and whose failure the authors of these utopias tend to blame on the lack of control that copyright offers them over their projects. The strange thing here is that they have had the chance to put all these things into practice in

their university haven. But, strangely enough, the more successful university projects are carried out in a bazaar-like open-source manner.' -- Hartmut Pilch
See also software development processes and methodologies.
[edit]

Roles in industry

Practitioners specialize in many roles in industry (analysts, architects, developers, testers, technical support, managers) and academia (educators, researchers).
In large projects, people may specialize in only 1 role. In small projects, people may fill several or all roles at the same time.
Most software engineers work as employees or contractors. Software engineers work with businesses, government agencies (civilian or military), and non-profit agencies (a school or .org like Wikipedia). Some software engineers work for themselves as free agents.
There is considerable debate over the future employment prospects for Software Engineers and other IT Professionals. For example, an online futures market called the Future of IT Jobs in America (*http://www.ideosphere.com/fx-bin/Claim?claim=ITJOBS*) attempts to answer the question as to whether there will be more IT jobs, including software engineers, in 2012 than there were in 2002.
The IEEE (*http://www.ieee.org*) lead a professional certification for software engineers Certified Software Development Professional CSDP (*http://www.computer.org/certification*).
See also software engineering demographics.
[edit]

Comparing related fields

Many fields are closely related to software engineering. Here are some key similarities and distinctions. Comparing SE with other fields helps explain what

SE is and helps define what SE might or should become. There is considerable debate over which fields SE most resembles (or should most resemble). These complex and inexact comparisons explain why some see software engineering as its own field.

[edit]

What is the nature of SE?

Software engineering resembles many different fields in many different ways. The following paragraphs make some simple comparisons.

Mathematics

Programs have many mathematical properties. For example the correctness and complexity of many algorithms are mathematical concepts that can be rigorously proven. Programs are finite, so in principle, developers could know many things about a program in a mathematical way. This is often called formal methods. However, computability theory shows that not everything useful about a program can be proven. Mathematics works best for small pieces of code and has difficulty scaling up. Edsger Dijkstra has argued that software engineering is a branch of mathematics.

Science

Programs have many scientific properties thaty can be measured. For example, the performance and scalability of programs under various workloads can be measured. The effectiveness of caches, bigger processors, faster networks, newer databases are scientific issues. Mathematical equations can sometimes be deduced from the measurements. Scientific approaches work best for system-wide analysis, but often are meaningless when comparing different small fragments of code.

Engineering

Software Engineering is considered by many to be an engineering discipline because there are pragmatic approaches and expected characteristics of engineers. Proper analysis, documentation, and commented code are signs of an engineer. David Parnas has argued that software engineering is engineering.

Manufacturing

Programs are built in as a sequence of steps. By properly defining and carrying out those steps, much like a manufacturing assembly line, advocates hope to improve the productivity of developers and the quality of final programs. This approach inspires the many different processes and methodologies.

Project Management

Commercial (and many non-commercial) software projects require management. There are budgets and schedules to set. People to hire and lead. Resources (office space, computers) to acquire. All of this fits more appropriately within the purview of management.

Art

Programs contain many artistic elements, akin to writing or painting. User interfaces should be aesthetically pleasing to users. Code should be aesthetically pleasing to programmers. Many goals of good design are NP-complete or worse (such as minimizing the number of lines of code, minimizing number of variables, etc.), meaning they are not decided objectively by either man or computer, so they must be decided by one's own sense of aesthetics. Even the decision of whether a variable name or class name is clear and simple is an artistic question. Donald Knuth famously argued that programming is an art.

Performance

The act of writing software requires that developers summon the energy to find the answers they need while they are at the keyboard. Creating software is a performance that resembles what athletes do on the field, and actors and musicians do on stage. Some argue that SEs need inspiration to spark the creation of code. Sometimes a creative spark is needed to create the architecture or develop a piece of code. Others argue that discipline is the key attribute. Pair programming emphasizes this point of view. Both Kent Beck and Watts Humphrey have argued this emphasis.

[edit]

Branch of which field?

Is SE (or should SE be) a branch of programming, a branch of computer science, a branch of traditional engineering, or a field that stands on its own? There is considerable debate over this. This has important implications for professionalism, licensing, and ethics. Licensing is a polarizing issue: some fiercely advocate it while others staunchly oppose it.

Branch of programming

Programming emphasizes writing code, independent of projects and customers. Software engineering emphasizes writing code in the context of projects and customers by making plans and delivering applications. As a branch of programming, SE would probably have no significant licensing or professionalism issues.

Branch of computer science

Many believe that software engineering is a part of computer science, because of their close historical connections and their relationship to mathematics. They advocate keeping SE a part of computer science. Both computer science and software engineering care about programs. Computer science emphasizes the theoretical, eternal truths while software engineering emphasizes practical, everyday usefulness. Some argue that computer science is to software engineering as physics and chemistry are to traditional engineering. As a branch of computer science, SE would probably have few licensing or professionalism concerns.

Branch of engineering

Others advocate making SE a part of traditional engineering. This is especially true for people who want to emulate other elements of engineering, such as licensing. Both engineering and software engineering share many project management problems and solutions. But, they apply different technologies, they use different kinds of processes, and are driven by different economics. As a branch of engineering, SE would probably adopt the engineering model of licensing and professionalism.

Freestanding field

Recently, software engineering has been finding its own identity and emerging as an important freestanding field. Practitioners are slowly realizing that they form a huge community in their own right. Software engineering may need to create a form of regulation/licensing appropriate to its own circumstances.

See also Comparing software engineering and related fields.

[edit]

History

Software engineering has a long evolving history. Both the tools that are used and the applications that are written have evolved over time. It seems likely that software engineering will continue evolving for many decades to come.

See also History of software engineering.

[edit]

60 year time line

- 1940s: First computer users struggled to write code using machine code.
- 1950s: Early tools, such as macro assemblers and interpreters were available to improve productivity and quality.
- 1960s: Second generation tools like optimizing compilers and inspections were being used to improve productivity and quality. The concept of software engineering was widely discussed. First really big (1000 programmer) projects.
- 1970s: Collaborative software tools, such as unix, code repositories, make, and so on.
- 1980s: Personal computers and workstations and an emphasis on process like the CMM.
- 1990s: The WWW and Agile processes like Extreme programming.
- 2000s: Acceptance of software engineering profession. Fight over what it means.

[edit]

Future directions for software engineering

Aspect-oriented programming and agile methods are important emerging SE technologies and practices.

Aspects

Aspects help programmers deal with *ilities* by providing tools to add or remove boilerplate code from many areas in the source code. Aspects describe how all objects or functions should behave in particular circumstances. For example, aspects can add debugging, logging, or locking control into all objects of particular types. Researchers are currently working to understand how to use aspects to design general-purpose code. Related concepts include generative programming and templates.

Agile

Agile software development guides software development projects that evolve rapidly with changing expectations and competitive markets. The heavy, document-driven processes (like TickIT, CMM and ISO 9000) are fading in importance. Some people believe that companies and agencies export many of the jobs that can be guided by heavy-weight processes. Related concepts include extreme programming and lean software development.

The *Future of Software Engineering (http://www.softwaresystems.org/future.html)* conference (FOSE) held at the ICSE 2000 documented the state of the art of SE in 2000 and listed many problems to be solved over the next decade. The Feyerabend project (*http://www.dreamsongs.com/Feyerabend/Feyerabend.html*) attempts to discover the future of software engineering by seeking and publishing innovative ideas.

[edit]

Conferences, organizations and publications

[edit]

Conferences

Several academic conferences devoted to software engineering are held every year. There are also many other academic conferences every year devoted to special topics within SE, such as programming languages, requirements, testing, and so on.

ICSE

The biggest and oldest conference devoted to software engineering is the International Conference on Software Engineering (*http://www.icse-*

conferences.org/). This conference meets every year to discuss improvements in research, education, and practice.

ESEC

The European Software Engineering Conference (*http://esecfse.cs.helsinki.fi/*).

FSE

The Foundations of Software Engineering (*http://www.isr.uci.edu/FSE-12/*) conference is held every year, alternating between Europe and North America. It emphasizes theoretical and foundational issues.

CUSEC

Conferences dedicated to inform undergraduate students like the annual *Canadian University Software Engineering Conference (http://www.cusec.ca)* are also very promising for the future generation. It is completely organized by undergraduate students and lets different Canadian Universities interested in Software Engineering host the conference each year. Past guests includes Kent Beck, Joel Spolsky, Philippe Kruchten, Hal Helms, Craig Larman as well as university professors and students.

[edit]

Organizations

- Association for Computing Machinery (ACM)
- IEEE Computer Society

[edit]

Publications

- Important publications in software engineering

[edit]

Quotes

- *[...] software engineering has accepted as its charter "How to program if you cannot."* (Edsger Dijkstra in The Cruelty of Really Teaching Computer Science)
- *The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.* (Fred Brooks in No Silver Bullet)