Software engineering

Software engineering is the application of engineering to the development of software in a systematic method.[1][2][3]

Notable definitions of software engineering include:

- "the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software"—The Bureau of Labor Statistics—IEEE *Systems and software engineering - Vocabulary*[4]
- "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"—IEEE *Standard Glossary of Software Engineering Terminology*[5]
- "an engineering discipline that is concerned with all aspects of software production"—Ian Sommerville[6]
- "the establishment and use of sound engineering principles in order to economically obtain software that is reliable and works efficiently on real machines"—Fritz Bauer[7]

The term has also been used less formally:

- as the informal contemporary term for the broad range of activities that were formerly called computer programming and systems analysis;[8]
- as the broad term for all aspects of the *practice* of computer programming, as opposed to the *theory* of computer programming, which is formally studied as a sub-discipline of computer science;[9]
- as the term embodying the *advocacy* of a specific approach to computer programming, one that urges that it be treated as an engineering discipline rather than an art or a craft, and advocates the codification of recommended practices.[10]

Contents

History

When the first digital computers appeared in the early 1940s,[11] the instructions to make them operate were wired into the machine. Practitioners quickly realized that this design was not flexible and came up with the "stored program architecture" or von Neumann architecture. Thus the division between "hardware" and "software" began with abstraction being used to deal with the complexity of computing.

Programming languages started to appear in the early 1950s[12] and this was also another major step in abstraction. Major languages such as Fortran, ALGOL, and COBOL were released in the late 1950s to deal with scientific, algorithmic, and business problems respectively. David Parnas introduced the key concept of modularity and information hiding in 1972[13] to help

programmers deal with the ever-increasing complexity of software systems.

The origins of the term "software engineering" have been attributed to various sources. The term "software engineering" appeared in a list of services offered by companies in the June 1965 issue of COMPUTERS and AUTOMATION (http://bitsavers.trailing-edge.com/pdf/computersAndAutomation/196506.pdf) and was used more formally in the August 1966 issue of Communications of the ACM (Volume 9, number 8) “letter to the ACM membership” by the ACM President Anthony A. Oettinger;[14], it is also associated with the title of a NATO conference in 1968 by Professor Friedrich L. Bauer, the first conference on software engineering.[15]. At the time there was perceived to be a "software crisis".[16][17][18] The 40th International Conference on Software Engineering (ICSE 2018) celebrates 50 years of "Software Engineering" with the Plenary Sessions' keynotes of Frederick Brooks[19] and Margaret Hamilton.[20]

In 1984, the Software Engineering Institute (SEI) was established as a federally funded research and development center headquartered on the campus of Carnegie Mellon University in Pittsburgh, Pennsylvania, United States. Watts Humphrey founded the SEI Software Process Program, aimed at understanding and managing the software engineering process. The Process Maturity Levels introduced would become the Capability Maturity Model Integration for Development(CMMI-DEV), which has defined how the US Government evaluates the abilities of a software development team.

Modern, generally accepted best-practices for software engineering have been collected by the ISO/IEC JTC 1/SC 7 subcommittee and published as the Software Engineering Body of Knowledge (SWEBOK).[21]

Subdisciplines

Software engineering can be divided into sub-disciplines.[22] Some of them are:

- Software requirements[1][22] (or Requirements engineering): The elicitation, analysis, specification, and validation of requirements for software.
- Software design:[1][22] The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.
- Software construction:[1][22] The detailed creation of working, meaningful software through a combination of programming (aka coding), verification, unit testing, integration testing, and debugging.
- Software testing:[1][22] An empirical, technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.
- Software maintenance:[1][22] The totality of activities required to provide cost-effective support to software.
- Software configuration management:[1][22] The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle. Modern processes use software versioning.
- Software engineering management:[1][22] The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
- Software development process:[1][22] The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.
- Software engineering models and methods[22] impose structure on software engineering with the goal of making that activity systematic, repeatable, and ultimately more success-oriented
- Software quality[22]
- Software engineering professional practice[22] is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner
- Software engineering economics[22] is about making decisions related to software engineering in a business context
- Computing foundations[22]
- Mathematical foundations[22]
- Engineering foundations[22]

Education

Knowledge of computer programming is a prerequisite for becoming a software engineer. In 2004 the IEEE Computer Society produced the SWEBOK, which has been published as ISO/IEC Technical Report 1979:2004, describing the body of knowledge that they recommend to be mastered by a graduate software engineer with four years of experience.[23] Many software engineers enter the profession by obtaining a university degree or training at a vocational school. One standard international curriculum for undergraduate software engineering degrees was defined by the Joint Task Force on Computing Curricula of the IEEE Computer

Society and the Association for Computing Machinery, and updated in 2014.[24] A number of universities have Software Engineering degree programs; as of 2010, there were 244 Campus Bachelor of Software Engineering programs, 70 Online programs, 230 Masters-level programs, 41 Doctorate-level programs, and 69 Certificate-level programs in the United States.[25]

In addition to university education, many companies sponsor internships for students wishing to pursue careers in information technology. These internships can introduce the student to interesting real-world tasks that typical software engineers encounter every day. Similar experience can be gained through military service in software engineering.

Profession

Legal requirements for the licensing or certification of professional software engineers vary around the world. In the UK, there is no licensing or legal requirement to assume or use the job title Software Engineer. In some areas of Canada, such as Alberta, British Columbia, Ontario,[26] and Quebec, software engineers can hold the Professional Engineer (P.Eng) designation and/or the Information Systems Professional (I.S.P.) designation. In Europe, Software Engineers can obtain the European Engineer (EUR ING) professional title.

The United States, since 2013, has offered an *NCEES Professional Engineer* exam for Software Engineering, thereby allowing Software Engineers to be licensed and recognized.[27] NCEES will end the exam after April 2019 due to lack of participation.[28] Mandatory licensing is currently still largely debated, and perceived as controversial. In some parts of the US such as Texas, the use of the term Engineer is regulated by law and reserved only for use by individuals who have a Professional Engineer license.

The IEEE Computer Society and the ACM, the two main US-based professional organizations of software engineering, publish guides to the profession of software engineering. The IEEE's *Guide to the Software Engineering Body of Knowledge - 2004 Version*, or SWEBOK, defines the field and describes the knowledge the IEEE expects a practicing software engineer to have. The most current SWEBOK v3 is an updated version and was released in 2014.[29] The IEEE also promulgates a "Software Engineering Code of Ethics".[30]

Employment

The U. S. Bureau of Labor Statistics counted 1,256,200 software Developers (Engineers) holding jobs in the U.S. in 2016[31]. Employment of computer and information technology occupations is projected to grow 13 percent from 2016 to 2026, faster than the average for all occupations. These occupations are projected to add about 557,100 new jobs. Demand for these workers will stem from greater emphasis on cloud computing, the collection and storage of big data, and information security[32]. Yet, the BLS also says some employment in these occupations are slowing and computer programmers is projected to decline 7 percent from 2016 to 2026 since computer programming can be done from anywhere in the world, so companies sometimes hire programmers in countries where wages are lower[33]. Due to its relative newness as a field of study, formal education in software engineering is often taught as part of a computer science curriculum, and many software engineers hold computer science degrees and have no engineering background whatsoever.[34]

Many software engineers work as employees or contractors. Software engineers work with businesses, government agencies (civilian or military), and non-profit organizations. Some software engineers work for themselves as freelancers. Some organizations have specialists to perform each of the tasks in the software development process. Other organizations require software engineers to do many or all of them. In large projects, people may specialize in only one role. In small projects, people may fill several or all roles at the same time. Specializations include: in industry (analysts, architects, developers, testers, technical support, middleware analysts, managers) and in academia (educators, researchers).

Most software engineers and programmers work 40 hours a week, but about 15 percent of software engineers and 11 percent of programmers worked more than 50 hours a week in 2008. Potential injuries in these occupations are possible because like other workers who spend long periods sitting in front of a computer terminal typing at a keyboard, engineers and programmers are susceptible to eyestrain, back discomfort, and hand and wrist problems such as carpal tunnel syndrome.[35]

Certification

The Software Engineering Institute offers certifications on specific topics like security, process improvement and software architecture.[36] IBM, Microsoft and other companies also sponsor their own certification examinations. Many IT certification programs are oriented toward specific technologies, and managed by the vendors of these technologies.[37] These certification programs are tailored to the institutions that would employ people who use these technologies.

Broader certification of general software engineering skills is available through various professional societies. As of 2006, the IEEE had certified over 575 software professionals as a Certified Software Development Professional (CSDP).[38] In 2008 they added an entry-level certification known as the Certified Software Development Associate (CSDA).[39] The ACM had a professional certification program in the early 1980s, which was discontinued due to lack of interest. The ACM examined the possibility of professional certification of software engineers in the late 1990s, but eventually decided that such certification was inappropriate for the professional industrial practice of software engineering.[40]

In the U.K. the British Computer Society has developed a legally recognized professional certification called *Chartered IT Professional (CITP)*, available to fully qualified members (*MBCS*). Software engineers may be eligible for membership of the Institution of Engineering and Technology and so qualify for Chartered Engineer status. In Canada the Canadian Information Processing Society has developed a legally recognized professional certification called *Information Systems Professional (ISP)*.[41] In Ontario, Canada, Software Engineers who graduate from a *Canadian Engineering Accreditation Board (CEAB)* accredited program, successfully complete PEO's (*Professional Engineers Ontario*) Professional Practice Examination (PPE) and have at least 48 months of acceptable engineering experience are eligible to be licensed through the *Professional Engineers Ontario* and can become Professional Engineers P.Eng.[42] The PEO does not recognize any online or distance education however; and does not consider Computer Science programs to be equivalent to software engineering programs despite the tremendous overlap between the two. This has sparked controversy and a certification war. It has also held the number of P.Eng holders for the profession exceptionally low. The vast majority of working professionals in the field hold a degree in CS, not SE. Given the difficult certification path for holders of non-SE degrees, most never bother to pursue the license.

Impact of globalization

The initial impact of outsourcing, and the relatively lower cost of international human resources in developing third world countries led to a massive migration of software development activities from corporations in North America and Europe to India and later: China, Russia, and other developing countries. This approach had some flaws, mainly the distance / timezone difference that prevented human interaction between clients and developers and the massive job transfer. This had a negative impact on many aspects of the software engineering profession. For example, some students in the developed world avoid education related to software engineering because of the fear of offshore outsourcing (importing software products or services from other countries) and of being displaced by foreign visa workers.[43] Although statistics do not currently show a threat to software engineering itself; a related career, computer programming does appear to have been affected.[44][45] Nevertheless, the ability to smartly leverage offshore and near-shore resources via the follow-the-sun workflow has improved the overall operational capability of many organizations.[46] When North Americans are leaving work, Asians are just arriving to work. When Asians are leaving work, Europeans are arriving to work. This provides a continuous ability to have human oversight on business-critical processes 24 hours per day, without paying overtime compensation or disrupting a key human resource, sleep patterns.

While global outsourcing has several advantages, global - and generally distributed - development can run into serious difficulties resulting from the distance between developers. This is due to the key elements of this type of distance that have been identified as geographical, temporal, cultural and communication (that includes the use of different languages and dialects of English in different locations).[47] Research has been carried out in the area of global software development over the last 15 years and an extensive body of relevant work published that highlights the benefits and problems associated with the complex activity. As with other aspects of software engineering research is ongoing in this and related areas.

Related fields

Software engineering is a direct sub-field of engineering and has an overlap with computer science and management science [48]. It is also considered a part of overall systems engineering.

Computer Science

In general, software engineering focuses more on techniques for the application of software development in industry,[49][50] while computer science focuses more on algorithms and theory.

Controversy

Criticism

Software engineering sees its practitioners as individuals who follow well-defined engineering approaches to problem-solving. These approaches are specified in various software engineering books and research papers, always with the connotations of predictability, precision, mitigated risk and professionalism. This perspective has led to calls for licensing, certification and codified bodies of knowledge as mechanisms for spreading the engineering knowledge and maturing the field.

Software craftsmanship has been proposed by a body of software developers as an alternative that emphasizes the coding skills and accountability of the software developers themselves without professionalism or any prescribed curriculum leading to ad-hoc problem-solving (craftmanship) without engineering (lack of predictability, precision, missing risk mitigation, methods are informal and poorly defined). The Software Craftsmanship Manifesto (http://manifesto.softwarecraftsmanship.org/) extends the Agile Software Manifesto[51] and draws a metaphor between modern software development and the apprenticeship model of medieval Europe.

Software engineering extends engineering and draws on the engineering model, i.e. engineering process, engineering project management, engineering requirements, engineering design, engineering construction, and engineering validation. The concept is so new that it is rarely understood, and it is widely misinterpreted, including in software engineering textbooks, papers, and among the communities of programmers and crafters.

One of the core issues in software engineering is that its approaches are not empirical enough because a real-world validation of approaches is usually absent, or very limited and hence software engineering is often misinterpreted as feasible only in a "theoretical environment."

Edsger Dijkstra, the founder of many of the concepts used within software development today, rejected the idea of "software engineering" up until his death in 2002, arguing that those terms were poor analogies for what he called the "radical novelty" of computer science:

> A number of these phenomena have been bundled under the name "Software Engineering". As economics is known as "The Miserable Science", software engineering should be known as "The Doomed Discipline", doomed because it cannot even approach its goal since its goal is self-contradictory. Software engineering, of course, presents itself as another worthy cause, but that is eyewash: if you carefully read its literature and analyse what its devotees actually do, you will discover that software engineering has accepted as its charter "How to program if you cannot."[52]

See also

- Bachelor of Science in Information Technology
- Bachelor of Software Engineering
- List of software engineering conferences
- List of software engineering publications
- Software craftsmanship
- Software Engineering Institute

Notes

1. Abran et al. 2004, pp. 1–1
2. ACM (2007). "Computing Degrees & Careers" (http://computingcareers.acm.org/?page_id=12). ACM. Retrieved 2010-11-23.
3. Laplante, Phillip (2007). *What Every Engineer Should Know about Software Engineering* (https://books.google.com/?id=pFHYk0KWAEgC&lpg=PP1&dq=What%20Every%20Engineer%20Should%20Know%20about%20Software%20Engineering.&pg=PA1#v=onepage&q&f=false). Boca Raton: CRC. ISBN 978-0-8493-7228-5. Retrieved 2011-01-21.

4. *Systems and software engineering - Vocabulary*, ISO/IEC/IEEE std 24765:2010(E), 2010.
5. *IEEE Standard Glossary of Software Engineering Terminology*, IEEE std 610.12-1990, 1990.
6. Sommerville, Ian (2007) [1982]. "1.1.2 What is software engineering?". *Software Engineering* (http://www.pearsoned.co.uk/HigherEducation/Booksby/Sommerville/) (8th ed.). Harlow, England: Pearson Education. p. 7. ISBN 0-321-31379-8. "Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use. In this definition, there are two key phrases:
1. *Engineering discipline* Engineers make things work. They apply theories, methods and tools where these are appropriate [. . .] Engineers also recognize that they must work to organizational and financial constraints. [. . .]
2. *All aspects of software production* Software engineering is not just concerned with the technical processes of software development but also with activities such as software project management and with the development of tools, methods and theories to support software production."
7. "Software Engineering". *Information Processing*. North-Holland Publishing Co.year = 1972. **71**: 530–538.
8. Akram I. Salah (2002-04-05). "Engineering an Academic Program in Software Engineering" (http://www.micsymposium.org/mics_2002/SALAH.PDF) (PDF). 35th Annual Midwest Instruction and Computing Symposium. Retrieved 2006-09-13.: "For some, software engineering is just a glorified name for programming. If you are a programmer, you might put 'software engineer' on your business card—never 'programmer' though."
9. Mills, Harlan D., J. R. Newman, and C. B. Engle, Jr., "An Undergraduate Curriculum in Software Engineering," in Deimel, Lionel E. (1990). *Software Engineering Education: SEI Conference 1990, Pittsburgh, Pennsylvania, USA, April 2–3,...* Springer. ISBN 0-387-97274-9.,p. 26 (https://books.google.com/books?vid=ISBN3540972749&id=ZuWbyy2blMEC&pg=PA26&lpg=PA26&sig=Yxs2mS5S0Xs2cnGv2vYsYUcFNQM): "As a practical matter, we regard software engineering as the necessary preparation for the practicing, software development and maintenance professional. The Computer Scientist is preparing for further theoretical studies..."
10. David Budgen; Pearl Brereton; Barbara Kitchenham; Stephen Linkman (2004-12-14). "Realizing Evidence-based Software Engineering" (https://web.archive.org/web/20061217013922/http://evidence.cs.keele.ac.uk/rebse.html). Archived from the original (http://evidence.cs.keele.ac.uk/rebse.html) on 2006-12-17. Retrieved 2006-10-18.: "We believe that software engineering can only advance as an engineering discipline by moving away from its current dependence upon advocacy and analysis,...."
11. Leondes, Cornelius T. (2002). *Intelligent Systems: Technology and Applications*. CRC Press. p. I-6. ISBN 978-0-8493-1121-5. "1.4 Computers and a First Glimpse at AI (1940s)"
12. Campbell-Kelly, Martin (April 1982). "The Development of Computer Programming in Britain (1945 to 1955)". *IEEE Annals of the History of Computing*. **4** (2): 121–139. doi:10.1109/MAHC.1982.10016 (https://doi.org/10.1109%2FMAHC.1982.10016).
13. Parnas, David (December 1972). "On the Criteria To Be Used in Decomposing Systems into Modules" (http://www.acm.org/classics/may96/). *Communications of the ACM*. **15** (12): 1053–1058. doi:10.1145/361598.361623 (https://doi.org/10.1145%2F361598.361623). Retrieved 2008-12-26.
14. "The origin of "software engineering"" (https://bertrandmeyer.com/2013/04/04/the-origin-of-software-engineering/). Retrieved 17 Nov 2017.
15. Randall, Brian. "The 1968/69 NATO Software Engineering Reports" (http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/). Retrieved 17 Nov 2017.
16. Sommerville 2008, p. 26
17. Peter, Naur; Randell, Brian (7–11 October 1968). *Software Engineering: Report of a conference sponsored by the NATO Science Committee* (http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF) (PDF). Garmisch, Germany: Scientific Affairs Division, NATO. Retrieved 2008-12-26.
18. Randell, Brian (10 August 2001). "The 1968/69 NATO Software Engineering Reports" (http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/index.html). *Brian Randell's University Homepage*. The School of the Computer Sciences, Newcastle University. Retrieved 2008-10-11. "The idea for the first NATO Software Engineering Conference, and in particular that of adopting the then practically unknown term "software engineering" as its (deliberately provocative) title, I believe came originally from Professor Fritz Bauer."
19. 2018 International Conference on Software Engineering celebrating its 40th anniversary, and 50 years of Software engineering. "ICSE 2018 - Plenary Sessions - Fred Brooks" (https://www.youtube.com/watch?v=StN49re9Nq8&t=67s). Retrieved 9 Aug 2018.
20. 2018 International Conference on Software Engineering celebrating its 40th anniversary, and 50 years of Software engineering. "ICSE 2018 - Plenary Sessions - Margaret Hamilton" (https://www.youtube.com/watch?v=ZbVOF0Uk5lU). Retrieved 9 Aug 2018.
21. "ISO/IEC TR 19759:2005" (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=33897). Retrieved 2012-04-01.
22. "Software Engineering Body of Knowledge (SWEBOK Version 3), 2014" (https://www.computer.org/web/swebok/v3) (pdf). *www.swebok.org*. IEEE Computer Society. Retrieved 24 May 2016.

23. Abran, Alain, ed. (2005) [2004]. "Chapter 1: Introduction to the Guide" (http://www.computer.org/portal/web/swebok/html/ch1). *Guide to the Software Engineering Body of Knowledge* (http://www.computer.org/portal/web/swebok). Los Alamitos: IEEE Computer Society. ISBN 0-7695-2330-7. Retrieved 2010-09-13. "The total volume of cited literature is intended to be suitable for mastery through the completion of an undergraduate education plus four years of experience."
24. "SE2014 Software Engineering Curriculum" (https://www.acm.org/binaries/content/assets/education/se2014.pdf) (PDF).
25. [1] (http://www.gradschools.com/search-programs/software-engineering) Degree programs in Software Engineering
26. Williams, N.S.W. (19–21 February 2001). "Professional Engineers Ontario's approach to licensing software engineering practitioners". *Software Engineering Education and Training, 2001 Proceedings. 14th Conference on*. Charlotte, NC: IEEE. pp. 77–78.
27. "NCEES Software Engineering Exam Specifications" (https://web.archive.org/web/20130827220334/http://cdn1.ncees.co/wp-content/uploads/2012/11/Exam-specifications_PE-Software-Apr-2013.pdf) (PDF). Archived from the original (http://cdn1.ncees.co/wp-content/uploads/2012/11/Exam-specifications_PE-Software-Apr-2013.pdf) (PDF) on 2013-08-27. Retrieved 2012-04-01.
28. "NCEES discontinuing PE Software Engineering exam" (https://ncees.org/ncees-discontinuing-pe-software-engineering-exam/). National Council of Examiners for Engineering and Surveying. 13 March 2018. Retrieved 6 August 2018.
29. "'SWEBOK Guide Version 3'" (http://www.computer.org/web/swebok/v3). Retrieved 2015-03-09.
30. "'"Software Engineering Code of Ethics"" (http://www.computer.org/cms/Computer.org/Publications/code-of-ethics.pdf) (PDF). Retrieved 2012-03-25.
31. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm
32. https://www.bls.gov/ooh/computer-and-information-technology/home.htm
33. https://www.bls.gov/ooh/computer-and-information-technology/computer-programmers.htm
34. "Software Engineering" (http://computingcareers.acm.org/?page_id=12). Retrieved 2008-02-01.
35. "Computer Software Engineers and Computer Programmers" (http://www.bls.gov/oco/ocos303.htm#training). Retrieved 2009-12-17.
36. "SEI certification page" (http://www.sei.cmu.edu/certification/). Sei.cmu.edu. Retrieved 2012-03-25.
37. Wyrostek, Warren (March 14, 2008). "The Top 10 Problems with IT Certification in 2008" (http://www.informit.com/articles/article.aspx?p=1180991). *InformIT*. Retrieved 2009-03-03.
38. IEEE Computer Society. "2006 IEEE computer society report to the IFIP General Assembly" (http://www.ifip.org/minutes/GA2006/Tab18b-US-IEEE.pdf) (PDF). Retrieved 2007-04-10.
39. IEEE. "CSDA" (http://www.computer.org/portal/web/certification/csda). Retrieved 2010-04-20.
40. ACM (July 17, 2000). "A Summary of the ACM Position on Software Engineering as a Licensed Engineering Profession" (https://web.archive.org/web/20080517201804/http://www.cs.wm.edu/~coppit/csci690-spring2004/papers/selep_main.pdf) (PDF). Association for Computing Machinery (ACM). Archived from the original (http://www.cs.wm.edu/~coppit/csci690-spring2004/papers/selep_main.pdf) (PDF) on May 17, 2008. Retrieved 2009-03-03. "At its meeting in May 2000, the Council further concluded that the framework of a licensed professional engineer, originally developed for civil engineers, does not match the professional industrial practice of software engineering. Such licensing practices would give false assurances of competence even if the body of knowledge were mature; and would preclude many of the most qualified software engineers from becoming licensed."
41. Canadian Information Processing Society. "I.S.P. Designation" (http://www.cips.ca/standards/isp). Retrieved 2007-03-15.
42. "Professional Engineers Ontario: Welcome to PEO's website" (http://www.peo.on.ca). Peo.on.ca. Retrieved 2012-03-25.
43. Thibodaux, Patrick (2006-05-05). "As outsourcing gathers steam, computer science interest wanes" (http://www.computerworld.com/article/2555175/it-careers/as-outsourcing-gathers-steam--computer-science-interest-wanes.html). Computerworld.com. Retrieved 2016-12-06.
44. "Computer Programmers" (http://www.bls.gov/oco/ocos110.htm#outlook). Bls.gov. Retrieved 2012-03-25.
45. Mullins, Robert (2007-03-13). "Software developer growth slows in North America" (https://web.archive.org/web/20090404033214/http://www.infoworld.com/article/07/03/13/HNslowsoftdev_1.html). InfoWorld. Archived from the original (http://www.infoworld.com/article/07/03/13/HNslowsoftdev_1.html) on 2009-04-04. Retrieved 2012-03-25.
46. "Gartner Magic Quadrant" (http://www.cognizant.com/html/content/news/GartnerMQ_Cognizant.pdf) (PDF). Cognizant.com. Retrieved 2012-03-25.
47. Casey, Valentine (2010-08-20). "Virtual software team project management" (https://link.springer.com/article/10.1007%2Fs13173-010-0013-3). Springer. Retrieved 2013-12-06.
48. https://www.dagstuhl.de/Reports/96/9635.pdf
49. http://www.stevemcconnell.com/psd/04-senotcs.htm
50. https://engiegirlsatuwaterloo.wordpress.com/2013/08/29/computer-engineering-software-engineering-or-computer-science/
51. Beck, Kent; et al. (2001). "Manifesto for Agile Software Development" (http://agilemanifesto.org/). Agile Alliance. Retrieved 14 June 2010.

52. Dijkstra, E. W. (1988). "On the cruelty of really teaching computing science" (http://www.cs.utexas.edu/~EWD/transcriptions/EWD10xx/EWD1036.html). Retrieved 2014-01-10.

References

- Abran, Alain; Moore, James W.; Bourque, Pierre; Dupuis, Robert; Tripp, Leonard L. (2004). *Guide to the Software Engineering Body of Knowledge*. IEEE. ISBN 0-7695-2330-7.
- Sommerville, Ian (2008). *Software Engineering* (https://books.google.com/books?id=PqsWaBkFh1wC) (7 ed.). Pearson Education. ISBN 978-81-7758-530-8. Retrieved 10 January 2013.

Further reading

- Pressman, Roger S (2009). *Software Engineering: A Practitioner's Approach* (7th ed.). Boston, Mass: McGraw-Hill. ISBN 978-0073375977.
- Sommerville, Ian (2010) [2010]. *Software Engineering* (http://www.pearsoned.co.uk/HigherEducation/Booksby/Sommerville/) (9th ed.). Harlow, England: Pearson Education. ISBN 978-0137035151.
- Jalote, Pankaj (2005) [1991]. *An Integrated Approach to Software Engineering* (https://www.springer.com/gp/book/9780387208817) (3rd ed.). Springer. ISBN 0-387-20881-X.
- Bruegge, Bernd; Dutoit, Allen (2009). *Object-oriented software engineering : using UML, patterns, and Java* (3rd ed.). Prentice Hall. ISBN 978-0136061250.

External links

- Guide to the Software Engineering Body of Knowledge (http://www.swebok.org/)
- The Open Systems Engineering and Software Development Life Cycle Framework (http://OpenSDLC.org/) OpenSDLC.org the integrated Creative Commons SDLC
- Software Engineering Institute (http://www.sei.cmu.edu/) Carnegie Mellon

---