CHAPTER 11

SOFTWARE ENGINEERING PROFESSIONAL PRACTICE

ACRONYMS

ABET:	Accreditation Board for Engineering and Technology
CSAB:	Computer Sciences Accreditation Board

INTRODUCTION

The Software Engineering Professional Practice KA is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner. The study of professional practice includes the subareas of professionalism, group dynamics and psychology, and communication skills.

The phrase "professional practice" refers to a way of conducting services so as to achieve certain standards or criteria in both the process of performing a service and the end product resulting from the service. These standards and criteria can include both technical and non-technical aspects. Often, the concept of professional practice is more applicable within those professions that have a generally accepted body of knowledge; codes of ethics and professional conduct with penalties for violations; accepted processes for accreditation, certification, and licensing; and professional societies to provide and administer all of these. Admission to these professional societies is often predicated on a prescribed combination of education and experience.

A software engineer maintains a professional practice by following the codes of ethics and professional conduct and by performing all work in accordance with generally accepted practices, standards, and guidelines set by the applicable professional society. For example, the Association for Computing Machinery (ACM) and IEEE Computer Society (IEEE CS) have established a Code of Ethics for Software Professionals. ISO/IEC and IEEE have further provided internationally accepted software engineering standards. IEEE CS has established two international certifications programs and a corresponding Guide to the Software Engineering Body of Knowledge (SWEBOK).

BREAKDOWN OF TOPICS FOR SOFTWARE ENGINEERING PROFESSIONAL PRACTICE

The Software Engineering Professional Practice KA's breakdown of topics is shown in Figure 1.

1. Professionalism

A software engineer displays professionalism through adherence to codes of ethics and professional conduct, standards and practices which are established by the engineer's professional community.

The professional community is often represented by one or more professional societies; those societies publish codes of ethics and professional conduct and criteria for admittance to the community. Those criteria form the basis for accreditation and licensing activities, and may be used as a measure to determine engineering competence or negligence.

*1.1 Accreditation, Certification, and Licensing [1*s1.4.1, s1.5.1-1.5.4]*

1.1.1 Accreditation

Accreditation is a process to certify the competency, authority, or credibility of an organization. In many countries, the basic means by which engineers acquire knowledge is through completion of an accredited course of study. Often, engineering accreditation is performed by a government organization, such as the ministry of education. Such countries include France, Germany, China, Israel, Russia, and Italy. In the United States and Canada, however, the accreditation process is independent of government and performed by private membership associations. For example, in the United States, engineering accreditation is performed by the Accreditation Board for Engineering and Technology (ABET)—which includes representatives from the ACM and IEEE CS—which acts as the Computer Sciences Accreditation Board (CSAB). In Canada, the responsibility of accreditation falls to the Canadian Engineering Accreditation Board of Engineers Canada.

While the process of accreditation may be different for each country, the general meaning is the same. For an institution's course of study to be accredited means that the accreditation body recognizes "(an educational institution) as maintaining standards that qualify the graduates for admission to higher or more specialized institutions or for professional practice."[2] Earning accreditation requires that the curriculum encompass a "definition of a body of knowledge that registered practitioners must possess." [2]

1.1.2 Certification

Certification refers to the confirmation of a person's certain characteristics. A common type of certification is professional certification, where a person is certified as being able to competently complete a job or task in a certain discipline. An engineer obtains certification usually by passing an examination in conjunction with other, experience-based criteria. These examinations are

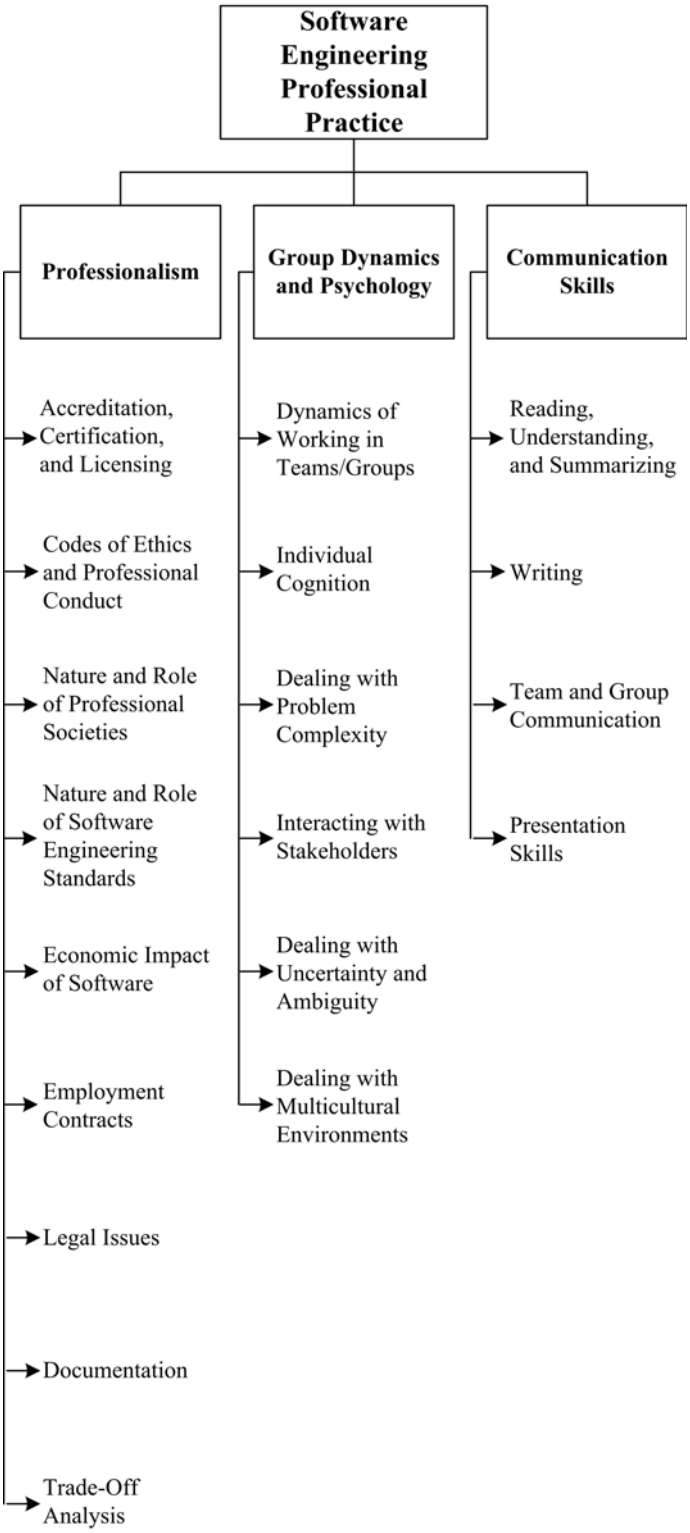

Figure 1: Breakdown of Topics

often administered by non-governmental organizations, such as professional societies.

In software engineering, certification testifies to one's qualification as a software engineer. Through certification, a software engineer gains third-party proof of knowledge and

fitness to practice, usually under a reputable certification program. For example, the IEEE CS has enacted two certification programs designed to confirm a software engineer's proficiency of standard software engineering practices and to advance one's career. Unlike an engineering license, however, a lack of certification does not exclude the individual from working as a software engineer.

1.1.3 Licensing

"Licensing" is the action of giving a person the authorization to perform certain kinds of activities while the noun "license" refers both to that authorization and the document recording that authorization. Licenses are usually issued by governmental authorities or statutory bodies to allow an activity that would otherwise be forbidden

Obtaining a license to practice requires not only that an individual meets a standard but that they do so with the ability to practice or operate. In general, engineers are licensed as a means of protecting the public from unqualified individuals. In some countries, no one can practice as a professional engineer unless licensed; further, no company may offer "engineering services" unless at least one licensed engineer is employed there.

The field of software engineering is a licensed discipline in Texas in the United States, Australia, and certain provinces in Canada. In the USA, the IEEE CS has been collaborating in an initiative led by the National Council of Examiner for Engineering and Survey to license software engineers who pass a competency exam. The first of such exams is expected to be given in 2013.

1.2 Codes of Ethics and Professional Conduct

*[1*s1.6-1.9, 3*s1.2, 4*c8, 5*c33, 6*]*

Codes of ethics and professional conduct comprise the values and behavior that an engineer's professional practice and decisions should embody.

Codes of ethics and professional conduct are established by the professional community (that is, engineering professional societies). They exist in the context of, and are adjusted to agree with, societal norms and local laws. Therefore, codes of ethics have to present guidance in the face of conflicting imperatives.

Once established, codes of ethics and professional conduct are enforced by the profession, as represented by professional societies or by a statutory body where the public interest is considered to be engaged. Violations may be acts of commission, such as concealing inadequate work; disclosing confidential information; falsifying information; or misrepresenting one's abilities. They may also occur through omission, including failure to disclose risks or to provide important information; failure to give proper credit or to acknowledge references; and failure to represent client interests. Violations of codes of ethics may result in penalties and possible expulsion from professional status.

A code of ethics and professional conduct for software engineering was approved by the ACM Council and the IEEE CS Board of Governors in 1999. Since standards and codes may be introduced, modified, or replaced at any time, software engineers bear responsibility for continuous study to stay current in their knowledge of the practice.

1.3 Nature and Role of Professional Societies

*[1*s1.1-1.2, 3*s1.2, 5*s35.1]*

Professional societies are comprised of a mix of practitioners and academics. These societies serve to define, advance, and regulate their corresponding professions. Professional societies help to establish professional standards as well as codes of ethics and professional conduct. Therefore, they also engage in related activities, which include:

- establishing and promulgating a body of generally accepted knowledge;
- accrediting, certifying, and licensing;
- dispensing disciplinary actions;
- advancing the profession through conferences, training, and publications.

Participation in professional societies assists the individual engineer in maintaining and sharpening their professional knowledge and relevancy.

1.4 Nature and Role of Software Engineering Standards

*[1*s5.3.2,s10.2.1, 5*s32.6, 7*c1s2]*

Software engineering standards covers a remarkable variety of topics. They provide guidelines for the practice and processes to be used during development, maintenance, and support of software. By codifying collective knowledge and experience, they establish a basis upon which further guidelines may be developed.

The benefits of software engineering standards are many and include (but are not limited to) improving software quality, protecting both software producers and buyers, increasing professional discipline, and helping technology transition. For example, software engineering standards play an important role in software technology transitions by providing software engineers with common terminology and a framework for communicating software technology across organizational boundaries, which is critical for the acceptance of such technology by late adopters.

1.5 Economic Impact of Software

*[3*s1.1.1, 4*s10.8, 8*c1]*

Software has economic effects at the individual, business, and societal levels. Software "success" may

be determined by the suitability of a product for a recognized problem as well as by its effectiveness when applied to that problem.

At the individual level, an engineer's continuing employment may depend on one's ability and willingness to interpret and execute tasks in meeting customers' or employers' needs and expectations. The customer or employer's financial situation may in turn be positively or negatively affected by the purchase of software.

At the business level, software properly applied to a problem can eliminate months of work and translate to elevated profits or more effective organizations. Furthermore, organizations that acquire or provide successful software may be a boon to the society in which they operate by providing both employment and improved services.

However, the development costs of software can also equate to those of any major acquisition, so loss of data, function, and reputation—with accompanying loss of goodwill in the marketplace—can occur.

At the societal level, in addition to the direct impact of software success or failure (accidents, interruptions, and loss of service), there are also indirect impacts, which include the success or failure of the organization that acquired or produced the software, increased or decreased societal productivity, harmonious or disruptive social order, and even the saving or loss of property and life.

1.6 Employment Contracts
*[1*c7]*

Software engineering services may be provided under a variety of client-engineer relationships. The software engineering work may be solicited through supplier (meaning company-to-customer), consultancy (meaning company-to-customer or engineer-to-customer), or through direct hire (meaning employment). In all of these situations, the customer and supplier agree that a product or service will be provided in return for some consideration. Here, we are most concerned with the engineer-to-customer arrangement and its attendant agreements or contracts, whether they are of the direct-hire or consultant variety, and the issues they typically address.

A common concern in software engineering contracts is confidentiality. Employers derive commercial advantage from intellectual property, so they strive to protect that property from disclosure. Therefore, software engineers are often required to sign non-disclosure (NDA) or intellectual property (IP) agreements as a precondition to work. These agreements typically apply to information the software engineer could only gain through association with the customer. The terms of these agreements may extend past termination of the association.

Another concern is copyright ownership. Rights to software engineering assets—products, innovations, inventions, discoveries, and ideas—may reside with the employer or customer, either under explicit contract terms or relevant laws, if those assets are obtained during the term of the software engineer's relationship with that employer or customer. Contracts differ in the ownership of assets created using non-employer-owned equipment or information.

Finally, contracts can also specify the location (company, customer, or home) at which work is to be performed; standards to which that work will be held; limitations of the software engineer's and employer's liability; and administrative details such as rates, frequency of compensation, working hours, and working conditions.

1.7 Legal Issues
*[1*c6,c11, 4*s5.3-5.4, 9*s1.10]*

Issues surrounding software engineering professional practice include matters related to standards, trademarks, patents, copyrights, trade secrets, professional liability, legal requirements, and computer crime. It is therefore beneficial to possess knowledge of these issues and their applicability.

1.7.1 Standards
Software engineering standards establish guidelines for generally accepted practices and minimum requirements for products and services provided by a software engineer. For example, a highly relevant standard for software engineering is ISO/IEC/IEEE 12207 (2008): Information Technology – Software Life Cycle Processes [10].

Standards are valuable for guidance during the everyday conduct of software engineering activities. Adherence to standards facilitates personal process discipline by enumerating minimal characteristics of products and practice. That discipline helps to mitigate subconscious assumptions or overconfidence in a design. Further, adherence to standards is a major component of defense from legal action or from allegations of malpractice.

1.7.2 Trademarks
A trademark relates to any word, name, symbol, or device that is used in business transactions. It is used "to indicate the source or origin of the goods."[2]

Trademark protection protects names, logos, images, and packaging. However, if a name, image, or other trademarked asset becomes a generic term, then trademark protection is nullified.

1.7.3 Patents
Patents protect an inventor's right to manufacture and sell an idea. It consists of a set of exclusive rights granted by a sovereign government to an individual, group of individuals, or organization for a limited period of time. Patents are an old form of idea-ownership protection and date back to the 15th century.

Application for a patent entails careful records of the process that led to the invention. Patent attorneys are helpful in writing patent disclosure claims in a manner most likely to protect the software engineer's rights.

Note that, if inventions are made during the course of a software engineering contract, ownership may belong to the employer or customer rather than to the software engineer.

1.7.4 Copyrights

Copyright is enacted by most governments in the world to give exclusive rights of an original work to its creator, usually for a limited time. Copyrights protect the way an idea is presented—not the idea itself. For example, they may protect the particular wording of an account of an historical event, whereas the event itself is not protected. Copyrights are long-term and renewable; they date back to the 17th century.

1.7.5 Trade Secrets

In many countries, an intellectual asset such as a formula, process, design, method, pattern, instrument, or compilation of information may be considered a "trade secret," provided that these assets are not generally known and may provide a business some economic advantage. The designation of "trade secret" provides legal protection if the asset is stolen and if that protection is not subject to a time limit. However, if another party derives or discovers the same asset legally, then the asset is no longer protected and the other party will also possess all rights to use it.

1.7.6 Professional Liability

It is common for software engineers to be concerned with matters of professional liability. As an individual provides services to a client or employer, it is vital to adhere to standards and generally accepted practices, thereby protecting against allegations or proceedings of or related to malpractice, negligence, or incompetence.

For engineers, including software engineers, professional liability is related to product liability. Under the laws and rules governing in their jurisdiction, engineers may be held to account for failing to fully and conscientiously follow recommended practice; this is known as "negligence." They may also be subject to laws governing "strict liability" and either implied or express warranty, where, by selling the product, the engineer is held to warrant that the product is both suitable and safe for use. In some countries (for example, in the USA), "privity" (the idea that one could only sue the person selling the product) is no longer a defense against liability actions.

In fact, suits for liability can be brought under tort law (the tort law in the USA is a common law that allows anyone who is harmed to recover their loss) even if no guarantees were made. Because it is difficult to measure the suitability or safety of software, failure to take due care can be used to prove negligence on the part of software engineers. The best defense against such allegations is to show that standards and generally accepted practices were followed in the development of the product.

1.7.7 Legal Requirements

Software engineers operate within the confines of local, national, and international legal frameworks. Therefore, software engineers must be aware of legal requirements for:

- registration and licensing—including examination, education, experience,, and training requirements;
- contractual agreements; and
- non-contractual legalities, such as those governing liability.

1.7.8 Computer Crime

Computer crime refers to any crime that involves a computer. The computer may have been used in the commission of a crime or it may have been the target. This category of crime includes fraud, unauthorized access, spam, obscene or offensive content, threats, and harassment.

Computer users commit fraud by altering electronic data to facilitate illegal activity. Forms of unauthorized access include hacking, eavesdropping, and using computer systems in a way that is concealed from their owners.

Many countries have separate laws to cover computer crimes, but it has sometimes been difficult to prosecute computer crimes due to a lack of precisely framed statutes.

1.8 Documentation

*[1*s10.5.8, 4*s1.5, 5*c32]*

Providing thorough, accurate documentation is the responsibility of each software engineer. The adequacy of documentation is judged by different criteria based on the viewpoint of the stakeholder.

From a regulatory viewpoint, good documentation must comply with accepted standards. Under those standards, a software engineer must:

- document all relevant facts,
- document all risks and tradeoffs, and
- provide all appropriate warnings.

And never:

- certify or approve below-standard products,
- disclose confidential information, and
- falsify facts or data.

From a customer's viewpoint, good documentation must provide the necessary information to maintain and use the product as well as make appropriately informed decisions. A software engineer must:

- document the way the software was constructed—including software requirements, software design, software tools, and methods;
- provide user documentation that is both relevant and readable by the intended audience; and
- clearly enumerate warnings and critical procedures.

*1.9 Trade-Off Analysis [4*s1.2,c10, 9*s9.5.10]*

Within the practice of software engineering, a software engineer often has to choose between two or more competing solutions to a problem. The outcome of these choices is determined by the software engineer's professional evaluation of the risks, costs, and benefits of alternatives, in cooperation with stakeholders. The software engineer's evaluation is called "trade-off analysis."

A first step in a trade-off analysis is establishing design goals. This permits identification of the solution that most nearly meets those goals; this means that the way the goals are stated is critically important.

Design goals may include minimization of monetary cost and maximization of reliability or performance. However, it is difficult to formulate a tradeoff analysis of cost against risk, especially where primary (production) and secondary (risk-based) costs must be traded against each other.

Most often, trade-off analyses are used to decide which software requirements can be relaxed or dropped given the effects thereof.

2. Group Dynamics and Psychology

Engineering work is conducted within and for the good of society. A software engineer must be able to interact cooperatively and constructively with others to first determine and then meet needs and expectations for products and services. Knowledge of group dynamics and psychology is an asset when interacting with customers, coworkers, suppliers, and subordinates.

*2.1 Dynamics of Working in Teams/Groups [4*s1.6, 9*s1.3.5, c10]*

Performing teams—those that demonstrate consistent quality of work and progress toward goals—are cohesive and possess a cooperative, honest, and focused atmosphere. Individual and team goals are aligned so that the members naturally commit to and feel ownership of shared outcomes.

Team members facilitate this atmosphere by being intellectually honest, admitting ignorance, and acknowledging mistakes. They share responsibility, rewards, and workload fairly. They take care to communicate clearly, both directly and in documents and source code, so that information is accessible to everyone. Their criticisms of work products are framed in a constructive and non-personal way. This allows all the members to pursue a cycle of continuous improvement and growth without personal risk. In general, members of cohesive teams demonstrate respect for each other and their leader.

*2.2 Individual Cognition [4*s1.6.5, 5*c33]*

Engineers solve problems. The ability to solve problems effectively and efficiently is what every engineer strives for. However, problem solving is affected by the limits and processes of individual cognition. In software engineering, due notably to the highly abstract nature of software itself, individual cognition plays a more prominent role in problem solving.

In general, an individual's (in particular, a software engineer's) ability to decompose a problem and creatively develop a solution can be inhibited by:

- the need for more knowledge,
- subconscious assumptions,
- volume of data,
- fear of failure,
- culture, either industrial or organizational, and
- the lack of ability to express the problem.

The impact of these inhibiting factors can be reduced by cultivating good problem-solving habits that minimize the impact of assumptions. The ability to focus is vital, as is intellectual humility: both allow a software engineer to suspend personal considerations and consult with others freely.

There is a set of basic methods engineers use to facilitate problem solving. Breaking down problems and solving them one piece at a time reduces cognitive overload. (Please refer to the "Problem-Solving Techniques" topic in the Computing Foundation KA for a detailed discussion.) Taking advantage of professional curiosity and pursuing continuous professional development through training and study add skills and knowledge to the software engineer's portfolio; reading, networking, and experimenting are all valid means of professional development.

*2.3 Dealing with Problem Complexity [4*s3.2, 5*c33]*

Many, if not most, software engineering problems are too complex and difficult to address as a whole or to be tackled by individual software engineers. When such circumstances arise, the usual means to adopt is teamwork and problem decomposition.

Teams work together to deal with complex problems by drawing upon each others' knowledge and creativity. When software engineers work in teams,

different views and abilities of the individual engineers compliment each other and help build a solution that is otherwise difficult to come by. Some teamwork examples in software engineering are pair programming and code review.

Problem decomposition has been used since ancient times to deal with complex problems. When a problem is decomposed, the complexity of the target solution is also reduced, which helps to find the solution to the original problem. Problem decomposition fits naturally with software engineering's practice of separation of concerns and component-based approaches. (Please refer to the "Problem-Solving Techniques" topic in the Computing Foundation KA for a detailed discussion on problem decomposition.)

*2.4 Interacting with Stakeholders [9*s2.3.1]*

The success of a project and the software engineers engaged upon it is dependent upon positive interactions with stakeholders.

Stakeholders should provide support, information, and feedback at all stages of the software life cycle process. For example, during the early stages, it is critical to identify all stakeholders and discover how the product will affect them, so that a full Software Requirements Definition Document and associated requirements may be accurately derived. (Please refer to the Software Requirements KA for details.)

During the stages of software requirements, software design, and software construction, stakeholders provide feedback on early versions of the software as well as clarification of detailed or emergent software requirements.

Last, during software maintenance and until the end of product life, stakeholders provide feedback and problem reports so that the software may be extended and improved.

It is therefore vital to maintain open and productive communication with stakeholders for the duration of the software lifecycle.

*2.5 Dealing with Uncertainty and Ambiguity [3*s24.4,s26.2, 9*s9.4]*

As with engineers of other fields, software engineers must often deal with and resolve uncertainty and ambiguities while providing services and developing products. The software engineer must attack and reduce or eliminate any lack of clarity that is an obstacle to performing work.

Often, uncertainty is simply a reflection of lack of knowledge. In this case, investigation through recourse to formal sources such as textbooks and professional journals, interviews with stakeholders, or consultation with teammates and peers can overcome it.

When uncertainty or ambiguity cannot be overcome easily, however, software engineers or organizations may choose to regard it as a project risk. In this case, work estimates or pricing are adjusted to mitigate the anticipated cost of addressing it. (See "Risk Management" in the Software Engineering Management KA.)

*2.6 Dealing with Multicultural Environments [9*s10.7]*

Multicultural environments can have an impact on the dynamics of a group. This is especially true when the group is geographically separated or communication is infrequent, since such separation elevates the importance of each contact. Intercultural communication is even more difficult if the difference in time zones make oral communication even more infrequent.

Multicultural environments are quite prevalent in software engineering projects, perhaps more than in projects of other fields of engineering, due to the strong trend of international outsourcing and easy shipment of software components instantaneously across the globe. For example, it is not uncommon for a software project to be divided into pieces across national and cultural borders, and it is also quite common that a software project team consist of people from diverse cultural backgrounds.

For a software project to be a success, team members must achieve a level of tolerance, acknowledging that some rules depend on societal norms and that not all societies derive the same solutions and expectations.

This tolerance and accompanying understanding can be facilitated by the support of leadership and management. More frequent communication, including face-to-face meetings, can help to mitigate geographical and cultural divisions, promote cohesiveness, and raise productivity.

3. Communication Skills

It is vital that a software engineer communicate well, both orally and in reading and writing. Successful attainment of software requirements and deadlines depends on developing clear understandings between the software engineer and customers, supervisors, coworkers, and suppliers. Optimal problem solving is made possible through the ability to investigate, comprehend, and summarize information. Customer product acceptance and safe product usage depend on the provision of relevant training and documentation. It follows that the software engineer's own career success is affected by the ability to consistently provide oral and written communications effectively and on time.

3.1 Reading, Understanding, and Summarizing

*[5*s33.3]*

A software engineer must be able to read and understand technical material. Technical material includes reference books, manuals, research papers, and program source code.

Reading is not only a primary way of improving skills, but also a way of gathering information necessary for the completion of engineering goals. A software engineer sifts through accumulated information, filtering out the pieces that will be most helpful. Customers may request that a software engineer summarize the results of such information gathering for them, simplifying or explaining it so that they may make the final choice between competing implementations.

Reading and comprehending source code is also a component of information gathering and problem solving. When modifying, extending, or re-writing software, it is critical to understand both its implementation (which is immediately apparent) and its design (which must often be inferred).

*3.2 Writing [4*s1.5]*

Software engineers must provide written products as required by customer request or generally accepted practice. These written products may include source code, software project plans, software requirement documents, software design documents, user manual, technical reports and evaluations, justifications, diagrams and charts, and so forth.

Writing clearly and concisely is very important because it serves as an important (and, sometimes, primary) method of communication between relevant parties.

In all cases, written software engineering products must be composed so that they are accessible and relevant for their intended audience(s).

*3.3 Team and Group Communication [3*s22.3, 4*s1.6.8, 5*s27.1, 9*s10.4]*

Effective communication among team and group members is essential to a collaborative software engineering effort. Stakeholders must be consulted, decisions must be made, and plans must be generated. The greater the number of team and group members, the greater the need to communicate.

However, the number of communication paths grows exponentially with the addition of each team member. Further, team members are unlikely to communicate with anyone perceived to be removed from them by more than two degrees (levels). This problem is more serious in the software industry because many software projects are spread across national and continental borders. How, then, to achieve effective information exchange?

Some communication can be accomplished in writing. Software documentation is a common substitute for direct interaction. Email is another but, although it is useful, it is not always enough; if recipients send too many messages, it becomes difficult to identify the important information.

Some software teams focus on face-to-face interaction and promote such interaction by office space arrangement. Although private offices improve individual productivity, co-locating team members in physical or virtual forms and providing communal work areas is important to collaborative efforts.

*3.4 Presentation Skills [3*c22, 4*s1.5, 9*s10.7-10.8]*

Software engineers rely on their presentation skills during software life cycle processes. For example, during the software requirements phase, software engineers may walk customers and teammates through software requirements and conduct formal requirements reviews (see the Software Plural KA). During and after software design, software construction, and software maintenance, software engineers lead reviews, product walkthroughs, and training (see "Review and Audits" in the Software Quality KA). All of these require the ability to present technical information to groups and solicit ideas or feedback.

The software engineer's ability to convey concepts effectively in a presentation therefore influences product acceptance, management, and customer support; it also influences the ability of stakeholders to comprehend and assist in the product effort.

[1*] F. Bott, *et al.*, *Professional Issues in Software Engineering*, 3rd ed. New York: Taylor & Francis, 2000.

[2] "Merriam-Webster's Collegiate Dictionary," 11th ed. Springfield, MA: Merriam-Webster, 2003.

[3*] I. Sommerville, *Software Engineering*, 9th ed. New York: Addison-Wesley, 2010.

[4*] G. Voland, *Engineering by Design*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[5*] S. McConnell, *Code Complete*, 2nd ed. Redmond, WA: Microsoft Press, 2004.

[6*] IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices. (1999, July 22, 2010). *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*. Available: http://www.acm.org/serving/se/code.htm

[7*] J. W. Moore, *The Road Map to Software Engineering: A Standards-Based Guide*, 1st ed. Hoboken, NJ: Wiley-IEEE Computer Society Press, 2006.

[8*] S. Tockey, *Return on Software: Maximizing the Return on Your Software Investment*, 1st ed. Boston: Addison-Wesley, 2004.

[9*] R. E. Fairley, *Managing and Leading Software Projects*. Hoboken, NJ: Wiley-IEEE Computer Society Press, 2009.

[10] ISO/IEC/IEEE, "ISO/IEC/IEEE 12207:2008: Information Technology – Software Life Cycle Processes," 2nd ed, 2008.

Matrix of Topics vs. References for the Software Engineering Professional Service KA

	[1*]	[2*]	[3*]	[4*]	[5*]	[6*]	[7*]	[8*]
1. Professionalism	c1, s5.3.2, c6-7, c10-11	c1	s1.2-1.5 s5.3, c8, c10	c32-33, s35.1	s1.10, s9.5.10	c1s2		
1.1 Accreditation, Certification and Licensing	s1.4.1, s1.5.1-1.5.4							
1.2 Codes of Ethics and Professional Conduct	s1.6-1.9	s1.2	c8	c33				*
1.3 Nature and Role of Professional Societies	s1.1-1.2,	s1.2		s35.1				
1.4 Nature & Role of Software Engineering Standard	s5.3.2, s10.2.1			s32.6		c1s2		
1.5 Economic Impact of Software		s1.1.1	s10.8				c1	
1.6 Employment Contracts	c7							
1.7 Legal Issues	c6, c11		s5.3-5.4		s1.10			
1.8 Documentation	s10.5.8		s1.5	c32				
1.9 Trade-Off Analysis			s1.2, c10		s9.5.10			
2. Group Dynamics and Psychology		c22, s24.4, s26.2	s1.6	c33	s1.3.5, s2.3.1, s9.4, c10			
2.1 Dynamics of Working in Teams/Groups			s1.6		s1.3.5, c10			
2.2 Individual Cognition			s1.6.5	c33				
2.3 Dealing with Problem Complexity			s3.2	c33				
2.4 Interacting with Stakeholders					s2.3.1			
2.5 Dealing with Uncertainty and Ambiguity		s24.4, s26.2			s9.4			
2.6 Dealing with Multicultural Environments					s10.7			
3. Communication Skills		c22	s1.5-1.6	s27.1, s33.3	s10.4-10.8			
3.1 Reading, Understanding and Summarizing				s33.3				
3.2 Writing			s1.5					
3.3 Team and Group Communication		s22.3	s1.6.8	s27.1	s10.4			
3.4 Presentation Skills		c22	s1.5		s10.7-10.8			

Recommended references for the Knowledge Area

[1*] F. Bott*, et al.*, *Professional Issues in Software Engineering*, 3rd ed. New York: Taylor & Francis, 2000.

[2*] I. Sommerville, *Software Engineering*, 9th ed. New York: Addison-Wesley, 2010.

[3*] G. Voland, *Engineering by Design*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[4*] S. McConnell, *Code Complete*, 2nd ed. Redmond, WA: Microsoft Press, 2004.

[5*] R. E. Fairley, *Managing and Leading Software Projects*. Hoboken, NJ: Wiley-IEEE Computer Society Press, 2009.

[6*] J. W. Moore, *The Road Map to Software Engineering: A Standards-Based Guide*, 1st ed. Hoboken, NJ: Wiley-IEEE Computer Society Press, 2006.

[7*] S. Tockey, *Return on Software: Maximizing the Return on Your Software Investment*, 1st ed. Boston: Addison-Wesley, 2004.

[8*] IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices. (1999, July 22, 2010). *Software Engineering Code of Ethics and Professional Practice (Version 5.2)*. Available: http://www.acm.org/serving/se/code.htm