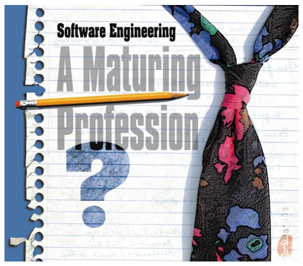
What Knowledge Is Important to a Software Professional?



Efforts to develop licensing requirements, curricula, or training programs for software professionals should consider the experience of the practitioners who actually perform the work.

Timothy C. Lethbridge
University of Ottawa

Software engineering licensing bodies, universities designing curricula, companies focusing on better training for their staff, and the IEEE in its Software Engineering Body of Knowledge (SWEBOK) project are all pursuing efforts to define the subject matter that software professionals should know. Whereas most groups are basing their decisions about the software engineering curriculum on the opinions of experts in the field, we were more interested in learning what subject matter practitioners themselves actually find most important in their work.

From May to October 1998, my colleagues and I surveyed software professionals representing a wide variety of industries, job functions, and countries to learn which educational topics have proved most important to them in their careers and to identify the topics for which their education or current knowledge could be improved. We used the responses to the 75 questions in our survey to develop three sets of data: the importance of various topics taught in computer science, software engineering, and computer engineering curricula, the emphasis educational institutions place on these topics, and what practitioners believe they currently know about the topics.

Our survey reinforces current perceptions about the importance of some topics, but it also highlights topics that are sometimes underemphasized or overemphasized. For example, the survey results indicate that education programs emphasize mathematics, chemistry, and physics more than their importance to practitioners seems to warrant; furthermore, practitioners tend to forget this material. On the other hand, there is a clear knowledge gap and a reliance on on-the-job learning for topics related to software processes, people skills, and human-computer interaction.

The sources of information about the knowledge that is important to software engineering include the latest technical literature, existing educational programs and licensing requirements, and the overviews of the field that experts are developing.

This survey incorporates significant improvements in methodology, questions, and sampling compared to our original survey on the importance of software engineering topics conducted in 1997.[1,2] Additional data from the current survey has been published elsewhere[3] and is also available at http://www.site.uottawa.ca/~tcl/edrel/.

The results of this survey should be useful to licensing and accreditation bodies, corporate training departments, and curriculum designers in universities, colleges, and training institutes. Students and professionals seeking continuing education will also be able to use the data to help select courses.

THE SURVEY

We recruited participants for the survey by directly approaching companies and by advertising on the Internet. Most participants used a Web-based form to complete the survey, although a few used a paper version. The participants indicated that it took slightly more than one-half hour to answer all the questions.

Participants were asked questions about 75 topics we selected by examining university curricula and the initial SWEBOK proposals. We listed all the topics

0018-9162/00/$10.00 © 2000 IEEE

found in compulsory courses in various computer science, software engineering, and computer engineering curricula. We also listed the most common types of elective courses and some material that participants in earlier surveys recommended that we include. We did not intend for the list of topics to correspond with a course list; some topics might only be components of courses, while others might be spread over many courses.

We tested and revised the survey several times before starting to gather the 1998 data. To help reduce bias, we used different versions of the survey, each with the topics in a different order.

Figure 1 shows the precise wording for the four questions the participants answered about each topic. Question 1 asked how much the participants learned about a topic during their formal education, and question 2 assessed their level of knowledge at the present time. The difference between the answers to these two questions indicates how much participants have either learned on the job (if positive) or forgotten (if negative).

Question 3 asked how useful the participants have found the specific details of the material to be in their careers, and question 4 evaluated the influence of the material both professionally and personally. We compute a topic's importance by averaging the responses to questions 3 and 4.

The participants

We received survey responses from 186 participants with a wide variety of backgrounds. The sample appears to provide balanced coverage of a wide spectrum of software professionals, with a bias toward those in North America—and possibly toward those who were interested enough to take the time to participate.

Participants represented 24 countries; 54 percent were from the US, and 23 percent were from Canada. Participants work on software in a variety of industries; only 42 percent indicated that software is their company's primary product.

Fifteen percent of the participants had only a high school or college level education, while 48 percent had received a bachelor's degree; the remaining 37 percent had postgraduate degrees. More than 60 percent of the participants had degrees in computer science, software engineering, or information systems; 50 percent had degrees in other areas of science and engineering, while 20 percent had degrees in other disciplines. These percentages total more than 100 percent because many participants had degrees in more than one area.

SURVEY RESULTS

We present the data in two complementary formats: as categorized lists of topics, and as graphs depicting

Question 1. How much did you learn about this in your formal education (e.g. **University or College**)?

0 = *Learned nothing* at all
1 = Became *vaguely familiar*
2 = Learned the *basics*
3 = Became *functional* (moderate working knowledge)
4 = Learned *a lot*
5 = Learned *in depth*; became *expert* (learned almost everything)

Question 2. What is your **current knowledge** about this, considering what you have learned on the job as well as forgotten?

0 = *Know nothing*
1 = Am *vaguely familiar*
2 = Know the *basics*
3 = Am *functional* (moderate working knowledge)
4 = Know *a lot*
5 = Know *in depth*/am *expert* (know almost everything)

Question 3. How useful have the details of this **specific material** been to you in your career as a software developer or software manager? Please leave blank if you know little about the material.

0 = Completely *useless*
1 = *Almost never* useful
2 = *Occasionally* useful
3 = *Moderately useful*, but perhaps only in certain activities
4 = *Very useful*
5 = *Essential*

Question 4. How much **influence** has learning the material had **on your thinking** (your approach to problems and your general intellectual maturity), **whether or not you have directly used the details** of the material? Please consider influence on **both your career and other aspects of your life**. Please leave blank if you know little about the material.

0 = *No influence* at all
1 = *Almost no* influence
2 = *Occasional* influence
3 = *Moderate* influence in some activities
4 = *Significant* influence in many activities
5 = *Profound* influence on almost everything I do

Figure 1. The four questions asked about each of the 75 topics. Keywords were emphasized to ensure that participants noticed them even if they read quickly.

the 25 most important and 25 least important topics. Tables 1 through 3 and Figure 2 show information about each topic's importance, the amount learned in formal education programs, and the amount learned (or forgotten) subsequent to education. We divided the topics in the survey into categories to facilitate locating topics and identifying clusters of topics where the responses were similar. In Tables 1 through 3, the solid red square means the topic is in the top quartile for a data set, while the red half-square means it is in the bottom quartile. Similarly, the solid blue circle means it is one of the top four topics, and the blue half-circle means it is one of the bottom four topics.

Overall importance

The connected line in Figures 2a and 2b indicates the overall importance of topics. Figure 2a shows that the respondents consider specific programming languages to be the most important topic, with data structures close behind, followed by several other software design topics. The tables include several clusters of topics with similar levels of importance. The most significant is general software design, followed by software engineering methods, software management, and essential subsystem design.

Table 3 shows that the business, science, and arts categories include another cluster of important topics.

Table 1. Core software engineering topics.

		Top (🟥) and bottom (⬓) quartile and top (●) and bottom (◒) four topics, in terms of:		
Category	**Topic**	**Overall importance (Q3 + Q4)**	**Learned in education (Q1)**	**Learned on the job (or forgotten since education) (Q2 – Q1)**
General software design	Data structures	●	🟥	
	Algorithm design	🟥	🟥	
	Software design and patterns	●		🟥
	Software architecture	●		
	Object-oriented concepts and technology	🟥		🟥
	Specific programming languages	●	●	
Software engineering methods	Requirements gathering and analysis	●		🟥
	Formal specification methods			
	Analysis and design methods	🟥		🟥
	Performance measurement and analysis			
	Testing, verification, and quality assurance	🟥		●
	Software reliability and fault tolerance			🟥
	Maintenance, reengineering, and reverse engineering		⬓	●
Software management	Project management	🟥		●
	Software metrics		⬓	
	Software cost estimation		⬓	🟥
	Configuration and release management	🟥	◒	●
	Process standards such as CMM, ISO9000		◒	🟥
Essential subsystem design	Human-computer interaction/user interfaces	🟥		🟥
	Databases	🟥		🟥
	File management			
Specialized application techniques	Computational methods for numerical problems		🟥	⬓
	Simulation			
	Artificial intelligence	⬓		
	Pattern recognition and image processing	⬓	⬓	
	Computer graphics	⬓		
	Parsing and compiler design			
	Information retrieval			
	Security and cryptography		⬓	

Table 2. Scientific topics other than core software engineering.

		Top (🟥) and bottom (⬓) quartile and top (●) and bottom (◒) four topics, in terms of:		
Category	Topic	Overall importance (Q3 + Q4)	Learned in education (Q1)	Learned on the job (or forgotten since education) (Q2 – Q1)
Real-time and systems programming	Operating systems	🟥	🟥	
	Systems programming			
	Data transmission and networks			
	Parallel and distributed processing			
	Real-time system design		⬓	
Computer hardware	Digital electronics and digital logic		🟥	⬓
	Microprocessor architecture			
	Computer system architecture		🟥	
	Network architecture and data transmission			
	Telephony and telecommunications		⬓	
Other electrical and computer engineering	Analog electronics	⬓		⬓
	Digital signal processing	⬓	⬓	
	Data acquisition		⬓	
	Robotics	◒	⬓	
	VLSI	◒	⬓	
Computer science theory	Programming language theory		🟥	
	Formal languages		🟥	⬓
	Computational complexity and algorithm analysis		🟥	
	Information theory	⬓		
Discrete mathematics	Predicate logic		🟥	⬓
	Set theory		🟥	⬓
	Graph theory	⬓		⬓
	Automata theory			⬓
	Queuing theory	⬓		
	Combinatorics	⬓		⬓
Probability and statistics			●	⬓
Linear algebra and matrices			●	◒
Continuous mathematics	Differential and integral calculus	⬓	●	◒
	Differential equations	◒	🟥	◒
	Control theory	⬓		⬓
	Laplace and Fourier transforms	⬓		⬓
Natural science	Physics	⬓	🟥	⬓
	Chemistry	◒	🟥	◒

This cluster includes people skills, technical writing, and ethics and professionalism, the latter being the most important nontechnical topic.

The least important category is continuous mathematics, followed by electrical and computer engineering and natural science.

Amount learned in education

The responses to Question 1 provide the amount learned in education data. The most extensively taught topic is specific programming languages. Table 1 includes this topic in a cluster of heavily taught topics in general software design. Most of the other extensively taught topics are found in Table 2 in the computer science theory and mathematics categories. The participants learned the least about software management, business, and people skills.

The amount learned data suggests the importance that educational institutions give to the various topics. We can compare this to the participants' judgments about each topic's importance to discover topics that might be overtaught or undertaught. Making absolute numerical comparisons between these two scales is not appropriate because different interpretations are given to points on each scale; nevertheless, we can compare relative differences.

Clearly, specific programming languages is both the most important and most learned topic. Tables 1 and 2 show that only a few other topics in the survey—for example, algorithm design and operating systems—are high on both scales. Many of the remaining highly important topics are not extensively taught, and some unimportant topics are extensively taught. This suggests that the education that today's computing professionals receive may not be entirely appropriate.

We define the educational knowledge gap as the difference between the amount learned in education and each topic's importance. The widest educational knowledge gaps occur in topics such as configuration and release management, negotiation, human-computer interaction/user interfaces, and leadership. Universities might consider increasing their coverage of such topics; corporate trainers might give new hires courses on these topics, expecting that they will be underprepared.

Figure 2b shows that ten topics—including calculus, differential equations, linear algebra, chemistry, and physics—have a negative educational knowledge gap, suggesting that they might be overtaught.

It seems that the overtaught topics either should be taught less or they should be taught so that the students are more likely to remember the material and perhaps apply it later on. For example, teaching mathematics and physics in conjunction with programming exercises might be helpful. Another suggestion is to shift the emphasis in mathematics away from continuous mathematics toward discrete mathematics and statistics.

Amount learned on the job

The amount learned on the job (or forgotten since education) is the difference between the responses to Questions 1 and 2. We can attribute learning since education—or on-the-job learning—to both training and experience performing work. Table 1 shows that the greatest on-the-job learning occurs in the software process category, especially in configuration and release management, project management, maintenance and reengineering, and testing, verification, and quality assurance. Topics with high on-the-job learning might be targets for new-hire training and increased university coverage, especially if they also have a high educational knowledge gap.

The respondents indicated that they have forgotten the most about theory and mathematics as well as natural science. This bolsters our argument for reexamining coverage and teaching methods for these topics to improve the educational investment return.

Amount currently known

For many topics, the amount currently known correlates well with importance, suggesting that practi-

Table 3. Categorized list of business, science, and arts topics.

		Top (🟥) and bottom (▄) quartile and top (●) and bottom (◒) four topics, in terms of:		
Category	**Topic**	**Overall importance (Q3 + Q4)**	**Learned in education (Q1)**	**Learned on the job (or forgotten since education) (Q2 – Q1)**
Business	Economics			
	Accounting		▄	
	Marketing		▄	
	Management			
	Entrepreneurship		◒	
Psychology and philosophy	Psychology			
	Philosophy			
	Ethics and professionalism	🟥		🟥
Technical writing		🟥		🟥
People skills	Giving presentations to an audience	🟥		🟥
	Leadership	🟥	▄	🟥
	Negotiation		◒	🟥
Second language other than English		▄		▄

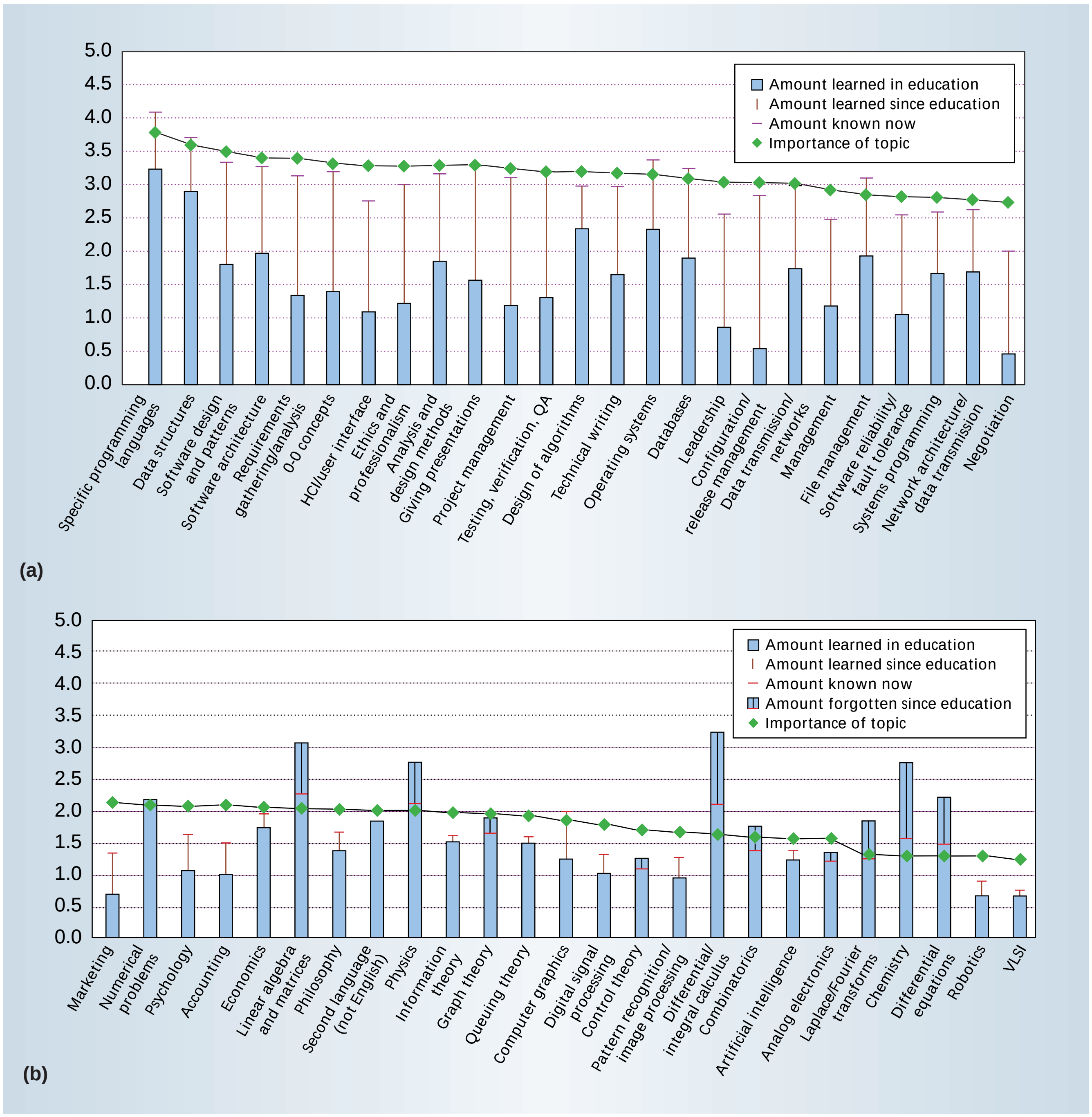


Figure 2. The 25 most important topics (a) and the 25 least important topics (b).

tioners eventually learn what is important, but not necessarily while attending a university or college.

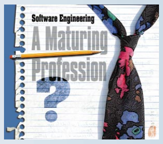
We define the current knowledge gap as the difference between the importance of a topic and the amount currently known. Table 4 lists the top 10 topics that have both a relatively large current knowledge gap and importance above some threshold. Using an importance threshold of 2.5 (the halfway point on the scale), the greatest current knowledge gap exists for topics such as negotiation, human-computer interaction, and leadership. Corporate training departments should consider giving additional courses to employees in these topics.

A variety of research efforts are trying to define exactly what topics constitute the software engineering discipline. The "Additional Information about Software Engineering Topics" sidebar provides a list of some of these efforts.

Some experts contend that software engineers, like all other engineers, ought to learn about chemistry, physics, and continuous mathematics. While some software engineers would benefit from learning this material, our survey shows that considering these topics to be essential is clearly a mistake.

Because of the low importance and high forgetability of continuous mathematics and basic science, uni-

Additional Information about Software Engineering Topics



- The IEEE's *Guide to the Software Engineering Body of Knowledge* (http://www.swebok.org) outlines and categorizes the SE body of knowledge. The Software Engineering Coordinating Committee drafted the SWEBOK guidelines after consulting a wide variety of sources, including many textbooks as well as experts from industry and academia. This committee is soliciting feedback from all parties, and they plan to revise the proposal several times during the next few years. The SWEBOK project also provides good references to related research.
- University software engineering curricula are another place to look. The IEEE Draft Software Engineering Curriculum is an important example.[1] The Working Group on Software Engineering Education has produced an even more extensive set of guidelines for software engineering programs.[2,3] They categorize topics as core (for example, requirements, design, project management), foundations (for example, computing principles and human-computer interaction), recurring themes (for example, quality, metrics, tools, and ethics), and supporting topics (for example, mathematics, business, and science).
- The Forum for Advancing Software Engineering Education's electronic newsletter frequently includes information on SE topics (http://www.cs.ttu.edu/fase).

References

1. "Draft Software Engineering Accreditation Criteria," *Computer,* Apr. 1998, pp. 73-75, 77.
2. T.B. Hilburn et al., "Software Engineering across Computing Curricula," *Proc. 3rd Ann. Conf. Integrating Technology into Computer Science Education*, ACM Press, New York, 1998, pp. 117-121.
3. D.J. Bagert et al., *Guidelines for Software Engineering Education*, tech. report CMU/SEI-99-TR-032, Carnegie Mellon Univ./Software Eng. Inst., 1999, http://www.sei.cmu.edu/topics/collaborating/ed/workgroup-ed.html.

versities and colleges should either place less emphasis on these topics or they should teach them in a way that makes them more relevant to software engineering students. Conversely, the large amount of on-the-job learning—and greater importance relative to amount known—suggest that educational institutions should place considerably more emphasis on teaching topics such as people skills, software processes, human-computer interaction, real-time system design, and management.

Our survey indicates that employees are likely to lack skills and knowledge in areas such as negotiation, leadership, and human-computer interaction. Therefore, each company might consider conducting its own version of this survey to discover its particular needs. ✷

Acknowledgments

This research was supported by the Consortium for Software Engineering Research. The author thanks the anonymous participants and companies, as well as others who helped with the research: Nancy Mead, W. Michael McCracken, Laurie Werth, Lawrence West, Lesley Beddie, Michael Lutz, Pearl Brereton, Anatol Kark, and K. Teresa Khidir.

Table 4. Topics with the greatest knowledge gap—where importance most exceeds current knowledge.

Rank	Topic
1	Negotiation
2	Human-computer interaction/user interfaces
3	Leadership
4	Real-time system design
5	Management
6	Software cost estimation
7	Software metrics
8	Software reliability and fault tolerance
9	Ethics and professionalism
10	Requirements gathering and analysis

References

1. T. Lethbridge, "The Relevance of Software Education: A Survey and Some Recommendations," *Ann. Software Eng.*, Dec. 1998, pp. 91-110.
2. T. Lethbridge, "A Survey of the Relevance of Computer Science and Software Engineering Education," *Proc. 11th Conf. Software Eng. Education and Training,* CSEE&T 98, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 56-66.
3. T. Lethbridge, "Priorities for the Education and Training of Software Engineers," to be published in *J. Systems Software*, 2000.

Timothy C. Lethbridge has been involved in the development of undergraduate software engineering programs at the University of Ottawa. His research interests include user interfaces, software engineering tools, and knowledge representation. Lethbridge received a PhD in computer science from the University of Ottawa. He is a member of the IEEE, the IEEE Computer Society, and the ACM. Contact him at tcl@site.uottawa.ca.

E-mail accounts on overload?

A free e-mail alias from the Computer Society forwards all your mail to one place.

you@computer.org

Sign Up Today at
computer.org/WebAccounts/alias.htm

IEEE Computer Society
computer.org

COMPUTER
Innovative technology for computer professionals