A Survey of the Relevance of Computer Science and Software Engineering Education

Timothy C. Lethbridge
School of Information Technology and Engineering (SITE)
150 Louis Pasteur, University of Ottawa, K1N 6N5, Canada
tcl@site.uottawa.ca

Abstract

We describe a study of 168 software professionals to determine how relevant their education has been to their careers. Starting with a list of 57 topics, we asked the participants to indicate how much they learned in university, how much they know now, how useful the material has been and whether they would like to learn more. We conclude from the results that certain software engineering topics should be given more emphasis, while the emphasis on certain mathematics topics should be changed.

A Survey of the Relevance of Computer Science and Software Engineering Education[1]

Timothy C. Lethbridge
School of Information Technology and Engineering (SITE)
150 Louis Pasteur, University of Ottawa, K1N 6N5, Canada
tcl@site.uottawa.ca

Abstract

We describe a study of 168 software professionals to determine how relevant their education has been to their careers. Starting with a list of 57 topics, we asked the participants to indicate how much they learned in university, how much they know now, how useful the material has been and whether they would like to learn more. We conclude from the results that certain software engineering topics should be given more emphasis, while the emphasis on certain mathematics topics should be changed.

1. Introduction

During the summer and early fall of 1997 we conducted an international survey of software developers, where we asked them a variety of questions about their education, and how useful that education has been.

Two important objectives of this research were:

1. To understand those areas where practitioners feel they need more or better education, and thus provide information to educational institutions and companies as they plan curricula and training programs.
2. To provide data that will assist in the evaluation of existing and proposed curricula.

2. Survey methodology

In this section we outline how we conducted the survey. We first explain the questions that were asked; we then discuss the sampling process and the validity of the survey.

2.1 The questions

At the beginning of the survey were a series of demographic questions that allowed us to determine: 1) The educational background of the participants; 2) the cumulative number of years (including co-op jobs) during which they have worked in the software industry; 3) the type of software they work with, and 3) the job functions they perform.

The main body of the survey asked participants to answer questions about a total of 57 topics.

The topics were grouped into five general curriculum categories as follows: 9 mathematics topics; 31 software topics; 4 engineering topics (mostly computer hardware),

[1] This work is supported by NSERC and sponsored by the Consortium for Software Engineering Research (CSER).

and 13 miscellaneous topics including natural and social science, and business. Participants were also given several blank areas to add unlisted topics that they thought to be important.

We obtained the list of topics in the following manner: We first scanned computer science and software engineering curricula (including [1, 2, 3, 4,]), and created an initial list of topics that are taught in most programs. We then divided some topics from this initial list so we could obtain more detailed information. For example, instead of asking about the general topic 'software engineering', we asked about specific subtopics such as 'configuration and release management', 'software metrics', etc. We modified the list to make sure it covered most of the proposed IEEE/ACM software engineering body of knowledge for software engineers [5]. Finally, we ran a pilot study that brought to light several additional topics.

For each topic we asked the participants to answer four key questions (Figure 1). Each of these was to be answered on a scale from 0 to 5. We were careful to ensure that each value on the scale had a well-defined meaning, and that the high and low values were true extremes.

i. How much did you learn about this at **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)	ii. 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)
iii. **How useful has this specific material been** to you in your career? **0**=Completely useless **1**=Almost never useful **2**=Occasionally useful **3**=Moderately useful, but perhaps only in certain activities **4**=Very useful **5**=Essential	iv. **How useful would it be** (or have been) **to learn more** about this (e.g. additional courses)?: **0**=Pointless learning more **1**=Very unlikely to be useful **2**=Possibly helpful **3**=Moderately helpful **4**=Important to learn more **5**=Critical to learn more

Figure 1: Scales for the four questions asked about each topic.

The first question sought information about how much each participant had learned in university or college. The second question, using the same scale, asked how much the person knows *now*. Presumably for some topics, participants would have learned more since completing their education; whereas for other topics, they would have forgotten material.

The third question asked how useful the topic has been in each participant's career. We expected to obtain strong agreement that certain topics are important or unimportant.

The final question asked how useful it would be for the participant to learn more about the material. We expected this to tell us much the same information as question 3, but we hypothesized that topics that are more currently needed would rate highly in this question.

The survey took the average participant 35 minutes to complete.

2.2 The sampling process

This section explains the physical format of the survey and how we distributed it.

There were two forms of the questionnaire: Paper-based (44 usable responses) and an Internet-based html form (124 usable responses). We felt it was important to use both means of distribution since some people do not have access to the Internet (e.g. due to company policies). Also, paper forms may be more suitable for busy people who want to complete the survey while, for example, commuting.

The survey was conducted from May to October 1997. We successfully sought the participation of six companies to distribute the survey among their employees. In several cases we received enthusiastic support for the survey from management, however we approached several other companies without success.

We also publicized the availability of the survey electronically using Internet mailing lists and newsgroups. We obtained 42 of the responses in this manner.

A cover letter was distributed with both versions of the survey. This made it clear that we sought participants who worked in the software industry in any capacity from programmer to manager. The letter also made it clear that details would be kept confidential.

We received six responses that we could not use because they were completed incorrectly.

2.3 Validity

We attempted to obtain a sample that would represent the general software development community. We were successful in obtaining responses from people who varied widely according to such parameters as type of education, years of experience, and work function.

However, there are five areas where we believe there may be biases in our survey.

• **Real-time and embedded software developers:** Much of our support came from high-technology software companies (especially in the telecommunications sector), but very little came from the public sector or from companies whose business is not primarily software development (but who nevertheless develop software for in-house use). This led to a bias towards people whose primary work is real-time or embedded software.

• **Postgraduate degrees:** Support from high-tech companies also led, we believe, to a bias towards people with postgraduate degrees. There were 55 responses from such people (33%), which is clearly higher than their proportion of the general software profession.

• **North America:** 74% of participants work in Canada, 23% in the USA, and only 3% in other countries. The survey should thus be considered primarily descriptive of the North American experience. For seven respondents,, we were not able to determine where they worked.

• **Feelings vs. actual usefulness:** Since our survey is seeking subjective opinions, there may be systematic biases that result in some topics being under-rated or over-rated. For example, respondents in general may dislike some subject and thus downplay its usefulness. Alternatively, the topic might not have proved directly useful to the software developer, but might have contributed to valuable problem-solving skills that the participant does not recognize. We believe this type of bias will have minimal effect on the overall rankings of the topics, since there were very significant differences between high-ranked and low-ranked topics. We suggest follow-up studies that try to objectively measure the use of particular subject matter by working software professionals.

• **Self selection:** We circulated many requests for people to complete the survey, and only received responses from a small percentage. This was true within companies as well as in the wider Internet distribution. People who completed the survey thus represent a special sub-population whose members are likely to have more free time, or an intellectual interest in this kind of data. To combat this bias, we were able to find several corporate executives to champion the survey and thus give participants a sense that completing the survey is a part of their work.

For the first two types of bias above, we had enough data to perform separate analyses of sub-samples (e.g. developers of real-time software vs. developers who did not work with real time software). In general, the results for such sub-samples did not differ significantly from results from the sample as a whole.

3. Results

3.1 Basic demographics

Geography: Of 165 respondents who gave this information, 123 (75%) work in Canada, 37 (22%) in the USA and 5 (3%) in other countries.

122 (73% of 168 who answered) had received higher education in Canada, 34 (20%) in the USA, 18 (11%) in Europe and 10 (6%) in other countries. Some had received education in more than one region.

Type of education and experience: 12 of 167 (7%) had at most college (non-university) education. For 100 (60%) their highest degree was a bachelors' degree; for 45 (27%) it was a masters degree, and 10 (6%) reported having a Ph.D.

Table 1 shows the field of study of the participants (from the 159 who reported this information). Where a participant reported more than one field of study, the table reflects only that field closest to computer science or software engineering (closest to the top of the table).

Computer Science or Software Engineering[2]	80	50%
Computer Engineering or Electrical Engineering	47	30%
Other Engineering	7	4%
Other Science (including mathematics)	19	12%
Other subjects (typically business or arts)	6	4%

Table 1: Fields of study of participants in the survey.

The earliest degree had been received in 1962, the latest in 1997 (117 respondents gave this information). On average it had been 11.5 years (standard deviation 7.6) since participants completed their first degree, and 9.6 years since their last degree.

Participants reported that they had worked in the software industry between 0 and 40 years (n=165, mean=10.5, standard deviation=7.7). It is interesting to note from these figures that the average time since receiving their latest degree is less than the average time working in the industry: Many participants returned to university to update their education at some point.

[2] Few undergraduate software engineering programs have yet produced graduates, however some respondents have a masters degree in software engineering, or have completed a software engineering option in a computer science degree.

For the purposes of subsequent analysis, we divided subjects into junior, intermediate and expert based on their years in the industry as shown in table 2.

Years of experience	Category		
0-4	Junior	46	28%
5-11	Intermediate	60	36%
≥12	Expert	59	36%

Table 2: Categorizations of participants based on numbers of years they have worked in the software industry.

Type of work: Participants were asked to state which of three categories of software they were involved with; results from the 149 who responded are in table 3. Thirty-seven respondents reported that they worked with more than one category, and 10 respondents listed software types outside these categories..

Real-time, telecommunications or embedded software	115	77%
Management information, in-house or Internet-based software	51	34%
Consumer or mass-market software	20	13%

Table 3: Type of software developed by participants. Multiple responses to this question were allowed.

Participants were also asked to describe their work. They were given several keywords, of which they could chose several and/or add their own. From the 167 responses we were able to divide participants into the categories described in table 4.

Developers	Reported some programming, and/or maintenance and/or general development (and possibly other activities)	133	80%
Managers	Reported management, (and possibly other activities)	57	34%
Developer-Managers	Developers ∩ Managers	37	22%
Pure Developers	Developers minus Managers	96	57%
Pure Managers	Reported management only	14	8%
Maintainers	Reported some maintenance	76	46%
Analysts	Analyzed requirements but did not do development	6	4%
Pure Testers	Nothing but testing	6	4%

Table 4: Intersecting categories of software workers identified in the survey.

3.2 General relevance of education

The survey asked two very general questions about the relevance of participants' education. These were added following the pilot study. Several respondents to that study told us that they had marked 'almost never useful' for many specific subjects, but that overall their education had proved useful. The questions were:

• Considering your university or college education taken as a whole, how relevant has it been to your career (0=Completely irrelevant; 3=Relevant at times; 5=Extremely relevant)?

• How important were specific subjects and details in your education, as opposed to learning how to think or approach problems (0=The specific subjects I learned were much more important; 2.5=Both were equally important; 5=learning how to think was much more important)?

The mean responses were 3.5 to the first question and 3.7 to the second; however there was a wide range with some people responding zero and others five.

For the first question, we can thus draw the conclusion that software practitioners consider their education to be moderately relevant, but with big differences of opinion.

For the second question, we can conclude software developers feel that learning how to think is somewhat more important than learning specific subjects. Maybe we should not therefore be overly concerned with the minutiae of curriculum design, but instead ensure that courses are stimulating and cover a wide variety of problems.

Differences among demographic sub-groups: We divided the participants based on whether they had responded ≥4 or <4 to the above questions. This provided an approximately even split for each question.

Fifty-one percent of respondents said they felt their education was relevant (response to the first question was ≥4) while the rest were less sure about this. However, there were some interesting differences depending on the demographics of the respondents to this question:

• Respondents educated in the US were significantly more likely to find their education relevant (65% found it relevant). Only 49% of those educated in Canada and 48% of those educated elsewhere found it relevant. One thus might conclude that one should look to US universities for examples of effective programs. It seems however, that there is a hidden bias towards the more élite universities in the sample: There were more respondents from universities such as Stanford and MIT than in the general software developer population.

• As would be expected, those with computer science or software engineering education were far more likely to report it relevant (70%). Interestingly, those with computer or electrical engineering backgrounds found their education significantly less relevant to their software careers (30%). Intuitively, one might expect this since their education contains more hardware than software, however many companies preferentially hire these people for software jobs, perhaps because they are *engineers*. This presents an argument in favor of new software engineering programs.

• As one would suppose, those with non-computer backgrounds were far less likely (29%) to find their education relevant.

• Junior respondents found their education less relevant (43%) than experts (56%). Possible explanations are: 1) more senior people might have had the chance to work on a wider variety of projects which, taken cumulatively, make use of a greater fraction of their education, or 2) education might be becoming less relevant. We prefer the first explanation.

56% of respondents felt that learning how to think (≥4) was more important than specific details. The following are some interesting demographic differences for this question:

• Those educated outside North America were significantly less likely (44%) to consider 'learning how to think' to be of prime importance. This might reflect the tendency of North American universities (especially in the USA) to require a wider diversity of subjects (i.e. 'liberal arts' or 'complementary studies') in a students' education.

• Those with postgraduate educations thought that learning how to think was more important than the details (67%), as did those whose education was other than computer science or computer engineering (74%). The latter seems reasonable, since clearly these people would not have been able to learn many relevant (software) details in university.

• Those who develop real-time, telecommunications or embedded software put less value on learning how to think (49%), while those who developed other types of software put great value on it (78%). This might reflect the fact that developing consumer or business software requires one to learn about topics that one is typically not taught in university, whereas developing software that is tightly connected to the hardware draws heavily on the technical curriculum.

• Experts put more value into learning how to think (56%) than junior respondents (49%). One might attribute this to the fact that experts tend to work on more analytic problems that require more original thought. This is borne out by another statistic: 33% of junior respondents perform analysis as part of their work, whereas 76% of expert respondents perform analysis.

There was a very low correlation coefficient (-0.2) between the education relevance question and the question that asked about details vs. learning how to think. Those who thought their education was relevant were significantly less likely (46%) to favor 'learning how to think', than were people who thought their education was less relevant (65%).

i. How much did you learn?	
Mathematics	2.7 (< functional)
Software	1.8 (basics)
Other Engineering	2.0 (basics)
Other Topics	1.4 (< basics)

ii. What is your current knowledge?	
Mathematics	1.9 (basics)
Software	2.7 (< functional)
Other Engineering	1.8 (basics)
Other Topics	2.0 (basics)

iii. How useful has this been?	
Mathematics	1.5 (< occasionally)
Software	2.8 (moderately)
Other Engineering	1.7 (< occasionally)
Other Topics	1.8 (occasionally)

iv. How useful to learn more?	
Mathematics	1.5 (< possibly)
Software	2.9 (moderately)
Other Engineering	1.9 (possibly)
Other Topics	1.9 (possibly)

Table 5: Averages of the subtopic averages for the four major categories.

3.3 Comparison of the curriculum categories

Table 5 shows responses for each of the four main questions, and the four main curriculum categories. The following are key observations from this data:

• Participants felt that their university education gave them a much better grounding in mathematics than in software topics. One possible reason for this is that most of the math useful to software developers is actually taught in the curriculum, whereas many software engineering topics have not been taught until recently. See section 3.4 for more about this.

• As one would expect, software is the highest rated category in terms of usefulness, and desire to learn more. The software figures in table 5 are relatively low because they include several low-rated software subtopics (e.g. artificial intelligence).

Table 6 shows the differences between question i (what they learned) and question ii (what they know now) for the categories. It is clear that much mathematics is being forgotten, whereas much new software knowledge is being acquired on-the-job.

There are several significant differences among the demographic subgroups with respect to the above data. The following is a selection of points illustrating such differences, most of which confirm intuition:

Mathematics	-0.8
Software	0.9
Other Engineering	-0.1
Other Topics	0.6

Table 6: Question ii minus question i: How much has been learned or forgotten?

• Those with postgraduate degrees learned more mathematics than others (2.9 vs. 2.6), and found it more useful in their careers (1.8 vs. 1.3).

• Expert participants believed that their software education had been significantly more useful to them than junior participants (2.9 vs. 2.6), even though their perceived differences in current knowledge were quite close (2.6 vs. 2.5). Expert participants felt they had learned less in university about software (1.6 vs. 2.0). For other subjects junior and expert participants gave very similar responses.

• Managers in general reported learning more about topics and finding them more useful than non managers. In particular, managers had a wider knowledge and appreciation of the miscellaneous topics (including management): for question ii the means were 1.8 for non managers and 2.3 for managers, whereas for question iii the means for miscellaneous topics were 1.5 and 2.2 respectively.

• Those who exclusively develop mass-market, consumer or management information software reported far less knowledge about hardware, as compared with those who develop real-time, embedded or telecommunications software (1.3 vs. 1.9). The former group also found their hardware knowledge less useful (1.1 vs. 1.9). Interestingly, the former group is less likely to forget their mathematics knowledge (-0.5 vs. -0.9), although part of this can be attributed to the fact that they had less mathematical education to start with (2.3 vs. 2.8).

3.4 Details within the categories

Tables 7 through 10 show the subtopics perceived to be most and least useful within each category. Question iii is considered the best indicator of usefulness, followed by question iv. The columns marked 'learn' are the difference between questions i and ii. A strong positive value indicates that material is being learned on-the-job, and hence is apparently also useful; universities might therefore consider teaching these subjects in order to better prepare students.

Boldface in the tables highlight values that are near the extremes for the question.

Just because a topic ranks low in these tables does not imply that it should not be taught – for example, most educators believe the low-rated calculus is needed for a proper understanding of aspects of the higher-rated statistics. Also, this survey results reflect only *opinions* which may not reflect usefulness in practice. However, universities might consider increasing the emphasis on high-rated topics as opposed to low-rated topics, and reducing the level of detail or method of teaching of low-rated topics.

The following conclusions can be drawn from the data presented in tables 7 though 10:

• For the purposes of educating software professionals, there appears to be an imbalance in the emphasis on mathematics subtopics taught in most computer science and software engineering curricula. Calculus is usually given heavier weight than the topics listed as 'most useful' in table 7.

Most useful Math	iii	iv	Learn
Probability & statistics	2.2	1.9	-0.8
Predicate logic	1.9	1.6	-0.4
Set theory	1.8	1.5	-0.8
Information theory	1.9	1.9	-0.2

Least Useful Math	iii	iv	Learn
Computational Geom.	***0.9***	1.2	**-0.4**
Differential Equations	***0.9***	***0.9***	***-1.3***
Calculus	1.1	1.0	***-1.4***

Table 7: The most useful (top left) and least useful (top right) out of 9 mathematics subtopics.

Most useful Software	iii	iv	Learn
General architecture & design	***4.3***	***3.9***	1.0
Data structures	***4.1***	3.2	0.5
Testing & quality assurance	***3.7***	***3.8***	***2.1***
Requirements gathering and analysis	***3.7***	***3.8***	***1.6***
Operating systems	***3.5***	**3.4**	0.6
Project management	***3.5***	***3.7***	***1.9***
Data transmission	***3.5***	***3.7***	1.1
Real-time software	***3.4***	***3.5***	1.5
Object oriented analysis and design	3.3	***3.8***	1.2
Configuration mgmt.	3.3	3.2	***2.4***
File and information management	3.2	2.8	0.6
User interface design	3.2	***3.4***	***1.7***
Maintenance	3.2	3.1	***2.0***

Least Useful Software	iii	iv	Learn
Artificial intelligence	***1.0***	***1.5***	***-0.1***
Pattern recognition	***1.0***	***1.4***	0.2
Graphics	1.5	1.7	0.4
Numerical methods	1.8	1.9	***-0.7***
Simulation	2.1	2.3	0.2

T able 8: The most and least useful out of 31 software subtopics.

Most useful Other Engineering	iii	iv	Learn
Computer architecture	2.9	2.8	0.2

Least Useful Other Engineering	iii	iv	Learn
Robotics	***0.6***	***1.0***	0.0
Analog electronics	1.3	1.4	***-0.5***

T able 9: The most and least useful out of 4 hardware subtopics.

Most useful Other	iii	iv	Learn
Technical writing	***3.6***	***3.5***	***1.6***
Ethics / professionalism	3.0	3.0	***1.6***
Management	2.8	3.2	***1.6***
Second language	2.1	2.6	0.8

Least Useful Other	iii	iv	Learn
Chemistry	**0.6**	**0.7**	**-0.8**
History	**0.9**	1.1	0.7

T able 10: The most and least useful out of 13 miscellaneous subtopics.

• It would appear from table 7 that certain topics which are either electives or a minor component of the curriculum ought to be emphasized far more. Such topics include testing and quality assurance, requirements gathering and analysis, project management, user interface design and configuration management.

• Conversely the data provides evidence that it would be reasonable to drop certain topics which are compulsory in many programs, but over which there is some disagreement about whether they should be included. Such subjects include chemistry, differential equations and numerical methods.

• Software practitioners are extremely eager to learn more about software architecture and design: They feel their education did not provide enough of this material. Both universities and corporate training departments ought to consider providing further courses in this topic.

4 General conclusions

The data presented in this paper lead us to believe that there is considerable room for improvement in what is taught to software students. Although the data we have presented represent opinions, there is considerable consistency among various sub-samples; this suggests that the same opinions are widely held. Also, the opinions are those of software developers with experience applying the material they have learned in university, not just students who have recently finished courses.

We have used the results of this study to help design and validate our own new software engineering program [6] at the University of Ottawa[3], the first such program in Canada. The following are some important features of the program that have been influenced by the results of this survey:

• All but one of the topics identified as most important in tables 7 through 10 will be taught as required topics in the software engineering program. The only exception is the second language. Of these 'important' topics, the following are not normally compulsory in typical computer science programs: Project management, real-time systems, user interface design, maintenance (includes reengineering), management, and ethics and professionalism.

• We have explicitly made certain topics optional that were found not to be important according to the survey; in particular numerical methods and differential equations. Certain other topics that were found to be of low importance (e.g. chemistry and calculus) have been retained for various reasons including conforming to standard engineering curriculum guidelines, as required for accreditation.

We hope that the data reported here will be useful to others designing university curricula or corporate training programs. Further data from this survey, along with additional recommendations for curriculum change, can be found in [8].

Acknowledgments

We would like to thank all those who completed questionnaires. We would also like to thank Anatol Kark of the National Research Council of Canada who prepared the electronic version of the survey, Maryam Banaei who helped circulate the questionnaires in one particular company, K. Teresa Khidir who helped organize distribution, performed the data entry and provided useful comments, and Nicolas Anquetil who also gave valuable criticism.

[3] Our new program replaces the option that was reported in last years' proceedings [7].

References

1. G. Ford, (1996) "The SEI Undergraduate Curriculum in Software Engineering", Software Engineering Institute.
2. D. Parnas (1997) "Draft: Software Systems Engineering", McMaster University.
3. J. Fernando Naveda (1997) "Crafting a Baccalaureate Program in Software Engineering", 10th SEI Conference on Software Engineering Education, Virginia Beach, pp. 74-79.
4. List of Undergraduate Software Engineering Programs at http://ricis.cl.uh.edu/virt-lib/se_programs.html
5. Joint IEEE Computer Society and ACM Steering Committee for the Establishment of Software Engineering as a Profession; at http://computer.org/tab/seprof/
6. T. Lethbridge, B. Probert, D. Ionescu, D. Gibbons, J. Raymond, L. Orozco-Barbosa, S. Szpakowicz, (1997)"Proposal for a Software Engineering Program", School of Information Technology and Engineering, University of Ottawa.
7. T. Lethbridge, D. Ionescu, A. Mili and D. Gibbons, (1997) "An Undergraduate Option in Software Engineering: Analysis and Rationale", 10th SEI Conference on Software Engineering Education, Virginia Beach.
8. Lethbridge, T. (1997), "The Relevance of Software Education: A Survey and Some Recommendations", submitted to Annals of Software Engineering, November, 1997.