The Relevance of Software Education: A Survey and Some Recommendations[1]

Timothy C. Lethbridge

School of Information Technology and Engineering
150 Louis Pasteur, University of Ottawa
Ottawa, K1N 6V9 Canada
tcl@site.uottawa.ca

Abstract

We summarize the results of a survey of software practitioners in which we asked about the relevance of their education. We analyze the data and highlight potential changes to computer science or software engineering curricula, based on mismatches between the extent to which the topic is taught and its importance in the careers of the participants. We also present some advice to companies regarding topics in which they should train employees; we compare this advice to a list of skills that employers have indicated they wish employees to possess.

1. Introduction

During the summer of 1997, we conducted a survey of 168 software practitioners. The participants answered four questions about each of 57 topics which are typically found in the curriculum of computer science or software engineering programs. We asked i) how much each participant learned about the topic in their formal education, ii) how much he or she knows now, iii) how important the topic has been in his or her career and iv) how much desire he or she has to learn more. From this data we have formulated some specific recommendations about curricula that will be of interest to universities and colleges. We also present some recommendations that will be of interest to companies involved in training employees, and to students or employees themselves.

The survey was distributed in paper form as well as electronically on the Internet. 75% of the participants were actively solicited to complete the survey, either by their companies or directly by the author. The remainder responded to an advertisement posted to a variety of Internet newsgroups, and thus were self-selected.

The work location of the participants broke down as follows: 74% of the participants from Canada (65% from the Ottawa area), 23% from the USA, 3% from other countries, and the remainder unknown. This represents a significant source of bias; however when we separately analyzed the data from only non-Canadian participants, there were no important differences to the conclusions we draw in this paper.

Another significant source of sampling bias is that a disproportionate percentage of participants (77%) spent at least part of their time working on real-time or embedded software. This compares with 34% who report working on in-house management information systems (MIS), and 13% who report working on consumer or mass-market

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

software (multiple answers were possible for this question). This preponderance of real-time developers is largely because we were not able to obtain the co-operation of many MIS departments in non-software companies. Again, this source of bias appear unimportant: A differential analysis shows that most of our conclusions are the same for the real-time and non-real-time individuals (see section 4.1).

With respect to education, 33% of the participants had postgraduate degrees, 60% had bachelors degrees and the remainder had college diplomas. 50% were educated in computer science or software engineering, 30% in computer or electrical engineering and the rest in a variety of other fields.

In terms of expertise, 28% of the participants can be considered junior, with up to four years experience in the software industry. 36% can be considered intermediate, with 5-11 years; and another 36% can be considered expert with 12 or more years of experience. 34% of the participants have some management function, although only 8% perform this role exclusively.

See [1] for further discussion of sources of bias and the demographics of participants in this survey.

The list of topics for the survey came from three sources. We started with topics commonly found in computer science programs (including electives commonly taken by students). We then added topics found in software engineering curricula such as that proposed by the Software Engineering Institute [2] and that at Rochester Institute of Technology [3], the first undergraduate program in North America. We also considered the Joint IEEE/ACM draft set of knowledge units [4], however we found it was lacking some topics that were already on our list. We refined our list by performing a pilot study in which participants added additional topics that we had not considered.

Among the important findings from the survey are the following: Firstly, participants reported that certain topics currently heavily taught in university, such as calculus and numerical methods, appear of little direct use to them in their software development careers. Secondly, the data suggest that topics central to software engineering, such as project management, quality assurance, requirements gathering and human-computer interaction, are under-emphasized in curricula.

The remainder of this paper is organized as follows: Section 2 presents an analysis of software topics in the entire sample, highlighting such issues as where knowledge is highest or lowest, and which topics are considered most or least important. Section 3 extends the analysis to other topics commonly studied by computing students. Section 4 then divides the sample so as to look at specific demographic subgroups, specifically real-time developers and managers. Section 5 then compares data collected in this survey of practitioners with the opinions of employers about what they seek in a prospective employee.

A word about statistical significance: For the sake of readability, we have generally restricted the data given below to mean responses to the questions. Standard deviations are uniformly in a range of 1.1 to 1.6 on scales of 0 to 5 (when the whole sample is under consideration). This means that, except for sampling bias towards real time systems, each mean statistic given would be within about ± 0.12 of the true population mean, 19 times out of 20.

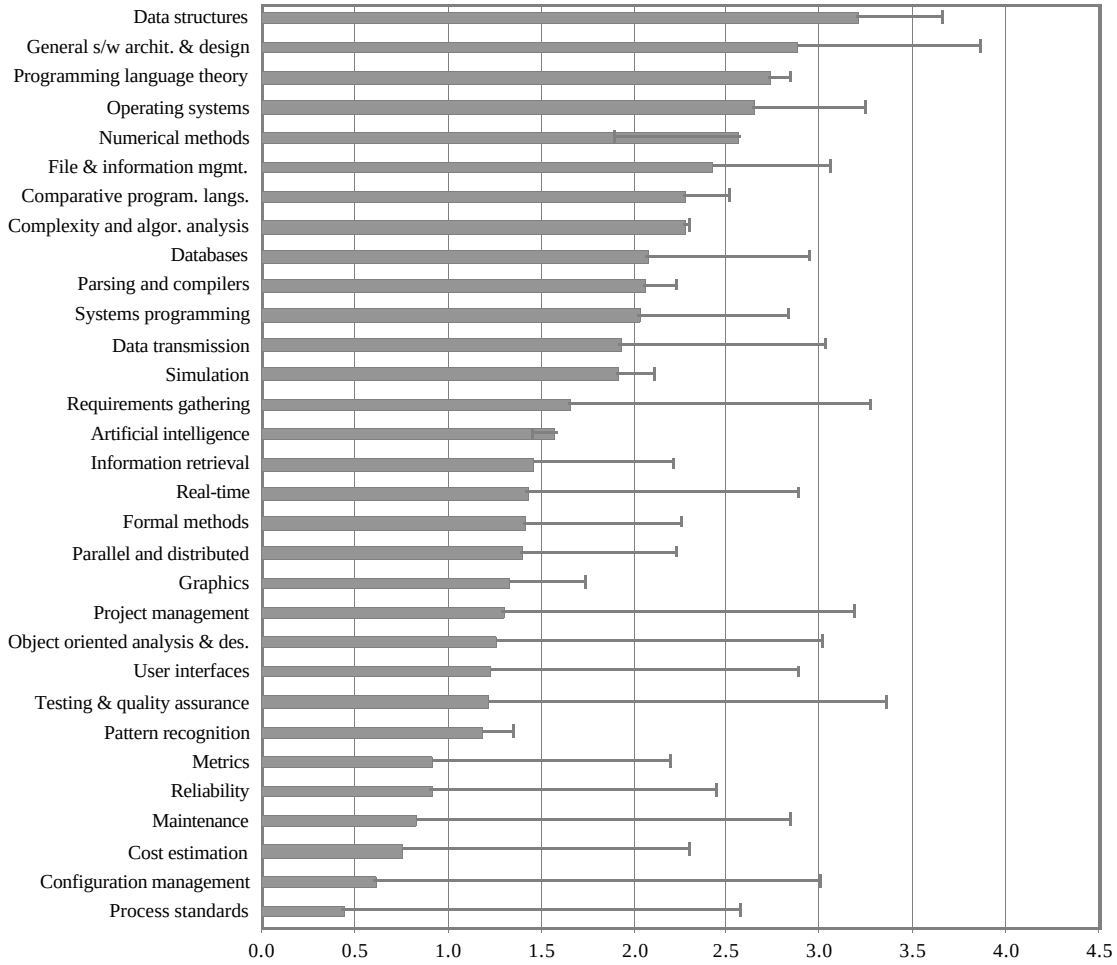

Figure 1: The amount participants say they learned in their formal education (gray bars) about software topics, compared with the amount they say they know now about the topics (ends of the black lines).

2. Analysis of software topics for the whole sample

Figure 1 illustrates how the participants in the survey perceive their knowledge of 31 distinct software topics. The light gray bars are the mean responses of participants to the following question:

Question i. How much did you learn about this at University or College?
0=Learned nothing at all.
1=Became vaguely familiar
2=Leaned the basics
3=Became functional (moderate working knowledge)
4=Learned a lot
5=Learned in depth ; became expert(Learned almost everything).

Participants were asked a second question about each topic:

Question 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)

Mean answers to question ii are plotted in figure 1 using the endpoints of the black lines. Lines that extend to the *right* of the gray bars indicate that the participant has learned more about the topic since completing formal education; lines that extend *left* from the end of the gray bars indicate that the participant has, on average, forgotten material. The length of the black lines indicates the magnitude of on-the-job learning.

The following are interesting observations about the data in figure 1:

• Participants felt they became functional (2.5 to 3.5) in only five of the 31 topics; of these, only data structures had a mean response above 3.

• Participants felt that they did not even learn the basics (less than 1.5) in over half the topics. Some of this lack of knowledge can be attributed to not having taken elective courses (e.g. pattern recognition, graphics) or to material that is very recent and which therefore would not have been covered in the education of older participants (e.g. object oriented analysis). However, it seems surprising that the typical participant did not even learn the basics about such well-established subjects as user interfaces, project management and testing.

• For most topics, participants have learned a considerable amount while in the workforce. Table 1 gives the eight topics for which on-the-job learning was highest (the longest black bars of figure 1). It is interesting that most of these are process-oriented topics that software engineers generally need to learn to function in the workplace. This provides evidence for boosting the coverage of these topics in educational institutions so as to better prepare graduates – particularly since some aspects of these topics are not likely to be effectively acquired on the job (e.g. the more theoretical aspects of testing, object-oriented analysis and user interfaces).

Topic	Extent of on-the-job learning
Configuration management	2.4
Testing and quality assurance	2.1
Process standards	2.1
Maintenance and reengineering	2.0
Project management	1.9
Object oriented analysis and design	1.8
User interfaces / human-computer interaction	1.7
Requirements gathering	1.6

Table 1. Topics for which on-the-job learning was highest (black bars in figure 1). The number is the difference between mean responses for the amount currently known about the topic (question ii) and the amount known in university (question i).

• In only two topics (data structures and general software architecture and design) did the participants feel that, on average, they currently know a lot (more than 3.5).

• For a few topics, namely numerical methods and artificial intelligence, the participants report a net loss of knowledge. Presumably the material has not been of much direct use in the workplace.

Figure 2 describes how important each software topic has been to the participants. The light gray bars are responses to the following question:

Question 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

The dark gray bars in figure 2 are responses to the following:

Question 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

The dashed lines compare the participants' perceived importance of the topic (question iii), to the importance educational institutes apparently give the topic (judged by how much students learned about it in their answers to question i). The average length of the dashed lines has been normalized to zero; i.e. when they extend to the right of the light gray bars, the topic is perhaps more heavily taught than might be justified by its perceived importance. When the dashed lines extend left, they indicate that the material is perhaps not taught enough.

The following are some observations from this data:

• Importance (question iii, light gray bars) and desire to learn more (question iv, dark gray bars) are very closely correlated; the correlation coefficient is 0.93. This is a sign of consistency in the participants' opinions.

• Data transmission, real-time software and systems programming rank surprisingly high in importance. This is likely due to sampling bias (see section 1).

• It would be expected, if curricula were highly relevant, that key compulsory topics would be ranked as the most important. In figure 2, certain core topics such as data structures operating systems and file management do indeed appear near the top; however many other topics which are not typically compulsory are ranked just as high (e.g. testing and quality assurance, requirements gathering, and project management). The data provide evidence that these ought to be emphasized more heavily.

• Conversely and as expected, many topics that are normally elective (e.g. artificial intelligence, pattern recognition, graphics and simulation) are ranked at the bottom of the importance scale. It is notable that the first three of these topics are rated on average as almost never useful (less than 1.5).

• Participants have ranked surprisingly low in importance a couple of topics that are widely cited as important for software engineering: metrics and formal methods. Assuming that there is value in the material, causes of this low ranking might be 1) lack of education about the material, or 2) lack of opportunity to have actively applied the material in the workplace, and hence to have come to understand its value.

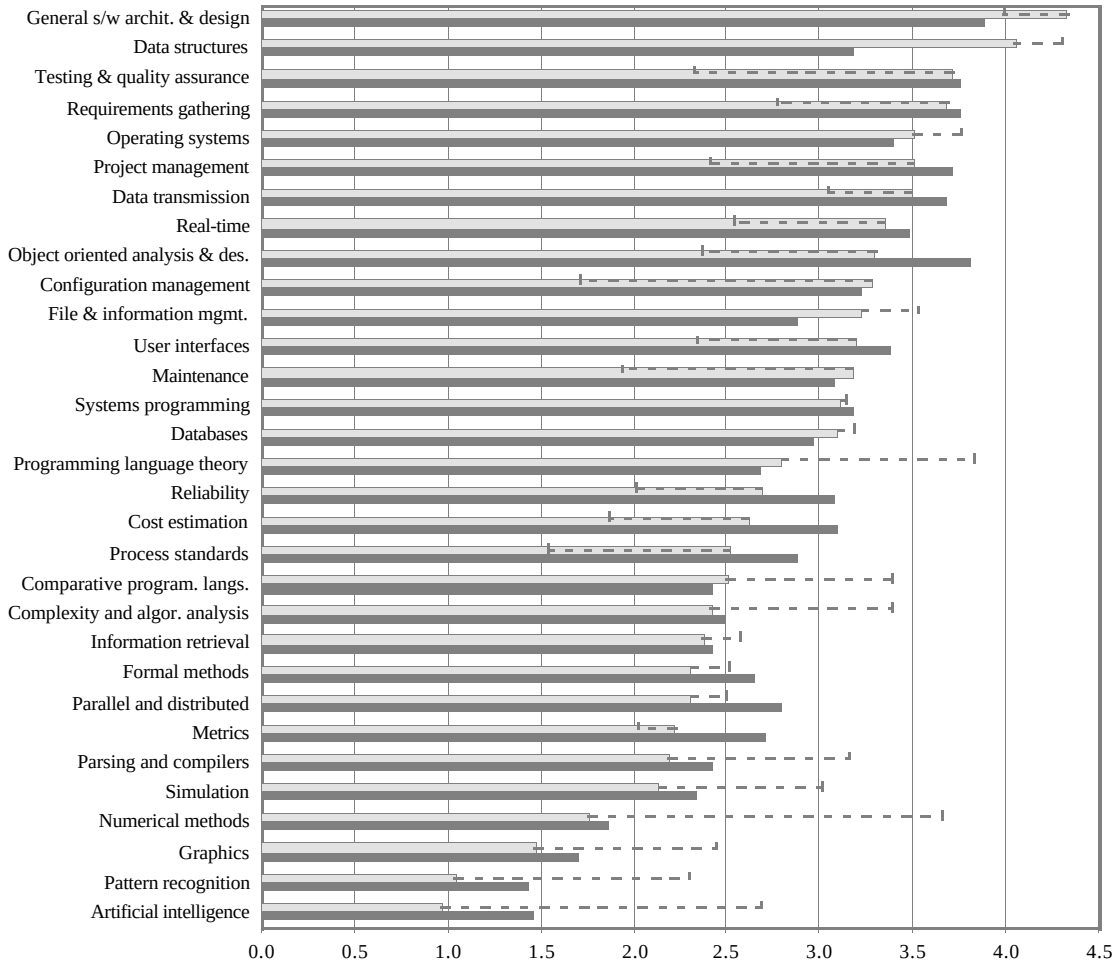

Figure 2: Perceived importance of the software topics. Light gray bars are mean answers to question iii (importance in career); dark gray bars are mean answers to question iv (desire to learn more). Dashed lines suggest whether the topic is overemphasized in university (extending right from importance bar) or under-emphasized (extending left).

• There is a poor correlation between importance (question iii), and emphasis in the curriculum (question i); the correlation coefficient is 0.25. This is a sign that, overall, curricula are far from optimal; the following points explore this issue further.

• Several topics appear to be overemphasized in curricula, as indicated by the dashed lines in figure 2, whose lengths are shown in table 2. Several topics appear in this list that are compulsory in most university programs (numerical methods, programming language theory and complexity and algorithm analysis) as well as other topics which are generally elective but which nevertheless are perceived to be overemphasized by participants.

• The topics for which curriculum emphasis is the lowest, as compared with perceived importance, are mostly software engineering topics (e.g. configuration management,

testing, project management etc.; see table 3). On the other hand, the topics which appear significantly over-emphasized (table 2) tend to be more theoretical or mathematical in nature. This suggests that separate software engineering programs might be needed, as distinct from the computer science programs which provided the education for most of the participants.

Topic	Apparent over-emphasis in curricula
Numerical methods	1.9
Artificial intelligence	1.7
Pattern recognition	1.3
Programming language theory	1.0
Parsing and compilers	1.0
Complexity and algorithm analysis	1.0
Graphics	1.0
Simulation	0.9
Comparative programming languages	0.9

Table 2: Topics for which the perceived importance was significantly less than the relative importance given the topic in educational institutions. The numbers are the length of the right-pointing dashed lines in figure 2.

Perceived importance (question iii, figure 2) is quite highly correlated with amount currently known (question ii, figure 1); the correlation coefficient is 0.93. However, there are some interesting differences between these variables. Table 4 lists top 10 topics for which there is the greatest difference .This provides a useful clue to companies regarding the topics in which they might consider training their employees.

Topic	Apparent under-emphasis in curricula
Configuration management	1.6
Testing and quality assurance	1.4
Maintenance	1.2
Project management	1.1
Process standards	1.0
Object oriented analysis and design	0.9
Requirements gathering	0.9
User interfaces / human-computer interaction	0.8
Real-time system design	0.8
Cost estimation	0.8
Reliability	0.7

Table 3: Topics for which the perceived importance was significantly greater than the relative importance given to the topic in educational institutions. The numbers are the length of the left-pointing dashed lines in figure 2.

It is interesting to compare the topics in table 4 to the top 10 topics in terms of importance (figure 2). Topics for which the ranking in table 1 is lower, are either well taught in university (data structures) or else perhaps relatively easy to pick up ‘on-the-job’ (testing

and quality assurance, and project management). Topics for which the ranking in table 4 is higher than the importance ranking perhaps have a greater need for classroom-based training (i.e. real-time systems, data transmission, maintenance and re-engineering, cost estimation and user interfaces)

Topic	Importance minus current knowledge
General software architecture and design	0.47
Real-time software development	0.47
Data transmission	0.46
Requirements gathering	0.41
Data structures	0.38
Testing & quality assurance	0.35
Maintenance and reengineering	0.34
Project management	0.33
Cost estimation	0.32
User interfaces / human computer interaction	0.30

Table 4: The top ten software topics in which companies might consider providing training courses for employees. The number is the difference between mean perceived importance of the topic and the mean current knowledge of the topic.

3. Analysis considering non-software topics

In addition to the software topics discussed in section 2, we asked the participants to answer the same four questions with respect to topics in mathematics, business, engineering, science and humanities. This section expands the analysis to include these areas.

Figure 3 shows the participants' perceptions of how much they learned about mathematics in their education, and how much they feel they know now. It is notable that participants have apparently forgotten much of their education in all nine topics – this is quite unlike most software topics where participants have learned considerably more since leaving university or college.

Calculus and linear algebra, rated 3.2, were the topics that the participants claimed to have learned most about from the entire list of topics in the survey, even more than the highest ranked software topic (data structures at 3.2). Along with differential equations, these topics had the largest loss of knowledge (between -1.2 and -1.4) in the survey following graduation. Although these topics provide prerequisite material for certain other topics, the data in figure 3 should result in some careful thought about how these topics are presented. Maybe the mathematics should be taught on an as-needed basis for the dependent topics, with additional math courses as electives for those who are interested: It seems a waste to invest 4-6 courses in subjects that will be so readily forgotten. Some might argue that the mathematics subjects teach problem-solving skills. This seems likely to be true, but perhaps the data indicate that mathematics education could be made more relevant (e.g. teaching with the use of software examples) so that the material is not so readily forgotten.

Figure 4 shows the amount learned and now known about a variety of other topics taught as either elective or complementary material in computer science programs. Some observations:

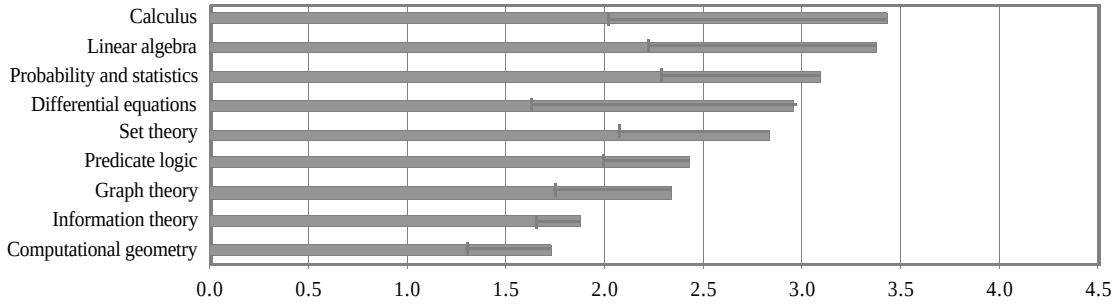

Figure 3: The amount participants say they learned in their formal education (gray bars) about mathematics topics compared with the amount they say they know now about the topics (ends of the black lines). This uses the same scale as figure 1; the software average is shown for comparison.

• Three of the topics (technical writing, ethics and professionalism, and management) were not apparently emphasized enough in formal education, judging by the large increase in knowledge since graduation.

• Basic science (physics and chemistry) and electronics were considered similar to mathematics in that they were heavily emphasized – more than the average software topic – but much of the material was later forgotten.

• On the other hand participants reported a net increase in knowledge since graduation of all the business and humanities subjects listed. This is especially interesting since there is a bias in our sample (see section 1) towards the more scientific aspects of computing (real-time) and away from management-information-system software. Clearly, educational institutions and accreditation bodies ought to consider requiring more business and humanities and less science.

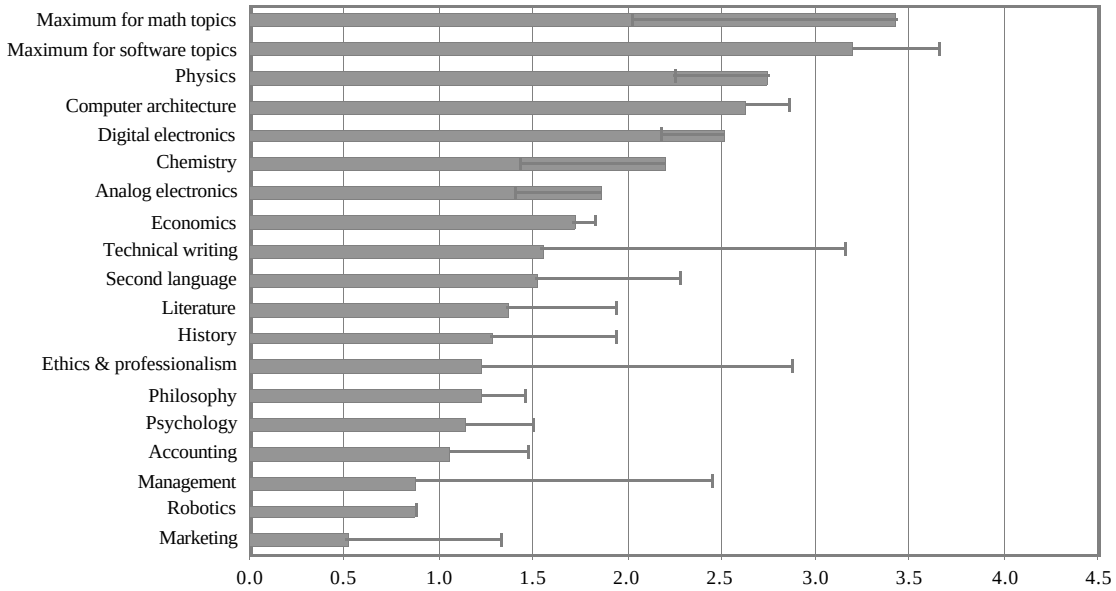

Figure 4: The amount participants say they learned in their formal education (gray bars) about miscellaneous topics compared with the amount they say they know now about the topics (ends of the black lines). This uses the same scale as figures 1 and 3; key statistics about software and mathematics are shown for comparison.

Figure 5 shows the importance with which the participants hold the various mathematics topics. Probability and statistics (needed for software reliability and interpretation of experimental data), and discrete mathematics (exemplified by predicate logic and set theory which are needed for formal methods) are clearly considered the most important. Calculus and differential equations, which are heavily taught according to figure 3, appear near the bottom of the list in terms of importance. This provides evidence that the balance in math curricula ought to be shifted towards the former topics.

It is interesting to note that there are three math topics where the answers to question iv (desire to learn more) were higher than the answers to question iii (importance): Information theory, graph theory and computational geometry are topics that are not universally taught, and thus participants might have ranked these relatively higher due to curiosity.

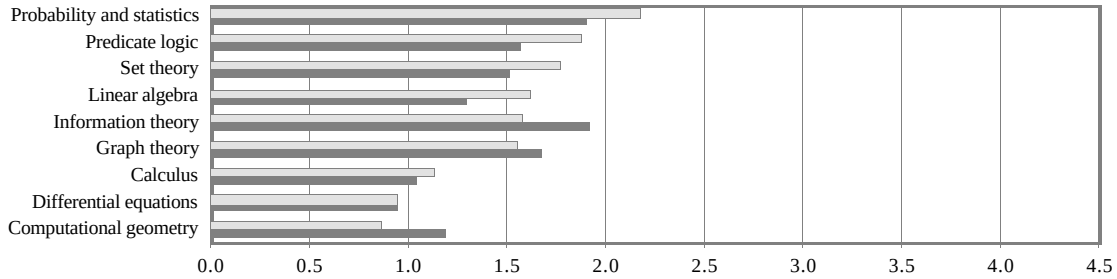

Figure 5. Perceived importance of the mathematics topics. Light gray bars are mean answers to question iii (importance in career); dark gray bars are mean answers to question iv (desire to learn more). This uses the same scale as figure 2.

Figure 6 shows the perceived importance of the miscellaneous topics. Technical writing is ranked particularly high: Of all the topics in the survey, only general software architecture and design came higher (3.9) than technical writing (3.6). Other topics given relatively high importance – above the average level for software topics – were ethics and professionalism, computer architecture, and management. The latter topic is particularly high in terms of desire of the participants to learn more.

Several other business topics (marketing, economics and accounting) are comparable with typical mathematics topics in terms of importance, but are given far less coverage (often none) in curricula.

4. Differences for real-time developers and managers

We performed several analyses of the data to detect differences in the responses of participants according to their backgrounds. The following subsections discuss two important such analyses. These differential analyses serve two purposes: 1) To help validate the survey (ensuring the general conclusions remain the same when one corrects for the real-time bias). 2) To provide additional data for those who want to educate or train people who are more (or less) oriented than the general software community towards real-time-systems or management

4.1 Real time developers compared to non-real-time developers

Since the survey had a bias towards software developers involved with real-time software, we wanted to see what differences exist among the following groups: 82 participants who

develop real time software exclusively; 33 participants who develop both real-time and non-real-time software, and 34 participants who do not develop real time software at all.

Most important demographic factors were similar for these three groups; for example, the proportions of participants with graduate degrees, bachelors degrees and college diplomas were practically identical. The percentages with computer science degrees were also very similar, although there were more engineers among the real-time group, and more science graduates among the non-real-time group.

The greater preponderance of engineering backgrounds among the real-time developers shows up when one notices that real-time developers learned significantly more about math, hardware and even software at university. Table 5 shows the average amount learned about various topics in the participants' formal education (question i).

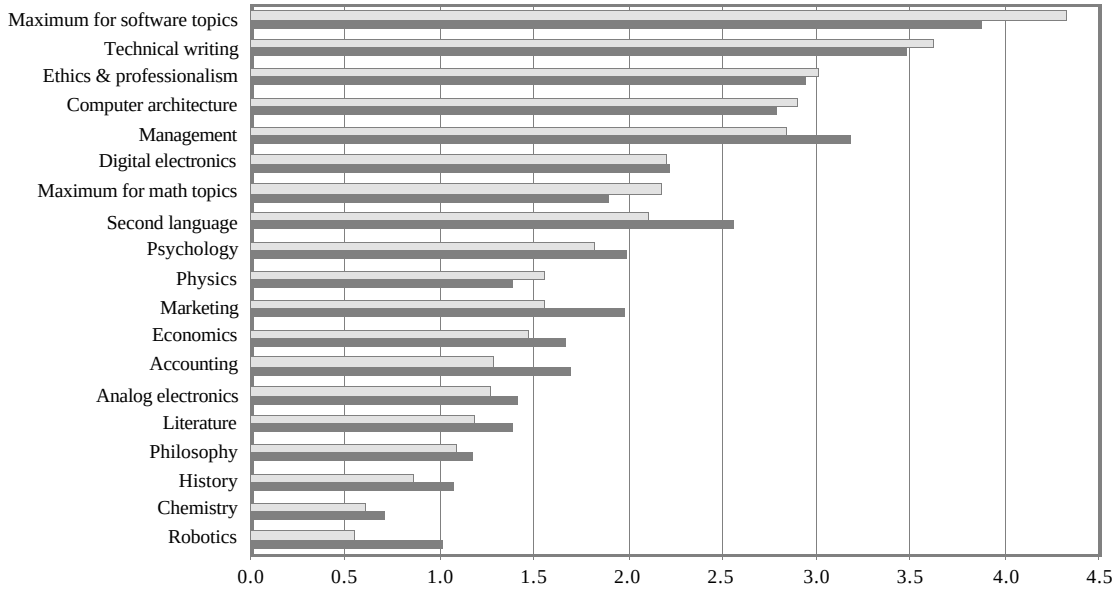

Figure 6: Perceived importance of the miscellaneous topics. Light gray bars are mean answers to question iii (importance in career); dark gray bars are mean answers to question iv (desire to learn more). This uses the same scale as figures 2 and 5.

Topic	Exclusively non-real-time developers	Developers who work on some real-time and some non-real-time systems	Exclusively real-time developers
Overall math	2.3	**2.8**	**2.8**
Overall software	1.3	**2.1**	1.8
Overall hardware	1.3	**2.0**	**2.2**

Table 5: Comparison of amount learned in university and college among developers who work with or do not work with real time systems.

The relative importance of topics (question iii) also differs among these groups of participants. Table 6 summarizes the key differences. It is interesting that the group which performs both types of work tend to give higher importance ratings to many topics. It might be because this 'both' group contains many more managers – as we will see in the next section, managers consistently rate almost all topics as more important than do non-managers.

Topic	Exclusively non-real-time developers	Developers who work on some real-time and some non-real-time systems	Exclusively real-time developers
<u>More important to real-time developers</u>			
Real-time	1.4	3.8	4.1
<u>More important to those who do both real-time and non-real-time work</u>			
Data transmission	2.6	3.9	3.8
Systems prog.	2.3	3.8	3.3
Operating systems	3	3.7	3.6
Software reliability	1.6	3.1	2.9
Algorithm analysis	1.9	3.0	2.4
Process standards	1.5	2.9	2.8
Parsing and compilers	1.6	2.7	2.1
Parallel & distributed	1.9	2.7	2.4
Software metrics	1.6	2.6	2.3
Predicate Logic	2.1	2.5	1.6
Simulation	1.5	2.5	2.3
Math as a whole	1.5	1.8	1.4
<u>More important to non-real-time developers</u>			
Databases	3.6	3.5	2.6
User Interfaces	3.6	3.5	2.9
Information retrieval	2.6	2.6	2.1

Table 6: Comparison of topic importance among developers who work with or do not work with real time systems.

4.2 Managers compared to non-managers

When comparing the 57 managers to the 110 non-managers, demographic factors such as type and level of education, as well as location of work, were very similar. As expected, managers tended to be more expert (63% vs. 22%) and to have a wider variety of experience with different types of software (more of them fell in the 'both' category of section 4.1).

Managers felt they had been slightly less well educated than non-managers, particularly in real-time systems, project management, requirements gathering and object oriented analysis. This might be attributed to them having been educated earlier, before some of

these topics were as widely developed or taught. Managers felt they had been better educated in general management, however.

There were significant differences in the perceived importance of topics among these two groups. Every single topic was considered more important to managers than to non-managers, with the notable and strong exception of second language (1.7 for managers, vs. 2.3 to non-managers). It is hard to explain the above, except that perhaps managers have to deal with a wider spectrum of topics in their work – and tend not to be foreigners.

There were several topics which managers found very much more important than non-managers; these are listed in table 7.

Topic	Managers	Non-managers
Project management	4.5	3.0
Management	4.2	2.1
Process standards	3.2	2.1
Marketing	2.5	1.1
Accounting	2.1	0.9

Table 7: Topics that managers found very much more important than non managers.

5. Perceived needs of employers vs. perceived needs of employees

Those who hire employees for their software organizations create lists of knowledge and skills they would like prospective employees to possess. A group of Ottawa-area companies produced such a list [5] in conjunction with OCRI (Ottawa Centre for Research and Innovation). It is interesting to compare the perceived importance of topics in the OCRI document to the opinions of the software professionals in our study, many of whom worked for the same companies as the authors of the OCRI document.

Table 8 lists the topics in the OCRI document in order of importance[2]. The third and fourth columns give a comparison to the ranking of similar topics in our survey. Table 9 restates the results of our survey, taking the more important software and miscellaneous topics so as to be in correspondence with the coverage in the OCRI document. In both tables, important differences are highlighted using bold type; asterisks mark correspondences between topics that are very approximate.

The following observations can be made about tables 8 and 9:

• The employers ranked several topics substantially lower than their rankings in our survey of practitioners. The most important such case is architecture and design, which appears at the top of the practitioners' list. Requirements definition and gathering as well as data transmission and networking were also substantially lower in the employers' list.

• More interesting, perhaps, is that several topics appearing in our survey have no match in the employers' list. Some of these topics (e.g. data structures, file management and computer architecture) may not have been listed because the employers consider them very

[2] The OCRI document did not actually present this ordering, however it has been computed from the data given.

basic, and thus not necessary to include since everybody will have this background. The same cannot be said, however, for such topics as user interfaces, and maintenance and reengineering: These appear reasonably high on our list but are completely absent from the employers' list.

OCRI Rank	Topic in employers' (OCRI) document	Whether employers ranked the topic higher or lower than practitioners in our relevance survey	Primary (and secondary) topics in the relevance survey to which the employers' topic was compared. Numbers for survey topics are given in table 9.
1	Object oriented languages	**Higher**	10 (20, 24)
2	Real time / Operating systems	Slightly higher	6 (9)
3	Testing	Same	3
4	MS-Windows environment	**Not in relevance survey**	
5	Software process	Similar	7 (11, 22, 23)
6	CASE	**Not in survey**	
7	Reviewing	Similar	3
8	Analysis tools	**Not in survey**	
9	Unix environment	**Not in survey**	
10	Project experience	**Not in survey**	
11	Technical writing	Slightly lower	5
12	Requirements definition	**Lower**	4
13	Databases	Similar	16
14	Class libraries	**Not in survey**	
15	Distributed systems	Slightly higher	28
16	System architecture & design	**Much lower**	1
17	Networking	**Lower**	8
18	Call control (telephony)	**Not in survey**	
19	Keyboarding	**Not in survey**	
20	Drivers	Similar	15
21	Embedded systems	Slightly lower	9 (15)
22	Compilers and parsers	Similar	31

Table 8: Topics in which companies want employees to have skills, in order of importance. Data is computed from a document prepared by several companies in conjunction with OCRI [5]. Interesting differences with respect to the relevance survey are highlighted in bold.

• Conversely, the employers include specific tools and languages that were not part of our survey – this is consistent with typical job advertisements which give such skills prominence. We chose not to ask specific languages and tools because there are just too many of them, they change frequently and can be learned quickly (we believe) if people have been educated in the core topics in our list. Of particular note is the presence of object oriented languages at the very top of the employers' list (which we have roughly tied to object oriented analysis in the tables). We believe that, to some extent, this indicates that employers have too short a time horizon when seeking new employees: They tend to think

of specific work they want done which requires, for example, C++. This in turn causes students (who read job ads) to put languages and tools at the top of their must-learn lists and resumés. It might be more prudent for employers and students to look at the results of our survey to see what topics prove more useful and enduring in the long run.

Our Rank	Topic in relevance survey	Whether participants in the relevance survey ranked the topic higher or lower than employers.	Primary (and secondary) topics in the employers document to which the relevance survey topic was compared. Numbers for employers' topics are given in table 8.
1	General s/w architecture & design	**Much higher**	16
2	Data structures	**Not listed by employers.**	
3	Testing & quality assurance	Similar	3 (7)
4	Requirements gathering	**Higher**	12
5	Technical writing	Slightly higher	11
6	Operating systems	Slightly lower	2 (tie)
7	Project management	Similar	5
8	Data transmission	**Higher**	17
9	Real-time	Slightly lower	2 (tie)
10	Object oriented analysis & des.	**Lower**	1 (16, 14)
11	Configuration management	Slightly lower	5
12	File & information mgmt.	**Not listed.**	
13	User interfaces	**Not listed.**	
14	Maintenance & reengineering	**Not listed.**	
15	Systems programming	Similar	20
16	Databases	Similar	13
17	Ethics & professionalism	**Not listed.**	
18	Computer architecture	**Not listed.**	
19	Management	**Not listed.**	
20	Programming language theory	Not directly listed.	
21	Reliability	**Not listed.**	
22	Cost estimation	**Lower**	5
23	Process standards	**Lower**	5
24	Comparative program. langs.	Not directly listed.	
25	Complexity and algor. analysis	**Not listed.**	
26	Information retrieval	**Not listed.**	
27	Formal methods	**Not listed.**	
28	Parallel and distributed	Slightly lower	15
29	Metrics	**Lower**	5
30	Digital electronics	**Not listed.**	not in survey
31	Parsing and compilers	Similar	22

Table 9: Restatement of selected data from the relevance survey (figures 2 and 6), in order of importance. Interesting difference with respect to a list of topics compiled by employers are highlighted in bold.

6. Conclusions and recommendations

We have presented an analysis of data from a survey of 168 software practitioners, each of whom was asked about topics typically found in a computer science or software engineering curriculum. The survey asked the practitioners about their knowledge of the topics, and how important they perceived each to be.

Most of the participants were from Canada, but when data from the United States was isolated and analyzed separately, few important differences were found. Similarly, there was a bias towards real-time developers; however, when the data from non-real-time developers were separately analyzed, most conclusions were very similar.

It should be noted that while the statistics in this paper may apparently present strong evidence that the emphasis on certain topics in curricula ought to be either increased or decreased, it is important to use the data presented here as only one of many sources of information when making curriculum decisions. There may be very valid reasons for including unpopular topics in curricula – e.g. a belief that the material will help students in their general problem-solving abilities, or that the material will become more important in the future. There may similarly be reasons why certain educational institutions should under-emphasize some topics that seem to be in high demand; for example certain topics may be considered 'passing fads' or might better be left to on-the-job training. Furthermore, the statistics presented here are purely opinions: Although the sample is large enough to prevent any individual's strong opinion about a certain topic from skewing the results, there may be systematic tendencies for people to dislike or undervalue certain topics, and to overemphasize others. For example, the data show that mathematics topics are considered low in importance, but high in terms of amount learned. It might be the case that the nature of the mathematics education made it stand out in the minds of participants so that they thought it was emphasized more than it really was in their studies. Similarly, the participants may downplay the usefulness of mathematics if it provides them merely with background skills (that they don't notice themselves using) for their main job which is software development.

The most important conclusions we were able to draw from the data are that certain topics are significantly more highly emphasized in universities and colleges than would be warranted by their perceived importance, while for other topics the inverse holds true. We summarize our findings in the following two subsections, which give recommendations to educational institutions, companies and individuals.

6.1 Recommendations to educational institutions:

Table 10 lists topics for which evidence in this paper suggests there should be increased emphasis in university or college curricula. Table 11 lists topics for which there is evidence for a decrease. The rows in each table present the type of evidence that argues in favor of the topic having its emphasis changed – for more details see the earlier sections.

The proposed curriculum change can be accomplished in two ways:

1. Change the balance of existing programs in favor or against the topics listed.

2. Create a new program that has a balance as shown in the tables, while keeping the old program. In particular, universities might consider creating software engineering programs to complement computer science programs, since most of the topics in table 10 are of a software engineering nature.

Some specific discussion is warranted regarding mathematics: It would clearly not be possible to reduce the math content very far and still be able to call a program 'science' or

'engineering'. So how does one confront the evidence that much math is given far more weight in curricula relative to its importance, and is quickly forgotten? Some suggestions are 1) To tie mathematics education to software problems so that it becomes more relevant, and 2) to shift the emphasis from continuous mathematics towards discrete mathematics, probability and statistics (with relevant software examples).

All of the curriculum recommendations presented here should be taken as one piece of evidence among many. Clearly there will be pedagogical reasons for emphasizing material differently from what is presented here.

Reason for recommended increase in emphasis	Testing	Object orientation	User interfaces / HCI	Technical writing	Ethics & professionalism	Management	Project management	Requirements gathering	Real time systems	Data transmision	Reengineering	Cost Estimation	Psychology	Marketing	Economics	Accounting
Significant learning is required in the work force following graduation	x	x	x	x	x	x										
Most practitionners do not even know the basics	x		x				x									
Ranked particularly high in importance	x			x			x	x								
Knowledge of practitioners is low relative to importance, and the topic may be hard to pick up on the job			x						x	x	x	x				
Business and humanities topics that are comparable with mathematics and other sciences (e.g. chemistry) in terms of importance, but which are far less emphasized					x	x							x	x	x	x

Table 10: Summary of topics for which it is recommended that universities and colleges increase their education of computing students

6.2 Recommendations to companies, students and employees

The key recommendation to the consumers of education is to increase your training in areas that this survey shows:

a) are most important, particularly where knowledge of typical practitioners is relatively low;

b) that participants seek to learn more about.

Highly recommended courses would therefore be those in software architecture and design, real time systems, data transmission, requirements gathering, testing and quality assurance, maintenance and reengineering, project management, object oriented analysis and design, human-computer interaction, and technical writing.

Additionally, managers should receive extra training in general management skills, process standards, and marketing. Companies focusing on real-time systems should emphasize real-time design and data transmission; while companies focusing on MIS software should emphasize databases, human-computer interaction and information retrieval.

Reason for recommended decrease in emphasis	Numerical methods	Programming language theory	Algorithm analysis	Calculus	Linear algebra	Differential equations
Net loss of knowledge following graduation	x			x	x	x
Low importance with respect to emphasis	x	x	x	x	x	x

Table 11: Summary of topics for which it is recommended that universities and colleges reduce the requirements for at least some computing students

Employers who are in the process of hiring new staff should think carefully about the topics their current employees find important. Conducting a survey like this in-house could provide valuable information to help guide the advertising process and make hiring decisions. On the other hand, if the company is large and diverse enough (perhaps with a bias towards real-time software), then the results of this survey should be very applicable. In particular, employers should think more about longer-term skills rather than particular tools or languages, unless they are hiring staff merely for short term contracts.

Acknowledgments

We would like to thank all the participants in this survey, and the managers of the companies who encouraged their employees to participate. We also thank K. Teresa Khidir for her painstaking work at data entry, and Anatol Kark for his efforts at setting up and managing the web site for gathering the data.

References

[1] Lethbridge, T.C. “A Survey of the Relevance of Computer Science and Software Engineering Education”, submitted to *11th Conference of Software Engineering Education and Training*, Atlanta, 1998

[2] G. Ford, (1996) “The SEI Undergraduate Curriculum in Software Engineering”, Software Engineering Institute

[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] Joint IEEE Computer Society and ACM Steering Committee for the Establishment of Software Engineering as a Profession; at http://computer.org/tab/seprof/

[5] Mason, J., Pinard, D., and Thompson, N. “Software Skills List”, Technical Report of the Technology Resource Initiative, Ottawa Center for Research and Innovation (OCRI), June 1997.