By Sami Surakka

What Subjects and Skills are Important for Software Developers?

A small survey of Finnish IT professionals, academics, and students has important implications for computer science degree programs.

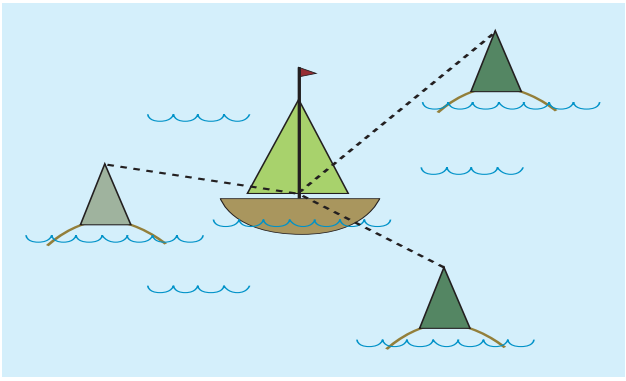
According to Cohen, Manion, and Morrison,[1] triangulation is defined in its original and literal sense as:

> *... a technique of physical measurement: maritime navigators, military strategists and surveyors, for example use (or used to use) several location markers in their endeavors to pinpoint a single spot or objective. By analogy, triangular techniques in the social sciences attempt to map out, or explain more fully, the richness and complexity of human behavior by studying it from more than one standpoint...*

The basic idea of triangulation in navigation is presented in the figure. A navigator uses a compass to measure

[1] *Research Methods in Education, 5th Edition*. L. Cohen, L. Manion, and K. Morrison. RoutledgeFarmer, London, 2000.

directions called bearings from different known locations called beacons. The bearings are drawn as dashed lines. Beacons can be, for example, lighthouses, buoys, or smokestacks. Two bearings are enough to calculate the location but a third bearing can be used to make sure.



Triangulation in navigation.

When triangulation is used in educational research, one tends to accept or assume that a single result is not so exact. However, when the results from different sources are combined, the combined results should be more exact. In this article, the results of three different groups—software developers, professors and lecturers, and master's students—will be presented.

Considering the wide range of IT professionals, including consultants, database administrators, project managers, and systems administrators, this article has been targeted toward software developers, a term that includes both programmers and software engineers. According to Gallivan, Truex, and Kvasny [2], "programmer" and "software engineer" were the most common IT job titles in 2001 (proportions 21% and 16%, respectively). In particular, software developer positions are important to education because they are common entry-level positions. That is, graduates do not typically start their careers as project managers or consultants.

Eleven experienced Finnish software developers and 19 Finnish professors or lecturers evaluated the importance of various subjects and skills related to software development. The research for the software developers was conducted between November 2003 and January 2004, and for the professors and lecturers in January and February 2005. Both research efforts were Delphi studies. The Delphi technique was originally used to forecast the future; the name originates from "the oracles of Delphi," referring to an ancient Greek island. In a Delphi study, the respondent group is typically small but is comprised of leading experts in the field of study. It is assumed that the quality of the respondents is more important than the number of respondents; that is, evaluation from a small group of leading experts should be reliable. In the conventional Delphi technique, several questionnaire rounds are conducted. In this article, only some results from the first questionnaire round are reported. The second questionnaire round was limited to a different topic and was excluded from this article for purposes of brevity.

The respondents evaluated the importance of 42 different subjects and skills such as discrete mathematics, object-oriented programming, and project management. The questionnaire was targeted more at the area of software systems than in the previous surveys because one purpose of this research was to collect data that could be used in planning for the specialization in software systems at the Helsinki University of Technology. The results were compared against the results of Lethbridge's survey [4–6], this being the most relevant previous research. He questioned 186 respondents about 75 educational topics. According to his results, the five most important topics were data structures, specific programming languages, software design and patterns, requirement gathering and analysis, and software architecture. Topics that were taught relatively more than their importance might warrant were physics, chemistry, and different areas of mathematics. However, Leth-

When triangulation is used in educational research, ONE TENDS TO ACCEPT OR ASSUME THAT A SINGLE RESULT IS NOT SO EXACT. However, when the results from different sources are combined, the combined results should be more exact.

bridge's results are already nine years old.

In addition, the small (n = 24) sample of master's students answered a similar survey in 2004. The students were in the process of graduating from the specialization in software systems at the Helsinki University of Technology. Obviously, the opinions of the students are not as reliable or convincing as the opinions of the software developers, and the professors and lecturers because the students have less work experience. However, these results are also included because this part of the research is easy to repeat by other institutions that use graduate exit surveys.

The main contribution of this work is the updating of Lethbridge's results. Most results of this study are as expected. For example, it is obvious that Web-related subjects and skills are now evaluated as being more important than in 1998. Such changes are interesting but it is also useful to know whether certain subjects or skills have become more or less important, and to notice that basic subjects and skills are still evaluated as being important even though various new technologies have been launched during the last 10 years. The results here are useful for training departments of companies, training institutes, and curriculum designers in universities—in particular for those professors responsible for the specialization in software systems in computer science programs. Students can use the results when they are selecting elective CS courses, especially in industry-oriented master's programs. Finally, software developers might want to compare their skills to the results.

Subject or skill	Software developers	Lethbridge's respondents	Professors and lecturers	Master's students
Mathematics, physics and theoretical computer science:				
Other areas of theoretical computer science (automata)	3.3	2.3	2.9	2.1
Logic (in particular, propositional and predicate logic)	2.8	2.3	2.9	1.7
Discrete mathematics	2.6	1.9	3.1	1.7
Mathematics for continuous systems	2.0	1.7	1.7	1.3
Physics	1.6	2.0	1.5	1.1
More technical or part of the operational system:				
Data structures and algorithms	3.8	3.1	3.9	3.6
Procedural programming	3.8	-	3.7	3.2
Object-oriented programming	3.6	3.0	3.9	3.8
Software architectures	3.5	3.1	3.6	3.5
Internet protocols	3.4	2.9	2.3	3.2
Script programming	3.4	-	2.8	3.1
Operating systems	3.3	3.0	3.7	3.0
Systems programming	3.2	2.8	2.9	2.6
Computer/data security	3.2	2.3	3.0	3.0
Distributed systems	3.1	2.4	3.5	2.6
Compilers	3.1	2.4	3.1	2.3
Concurrent programming	3.1	-	3.5	3.1
Computer architecture	3.0	2.6	3.2	2.5
Database management systems	2.7	3.0	3.3	3.1
Implementing techniques of user interfaces	2.7	3.0	3.1	3.0
Implementing techniques of WWW systems	2.7	-	2.8	2.8
Extensible Markup Language (XML) techniques	2.7	-	2.5	3.2
Functional programming	2.6	-	2.9	2.3
Real-time systems	2.6	2.6	2.7	2.0
Embedded systems	2.5	-	2.7	2.0
Logic programming	2.3	-	2.6	1.7
Other telecommunications techniques than Internet protocols	2.0	2.5	2.0	2.5
Computer graphics	1.9	2.2	2.3	1.5
Artificial intelligence and knowledge engineering	1.6	1.8	2.5	1.7
Software engineering (different phases of life cycle):				
Design	3.7	3.1	3.8	3.9
Implementation	3.7	-	3.9	3.7
Requirements	3.6	3.1	3.4	3.3
Test	3.5	3.0	3.8	3.3
Concept exploration	3.0	-	3.4	3.0
Approval	2.6	-	2.9	2.5
Operation and maintenance	2.5	2.7	2.9	2.4
Installation and checkout	2.3	-	2.8	2.3
Packaging and delivery	1.9	-	2.3	1.8
Retirement	1.8	-	2.3	1.7
Software engineering (possible in several phases):				
Version and configuration management	3.6	3.0	3.6	3.5
Project management	3.2	3.0	3.6	3.1
Documenting	3.0	3.1	3.4	3.2

The importance of various subjects and skills. Means from four research projects. Scale: 1 = Not at all important to 4 = Very important.

The 42 items used in the questionnaires (see the table) were selected using group work and previous literature. Three members of the group had a doctoral degree in CS and worked as professors or lecturers at the Helsinki University of Technology. The purpose was to select subjects and skills that were commonly required in computer science programs or might be important for software developers.

The software developers were selected using recommendations; the goal was to find 10 to 20 especially good software developers. In all, 59 people were recommended; 40 were not invited, for several different reasons (for example, the person had graduated less than five years ago). Thus, 19 people were asked to participate. From these, 11 promised to participate. Their mean age was 37.1 years. The positions were distributed into the following groups: senior software engineers and developers, 45%; researchers, 27%; and managers or directors, 27%.

For the sake of brevity, the selection and the demographics of the professors and lecturers and the demographics of the students are not presented here, but can be found in [11].

The results are presented in the table and are divided into the four categories used in the three questionnaires. Within each category, the rows are first ordered according to the results from the software developers, the second according to Lethbridge's results [4], and the third according to the results from the professors and lecturers. Lethbridge's scale of 0–5

was converted to a scale of 1–4 to enable comparison of the results. Lethbridge's items were pooled in some cases where 2–4 of Lethbridge's items corresponded to one item in my questionnaires. A dash (-) indicates that Lethbridge's survey did not include a corresponding item.

While I calculated the differences inside each group and between the groups that were statistically significant ($p < 0.01$, see [11] for details of statistical analysis), here I focus solely on the differences between the groups. First I discuss the differences between the software developers and Lethbridge's results, since these findings are most relevant to *Communications* readers. This discussion is followed by a brief analysis of the differences between the software developers, the professors and lecturers, and the students.

Perhaps the most important recommendation in Lethbridge's article concerned science and continuous mathematics. The respondents of the current study also evaluated the importance of physics and continuous mathematics as being very or quite low. The differences between the two groups were not statistically significant. Thus, the present research confirms Lethbridge's results in this respect.

In general, Lethbridge's respondents' means were a little lower than the current study respondents' means across the range of items (Lethbridge's 2.6 and my 2.9). A probable explanation is that the current questionnaire asked "How important do you think the following subjects and skills are for demanding programming tasks?" whereas Lethbridge asked about the usefulness of specific material during the entirety of a respondent's career.

Questionnaire variations could also explain the statistically significant differences between the following items: "Data structures and algorithms," "Design," "Other areas of theoretical computer science (for example, automata)," "Requirements," and "Version and configuration management." In all these items my respondents evaluated them as being clearly more important than Lethbridge's respondents. It is likely that respondents to the current questionnaire evaluated theoretical computer science as being more important because they believed theoretical tasks are often also more demanding—or vice versa. The importance of the items "Design," "Requirements," and "Version and configuration management" is probably greater in larger or more demanding projects. I do not have a clear explanation for the difference in the item "Data structures and algorithms." However, my assumption is that in more demanding projects it is important on the one hand to be aware of several different data structures and algorithms, and on the other hand understand the trade-offs arising from efficiency.

Lethbridge's survey was conducted in 1998. Since then, the use of the Web has increased dramatically. The number of Web sites was approximately three million in 1998 and nine million in 2002 [9]. The statistically significant differences in the items "Computer/data security" and "Distributed systems" are probably related to the increased use of the Web. In these two items, Lethbridge's respondents' means were smaller than the current respondents' means. One could argue the greater importance of "Computer/data security" is partly a consequence of the terrorist attacks on Sept.11, 2001. This could be a possible explanation in the U.S., but in Finland the most likely explanation is the increased use of the Web. Another explanation for the difference in the item "distributed systems" is that in Finland, telecommunications companies are such large employers in the IT area that this likely had an effect on the answers. Forty-five percent of the respondents worked for telecommunications companies when in Lethbridge's survey the proportion was 14%.

In addition, the current respondents evaluated the item "object-oriented programming" as being more important than Lethbridge's respondents did. However, this difference was statistically almost significant ($p < 0.05$). Based on Lethbridge's results and the con-

Professors in charge of software systems specialization should make sure that TECHNICAL TOPICS SUCH AS COMPILERS AND DISTRIBUTED SYSTEMS are studied as well as software engineering topics.

tent analysis of job advertisements (for example, Litecky and Arnett's research [7]), object-oriented programming was already very or at least relatively important five years ago. The following are probably explanations for the increased importance of object-oriented programming: the complexity of modern software systems; the evolution of object-oriented languages and tools; and the growth of the Web and use of Java in Web applications.

I analyzed the students', the professors and lecturers,' and software developers' results against each other because these three groups answered similar questionnaires during approximately the same time period and all three groups worked in Finland. In the following three items, the students evaluated them as being less important and the differences were statistically significant between the students and the other two groups: "Discrete mathematics," "Logic (in particular, propositional and predicate logic)," and "Other areas of theoretical computer science (for example, automata)." All these items refer to more mathematical subjects. A probable explanation is the students who selected the specialization in software systems like mathematics less than the students who selected mathematically more demanding specializations such as "Theoretical Computer Science" or "Neural Networks and Signal Processing." The professors and lecturers evaluated the item "Internet protocols" as being less important than the two other groups and the differences were statistically significant. A probable explanation is that 45% of the software developers and 20% of the students worked for telecommunications companies. In addition, the professors and lecturers evaluated the item "Artificial Intelligence and Knowledge Engineering" as being more important than the two other groups and the differences were statistically significant. A possible explanation is that some professors and lecturers just considered the subject to be (academically) interesting.

Implications for CS Degree Programs

In this article, I consider the implications of the results only for university-level education, further limiting the focus to the typical requirements of accredited CS programs in the U.S., because there are no statistics from other countries similar to the McCauley and Manaris survey [8]. Their survey is already four or five years old but apparently, a more recent survey has not been conducted.[2] The article by Parnas [10] concerned the differences between CS and software engineering (SE) programs. He wrote "In the SE program, the priority will be usefulness and applicability; for the CS program it is important to give priority to intellectual interest, to future developments in the field, and to teaching the scientific methods that are used in studying computers and software development." Obviously, Lethbridge's results and the present results are more relevant to SE programs. However, I write about CS programs and software systems specializations because the number of SE programs is so small. As Parnas notes, "Computer Science departments have tried to fill the gap by including so-called 'Systems' or 'Applied Computer Science' courses in their offerings." From the different topics, I limited (for brevity) inclusion to continuous mathematics, physics, theoretical computer science, programming paradigms, and the relationship between technical and software engineering subjects and skills.

Lethbridge [5] wrote: "Because of the low importance and high forgetability of continuous mathematics and basic science, universities 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." I agree with this recommendation. The role of mathematics in computer science education is a controversial subject that was covered in the September 2003 issue of *Communications*. One working group is "dedicated to promoting mathematics as an important tool for problem-solving and conceptual understanding in computing," [3] whether similar groups or articles about physics exist is uncertain. However, if physics is removed, an institution should take care that any requirements for scientific methods are not removed as an indirect consequence because according to Computing Curricula 2001 (CC2001) [1], scientific methods should be required. Still, CC2001 does not make physics or any other natural science compulsory because scientific methods can be taught, for example, using laboratory experiments about the performance of algorithms.

Based on these results, theoretical computer science should be compulsory. McCauley and Manaris [8] reported that in ABET/CAC accredited bachelor programs during the academic year 2001–2002, 49% required a course on the theory of computation. However, these results only concerned various upper-level courses. Some institutions require theoretical computer science on lower-level courses; that is, during the first or second year. Based on the report, the McCauley and Manaris survey did not ask about this area. According to CC2001, basic logic is included in the core topics but, for example, automata theory is not. Based on these results, some other areas of theoretical computer science might be more important

[2] Personal communications with R. McCauley, Sept. 9, 2005.

than logic. However, the questionnaires described here were not detailed enough to conclude what kind of theoretical computer science should be taught to CS students. Valmari [12] has written about CC2001: "In my opinion, the suggested content of discrete structures is small and partly poorly chosen. Instead of combinatorics and graphs there could be, for example, the theory of structure of clauses, BNF, constructing and analysis of definitions, or basic mathematics for reactivity and concurrency."

The five programming paradigms are imperative programming, functional programming, object-oriented programming, logic programming and constraint logic programming, and concurrent/distributed computing. Based on the results of this study, the order of importance for these five paradigms is the following:

1. Imperative programming and object-oriented programming (these two paradigms tied for first place);
3. Concurrent/distributed computing;
4. Functional programming; and
5. Logic programming and constraint logic programming.

McCauley and Manaris [8] reported that in ABET/CAC accredited bachelor programs during the academic year 2001–2002, 31% taught procedure-oriented and 36% taught object-oriented paradigm as the primary paradigm and the most common primary programming languages used were C++ (53%), Java (51%), and C (22%). The match between the curricula and the results of this study is good because the two most important paradigms were well covered. In addition, they reported how often various upper-level courses were required. Here, only the results concerning concurrent/distributed computing are presented because based on the results of this study, it is the third important paradigm. Their report had no results related to distributed systems and concurrent programming courses. However, 96% of the departments required an operating systems course. It is common that concurrency is part of an operating systems course. Thus, it is possible but not certain that the situation is good for the third important paradigm, too. I do not emphasize requiring concurrent programming and distributed systems in all CS bachelor programs, since not every undergraduate student is aiming to get software development positions. Based on the results of this study, concurrent programming and distributed systems are suitable compulsory topics for a specialization in software systems in a master's program.

Finally, professors in charge of software systems specialization should make sure that technical topics such as compilers and distributed systems are studied as well as software engineering topics. Obviously, technical topics should be emphasized more when specializing in software systems. However, omitting software engineering topics altogether might be a mistake because many software engineering topics were evaluated as being important.

References

1. Engel, G., and Roberts, E., Eds. Computing curricula 2001. Computer science. IEEE and ACM; www.acm.org/education/curric_vols/cc2001.pdf (accessed Sept. 18, 2006).
2. Gallivan, M., Truex III, D.P., and Kvasny. L. An analysis of the changing demand patterns for information technology professionals. In *Proceedings of SIGCRP'02* (Kristiansand, Norway, May 14–16, 2002).
3. Henderson, P.B., Baldwin, D., Dasigi, V., Dupras, M., Fritz, S.J., Ginat, D. Goelman, D., Hamer, J., Hitchner, L., Lloyd, W., Marion, B. Jr., Riedesel, C., and Walker, H. Striving for Mathematical Thinking. *SIGCSE Bulletin 33*, 4 (2001), 114–124.
4. Lethbridge, T.C. The relevance of education to software practitioners: Data from the 1998 survey. Technical Report TR-99-06 Rev. 2 (1999). University of Ottawa, Computer Science; www.site. uottawa.ca/~tcl/edrel/EdrelTechReport.doc (accessed Nov. 8, 2002)
5. Lethbridge, T.C. What knowledge is important to a software professional? *Computer* (May 2000), 44–50.
6. Lethbridge, T.C. Priorities for the education and training of software engineers. *Journal of Systems and Software 53*, 1 (2000), 53–71.
7. Litecky, C. and Arnett, K.P. An update on measurement of IT job skills for managers and professionals. In *Proceedings of the 7th Americas Conference on Information Systems* (Boston, MA, Aug. 2001), 1922–1924.
8. McCauley, R. and Manaris, B. Comprehensive report on the 2001 survey of departments offering CAC -accredited degree programs. Technical Report CoC/CS TR# 2002-9-1 (2002). Department of Computer Science, College of Charleston; stono.cs.cofc.edu/~mccauley/survey/report2001/CompRep2001.pdf (accessed Feb. 11, 2004).
9. OCLC Online Computer Library Center; wcp.oclc.org/stats/size.htm (accessed June 4, 2004).
10. Parnas, D.L. Software engineering programs are not computer science programs. *IEEE Software* (Nov.–Dec. 1999), 19–30.
11. Surakka, S. Needs assessment of Software Systems graduates. Doctoral Dissertation. Helsinki University of Technology, Department of Computer Science and Engineering, TKK-TKO-A43; lib.tkk.fi/Diss/2005/isbn9512279517/.
12. Valmari, A. The need for mathematics in software development. *Arkhimedes 2* (2001), 18–22. (In Finnish.)

Sami Surakka (sami.surakka@hut.fi) is a lecturer in the Department of Computer Science and Engineering at the Helsinki University of Technology, Finland.