The following is the first draft of the report on the CASCON workshop on software engineering education that you attended, I have done my best to weave together the various topics discussed at the workshop. Except for key speakers, I have omitted the names of speakers, since I did not have time to accurately record them all. I hope, however, that those who spoke will recognize what they said in what follows. I will be happy to accept any corrections, suggestions and contributions before issuing the final version in a week or so (if changes are extensive, I will recirculate for comments again). IBM have proposed that this become part of an IBM Technical Report.

Please ignore the funny quote symbols 'like this' and "like this". These are a result of converting to ASCII for your easy reading.

Thanks for participating. I look forward to your feedback..
Tim Lethbridge

Industrial Needs for Software Engineering Education: Report from the 1997 CASCON Workshop

FIRST ROUGH DRAFT: February 3, 1998 -- comments welcome.

Led by Timothy C. Lethbridge, SITE, University of Ottawa

1. Background and Goals

The 1997 CASCON Workshop on Industrial Needs for Software Engineering Education was held on Wednesday, November 12 1997. Its objective was to explore ways by which software engineering education could be improved so as to better meet the needs of industry and graduates.

The session was attended by about 21 participants. Presentations were made by David Alex Lamb of Queens University, Burton Leathers of Cognos Corporation, and workshop leader Timothy C. Lethbridge of the University of Ottawa.

The workshop was initiated at the request of CSER (the Consortium for Software Engineering Research) part of whose mandate is to have a positive impact on software engineering education in Canada. The workshop was held at a key point in time for two reasons: 1) There is an increasingly critical shortage of software developers worldwide; 2) Numerous software engineering options and programs [e.g. 1, 2, 3, 4] are being initiated; 3) There is a move to differentiate software engineering programs from computer science programs, accompanied by efforts to establish software engineering as a true engineering discipline [5].

The main themes of the workshop were: 1) What should be taught to software developers, and how it should be taught, with some thoughts about the differences between computer science and software engineering; 2) The importance of industrial experience, and 3) How industry can help improve software engineering education.

The format of the workshop was as follows: Three speakers made presentations; each presentation was intermingled with, and followed by, extensive debate among all participants. An outline of what was said by the key speakers is presented first below; the remainder of the report discusses each of the themes in turn.

2. Presentations

David Alex Lamb spoke first, focusing on two main issues: Firstly he criticized a prevalent attitude that students, "won't be able to do anything at all until fourth year." He said it is surprising how little education is necessary in order to do useful things, pointing out that students should be able to work on serious software problems very early.

Secondly, Lamb suggested what should be considered the fundamentals of a software engineering education. He said that there should be a focus on engineering content: For example, instead of merely working on exercises where students learn about a variety of data structures, students should learn *how to choose* a data structure, and to *justify* its choice. He said there should be a better balance between analysis and synthesis.

In a written summary [6] he delivered following the workshop, Lamb summed up what he believes should be the fundamentals as follows:

'- "Design", that is, creative synthesis (by deliberate choice among alternatives) of solutions for realistic problems under constraints (including calendar time, money, quality, ethical considerations).

- Techniques of coping with complexity

- "Modeling", that is, developing precise but abstract approximations of systems, the use of models to answer questions about the systems, and understanding the limitations of the models.

- Communication skills (written, visual, and verbal). Precision, clarity.

- Critical analysis and reasoning ability

- Working effectively in teams, including planning/monitoring, roles/behavior in groups, communication, documentation as "group memory"

- Attitude: the need to keep familiar with current standards of practice'

Burton Leathers, speaking second, emphasized that his company does not seek people who are "trained", in the sense that they know a lot about the latest languages and technologies. Rather, he seeks people who are capable of learning.

To illustrate, he said that if give a choice, he would hire somebody who has experience solving combinatorialy complex problems as opposed to somebody with mere factual knowledge.

Leathers also addressed what he perceived to be an imbalance in course content. He expressed a concern that today's curricula reflect the history of computer science in the sense that professors teach topics that have been important areas of research in the past. he said "You can tell a great deal about the curriculum in a CS department by looking at the Ph.D. theses of its faculty.". However, he said, the emphases given topics is often not proportional to what is important today: In some cases three to four courses could be condensed into one, and vice-versa.

When addressing the issue of what industry can provide, Leathers lamented that there is very little communication between industry and academe; "CASCON is a singular light in vast darkness". He highlighted one particular idea: That ways be found to have industrial practitioners work for periods of time in universities. However, he said this would be difficult to achieve.

Timothy C. Lethbridge, as third speaker, presented some results of a survey [7, 8] he had taken of 168 practicing software developers, most of which were from the Ottawa area. The respondents to the survey had been asked to characterize what they had learned in university, and to give a corresponding assessment of how useful each topic had been in their careers and how much they know now about each topic (i.e. to assess whether they have forgotten material, or learned more about each topic on the job).

Some key findings of Lethbridge's survey were

* Respondents had learned a great deal about most software topics on the job, the most significant exception being numerical methods, which most participants had largely forgotten.

* Respondents reported that they had learned relatively little about software topics at the core of software engineering such as process standards, metrics, user interface design and object oriented analysis; they had, however, learned much about such topics on the job (most notably testing and configuration management). An audience member at the CASCON workshop pointed out a statistical problem with this analysis however: Some topics did not exist when older respondents were educated, thus we shouldn't use these results to criticize the present state of education.

* The most important software topics, in the opinion of respondents, were software design and architecture, data structures, testing and quality assurance and requirements gathering. Least important topics are artificial intelligence, pattern recognition, graphics and numerical methods. The most severely under-taught courses, relative to importance, were testing and quality assurance, configuration management, project management and maintenance and reverse engineering. The most over-taught courses were numerical methods, programming language theory and artificial intelligence.

* The most important mathematics topics were probability and statistics, followed by predicate logic and set theory. Calculus was felt to be severely over-emphasized.

* Technical writing was considered to be one of the most important topics, more important than all software topics except the top four. It was also considered severely under-emphasized in the curriculum. Other topics considered important were ethics and professionalism and computer hardware architecture.

3. Themes Discussed

In this section, we present the results of the debates that ensured as a result of the above presentations. The debates swept back and forth over three main categories of topics, each of which is addressed in turn:

* What should be taught and how
* Achieving industrial experience
* What industry can provide.

3.1 Software Engineering: Attitudes and Topics

There was considerable discussion about what specific attitudes, teaching approaches and topics ought to distinguish software engineering education. It was felt that those defining curricula should pay more attention to defining the rationale behind their decisions, as expressed here, rather than just creating lists of topics.

The following are most of the points raised:

Approach to Teaching

In accord with David Alex Lamb's comments, it was asserted that the main difference between computer science and software engineering should be style of teaching. Engineering programs focus on design: Applying techniques to practical

problems -- as Lamb said in his introduction: Teaching students to effectively choose and justify solutions to problems. From this perspective, software engineering can be seen as an "eclectic assembly" of techniques from a variety of disciplines whose objective is to achieve practical utility: Software engineering is thus the "craft of software creation."

This distinction between the teaching styles of computer science and software engineering was seen as more important than the differences in topics taught, although topic differences will clearly exist.

The issue of Codified Practice

It is often said that engineering programs teach "codified practice". Although this may make sense for disciplines such as civil engineering, participants felt this can not yet be the case for software engineering. When the issue was raised of whether, consequently, software engineering is real engineering, a participant pointed out that there are other areas of engineering that do not have a well established codified practice because they are still too young and are developing fast: He cited integrated circuit design as an example.

The Core and the Extensions

There was agreement that it ought to be possible to define a core for software engineering programs -- topics that it is widely agreed should be taught. It was pointed out, however, that such material changes over time: For example, said a participant, human-computer interaction wasn't so prominent ten years ago.

It was pointed out that there are two types of organizations working on proposals for a software engineering core: The engineering and computer science accreditation boards, as well as the ACM and IEEE [5].

When a core is defined, it was suggested that extensions can be defined -- groups of courses that give students useful subspecialties. Telecommunications was mentioned as an area where specialization is in demand.

General Attitudes Students Should Be Taught

It was asserted that problem-solving skills, creativity, critical thinking and communications are more important than specific topics in a software engineering education.

One participant said, "Creativity and innovation have created much more economic improvement than has been created by improvements in efficiency". Yet, he explained, efficiency is often inappropriately focused on as a key value. "Imagine the most efficient world without [the innovation of] electricity".

An industrial participant said that the very high need for creative problem-solving skills stems from the fact that, unlike doctors, software developers are rarely presented with the same problem twice

Business Knowledge and Entrepreneurship

One participant emphasized that business or domain knowledge, and hence, an ability to learn about new domains, are key attributes that software engineers must learn in order to perform effectively. He felt this need be less true of computer science graduates.

Other participants emphasized that software engineers must build up a base of skills in various domains of application, so they are ready to tackle new problems. He said, "We would be remiss if we were not preparing people for shaky terrain", i.e. to be able to withstand changes in demand for various types of domain knowledge.

On a related theme, participants said that software engineers should be trained to be entrepreneurial, and to develop an ability to "cost out their time".

Knowledge about When to Call in an Expert

It was pointed out that engineers, as well as medical doctors, are licensed to perform any kind of engineering or medicine. A key part of professionalism, however, is knowing when to call in an expert as opposed to when one can do the work oneself. The participant felt that this should be taught.

Writing and Communications Skills

Confirming the data presented by Lethbridge, there was a strong consensus that students need to learn better technical writing skills. The following suggestions were made regarding how to achieve such improved skills:

* Improve the language requirements on co-op reports, i.e. by paying more attention to writing style, and having industrial supervisors critique a first draft.

* Motivating students about the importance of technical writing by having them write documents they will actually use (e.g. requirements for software they will develop)..

* Have students select their humanities electives from a pre-selected list of 10-12 courses with a strong writing component.

* In general, co-operate with other university departments and industry to tackle the problem of poor writing skills.

It was also pointed out that other kinds of communication (oral and visual) should also be emphasized more strongly. This had been the key finding of a CASCON workshop in a previous year.

Learning Specific Skills vs. Fundamentals

There was some discussion about the importance of learning specific languages and technologies, vs. more 'fundamental' topics such as approaches to design, mathematics etc. One participant mentioned that in his experience, there is a huge difference between software workers 3-5 years after graduation and those ten or more years after graduation. The former tend to believe that a single course was the key to their education (although different people mention different courses). The more senior people, on the other hand have a more balanced perspective and attribute value to many courses.

In the short term, the participant continued, students can benefit significantly from specific skills they have learned. However the advancement of these students tends to be limited by the weaknesses in more fundamental aspects of their education. In the long term, it is important to cover these weaknesses.

Numerical Methods

Following Lethbridge's presentation, there was there was discussion about courses in numerical methods, as an example of a topic that may well be unessential, and therefore should not be mandatory. An industrial participant said, "We don't do much with floating point... Much counting then a division [to yield] a number between zero and one."

It was felt by one participant that courses such as numerical methods are only of "academic interest", and that they are taught because they are defined as being "good for the students". In addition to numerical methods, he said, there is heavy emphasis on operating systems in academia, but not many graduates will actually build an operating system.

In further discussion, it was pointed out that numerical methods are important to some: "To write applications software, people have to become specialized"; and some specialized topics have a heavy numerical content. This attitude was reiterated by a participant who said a students minor is as important as their major -- "Software engineering without domain awareness is empty; awareness influences them and makes them motivated.".

There seemed to be consensus that some numerical methods is needed, "but the issue is, how much?", pointed out one participant. This participant said that

typical programs teach more than necessary. He asked: For most practitioners, "is a couple of weeks enough?" Those who will work in chemistry, physics or engineering domains will need "a lot" he posited.

Project Management and Economics

An international participant said that software students in his country are not taught project management or economics, and instead are given much more mathematics -- which makes them good researchers. Another participant agreed that project management is not essential, "We don't need 100 managers," he said.

However, others disagreed. One person pointed out that the software industry is dominated by small companies, and therefore a considerable percentage of software professionals should acquire these skills.

Mathematics

Lethbridge's survey had highlighted problems with retention of mathematics skills among software developers, and possible overemphasis of these topics. One person referred to mathematics knowledge as "fire insurance." By this he meant that you don't know whether you will need it, but it would be better to have it and not need it than vice-versa. He said it is not a problem that people forget their mathematics: There is a big difference between forgetting it and not knowing it exists. Others reinforced this by saying that mathematics teaches rigor and thinking approaches, which likely remain even if the details are lost. The only counterargument to this was that perhaps rigor and thinking approaches could be taught using more relevant mathematics.

There were suggestions that attention should be paid to reinforcing the utility of discrete mathematics: making it more relevant to software engineers by integrating it with the teaching of software skills.

There was a consensus that nothing be done about calculus: It would not be possible to make changes even if it were desirable.

Teaching Students Using Legacy Systems

The focus of most software education is on new code, not legacy systems. Yet participants felt that instilling in students an awareness of legacy systems is very important, and that industry should help by making such systems accessible.

Two aspects of this issue were raised: Working with legacy code: a) Gives students an awareness of how others have solved problems. b) exposes students to the real-world constraints imposed by legacy

environments.

3.2 The Importance of Industrial Experience

There was discussion about students' industrial experience before graduation, as well as the industrial experience of faculty.

One participant strongly felt that university faculty teaching software engineering should have industrial experience. This could be achieved by hiring people who have worked in industry, taking sabbaticals in industry or having industry practitioners teach courses.

Some universities [2] are making co-op work terms or internships mandatory -- there was discussion about whether this is a good idea for software engineering students.

Among the benefits of mandatory work terms are that it increases the relevance of coursework, and improves students skills. One participant, however, felt that there is low marginal value of making it mandatory: Doing so, he said, increases the length of the pipeline, limits diversity of types of program and erects unnecessary hurdles.

It was also pointed out that work terms should not be seen as a replacement for project courses -- the latter permit students to explore and to reflect on what they are doing.

Another participant said that industry must be trained, " to be responsible users of co-op students." The concern that led to this comment was that many students are not given jobs where they are able to gain useful experience.

3.3 What can Industry Provide to Improve Software Engineering Education?

Since corporations will directly benefit from improvements to software engineering education, it is natural to consider how they can invest in it. There was some disagreement over this, with one industrial participant saying, "rather little". Another participant however said, "Don't let industry off the hook so easily".

The following specific points were enumerated

* Industry can support co-op programs.

* Industry can stop encouraging students from quitting school.

* Industry can invest in students for the "long haul" - funding students for full degree programs, not just occasional training courses.

* Industry can provide leveraged support for equipment and software. In particular, it can provide the software it produces for free or at low cost. Suitable software includes software engineering tools, but also large systems that students can study.

4. Conclusions

The 1997 CASCON workshop provided a worthwhile exchange of ideas about the nature of software engineering, how students can be better educated, and what industry needs and can provide.

The most important single issue discussed was ensuring that software engineering students are taught with an engineering emphasis and mindset. This means such things as focusing on the rationale for design decisions, working in teams, design in the context of various constraints etc.

Participants felt that teaching critical thinking, problem-solving and creativity are.more important than the individual topics that are taught. They felt, however, that it would be possible to define a core software engineering curriculum. Software developers should also be trained to be able to work in a variety of domains.

The workshop identified technical writing (as well as other aspects of communucation) as needing much greater emphasis in software engineering programs. Suggestions to improve this included placing more emphasis on the quality of co-op reports and ensuring that elective courses have a strong writing content.

Although some types of software engineers need numerical methods background, it was generally agreed that most only need a few weeks of exposure to this topic. It was also concluded that more emphasis should be placed on discrete mathematics, probability and statistics than is currently provided. As for calculus: although there is evidence that software engineeers do not need as much as they receive, no change could or should be made at this time.

The participants felt that encouraging students to gain work experience through co-op programs or internships is important, but that such programs should not be mandatory. It was also felt that industry should be encouraged to give students more challenging experiences during work terms.

Finally, it was felt that industry could improve the quality of education by encouraging students to improve their education in the fundamentals (as opposed

to 'hot technology') and by providing software and personnel to help university educators provide a better experience for students.

References

[1] School of Information Technology and Engineering, University of Ottawa, (1997), "Proposal for a Software Engineering Program", http://www.site.uottawa.ca/~tcl/sepc/CompleteSEProgram.pdf

[2] J. Fernando Naveda (1997) "Crafting a Baccalaureate Program in Software Engineering", 10th SEI Conference on Software Engineering Education, Virginia Beach, pp. 74-79.

[3] G. Ford, (1996) "The SEI Undergraduate Curriculum in Software Engineering", Software Engineering Institute

[4] J.M. Atlee et al., (1996) "A Joint CS/E&CE Undergraduate Option in Software Engineering", Proc. Conf. on Software Engineering Education.

[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] D.A. Lamb, Position report prepared following CASCON, personal communication.

[7] Lethbridge, T.C. (1998) "A Survey of the Relevance of Computer Science and Software Engineering Education", 11th Conference of Software Engineering Education and Training, Atlanta, http://www.site.uottawa.ca/~tcl/papers/CSEET/CSEET98finalIEEE.pdf

[8] Lethbridge, T.C. (1997) "The Relevance of Software Education: A Survey and Some Recommendations", submitted for review to Annals of Software Engineering