Integrating the science of computing into undergraduate engineering curricula

Ian F.C. Smith, F.ASCE
School of Architecture, Civil and Environmental Engineering (ENAC),
Applied Computing and Mechanics Laboratory (IMAC), Station 18
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, 1015, Switzerland
Ian.Smith@epfl.ch, Tel. 011 4121 693-5245

ABSTRACT

Engineering computing is much more than a specialized skill that enables engineers to create programs and use commercial tools. The science of computing involves the study of representation and reasoning strategies as well as fundamental topics such as computational complexity. The core ideas associated with such topics are expected to have an important impact on decisions related to computing during the careers of current engineering students. This paper describes a course on the fundamentals of computing that has been taught to second-year civil-engineering undergraduates in Switzerland for ten years and more recently, to undergraduate and graduate students in India and the USA. Practical outcomes include a sustainable knowledge of fundamentals for competence across a wide band of technologies, agile adoption of future developments, better collaboration with computer scientists for large, and often out-sourced, projects in practice and more informed purchases of software.

Keywords: Education, Computing, Engineering Informatics, Emerging Computing Technologies

INTRODUCTION

Computing solutions are required to help engineers assess complex solution spaces and manage the tradeoffs that occur in modern engineering activities. These solution spaces are difficult to describe using formulas – especially when additional constraints and criteria are imposed by new challenges. In the current context of multi-dimensional requirements for global sustainability and with increasing uncertainty in environmental actions, the need for appropriate computer support is growing.

Unfortunately, the vast majority of today's engineers are unprepared to leverage computing applications effectively because they lack fundamental computing knowledge. The programming course they usually take is not about the science of computing; the focus is on the skill of writing code. While later on, they may learn specialized computer tools through optional courses and for projects; this builds only on application-specific skills. Engineers that graduate following such education are often unable, for example, to evaluate new computing opportunities and they may not work effectively in project teams with computer specialists.

This paper describes an alternative to computer-programming courses that is intended to prepare engineers for difficult tasks that they will encounter throughout their careers. This alternative is a course on the fundamentals of computing, which has been taught to second-year civil-engineering undergraduates in Switzerland for ten years. The next section includes a description of an example of topics in this course and the following section outlines recent teaching experiences in India and the USA as well as plans for the future.

COURSE CONTENT

Many course modules summarize kernel scientific knowledge of a sub-area of computer science while concentrating on aspects that are relevant for engineering tasks. Topics include engineering inference, O-notation for complexity classification, normal forms of databases, stochastic-search methods such as genetic algorithms and simulated annealing, data structures and network science (random and scale-free networks), distributed computing, and machine learning methods such as artificial neural networks, clustering as well as entropy-based induction. The last lecture deals with choosing the right tool for the task. This is where examples from research and practice are used to illustrate successful implementations.

Table 1 provides the modules given in the current offering of the course. Java is taught at the beginning as an example programming language so that subsequent exercises within other modules can include complete examples.

Table 1 Course Modules at EPFL

Module	**Hours (Course + exercises)**
Engineering tasks and inference	2.5
Programming languages and the example of Java	15.5
Data structures	2
Computational complexity	6
Databases	5
Search and optimization	8
Reasoning strategies	5
Machine learning	4
Distributed systems and applets	2
Which tool for which task?	1

Table 2 provides examples of take-home messages for selected modules. At the bottom of this table there are also examples of more general messages that are repeated throughout the course.

Table 2 Examples of important messages in the undergraduate course

Course module	Take-Home Message
Engineering tasks	Many important engineering tasks are best supported by computers that provide choices, not solutions.
	Simulation and engineering analysis tasks are different from design and diagnosis tasks on a basic logical-inference level.
Computational complexity	Computers are unable to perform certain tasks and faster computers cannot help.
	Small changes in programs can have huge effects on their sensitivity to task size.
	There are cases where parallel computing does not help.
	There are tasks for which the computational requirements are nearly independent of task size.
Database design	The best design depends on the way a company does its business.
	Small changes in business processes can reduce the ability for unchanged databases to be updated reliably.
Data structures	Stacks may lead to exponential complexity.
	Scale-free networks are robust for risks of random failures.
	Scale-free networks are vulnerable to targeted attacks.
Search and optimization	Use the simplest method that can be justified by the shape and form of the objective function.
	Gradient methods are good (and fast) when there is only one minimum (or maximum).
	Stochastic methods (such as evolutionary algorithms) are appropriate when there are many local minima (or maxima) and when exhaustive evaluation is not feasible.
Machine learning	Verify, verify, verify … and avoid extrapolation beyond the range of the training data.
	Avoid out-of-date and badly distributed training data.
General course messages	It is all about maintaining performance when things change, and changes always happen.
	Mental models of engineering tasks should guide representation and reasoning strategies.
	Context and local knowledge is important: there is no "silver bullet" type of representation and reasoning that always works best.

With the knowledge gained in this course, engineers are better prepared to take advantage of computing in addressing future engineering challenges because they are able to combine experience with data using approaches that are built on scientific concepts. Course slides are available for free downloading on the ASCE Global Center for Excellence in Computing website (www.ASCEGlobalCenter.org). The material is also supported by a text book (Raphael and Smith, 2003) that is already used by several universities in the USA and elsewhere. After more than 1000 copies sold, the book will be published in its second edition in 2013.

This course versus a programming-skills-only course

Students have increasing difficulty in linking the content of an exclusively computer programming course with what is required of them in practice. Over ten years, more than 500 students have provided positive feedback regarding their reaction to this course in part due to modules being linked to real examples in engineering practice and in engineering research. Contrary to typical student evaluations of traditional programming courses, evaluations of this course have been consistently very good.

Aside from the science-versus-skills debate, there are practical advantages of this course compared with a course that teaches only computer skills. Since hardware and software technology is undergoing continuing evolution, the course content of a skills course (such as one that teaches the functionality of EXCEL, MATLAB and AUTOCAD) must constantly adapt to new versions and functionalities. The content of this course evolves much slower since it focuses on computing fundamentals. Therefore, the risk of instructors providing soon-to-be obsolete information is much lower. Another practical advantage is that more informed purchases of software are possible because the underlying strengths (and weaknesses) of representation and reasoning methods are better understood.

Finally, increasing numbers of instructors recognize that the amount of computer-skills teaching in modern curricula should be reduced, especially in top-tier universities. Many students entering such universities already know how to program. We have found that those who do not are able to learn within a few weeks in exercise classes where fundamental concepts are illustrated through creation, adaptation and using small JAVA programs. Following this course, students readily teach themselves tools such as MATLAB in a few hours. As teaching hours are reduced for computer skills, they need to be replaced by teaching hours for fundamental aspects of computer science that are relevant for engineers.

RECENT TEACHING EXPERIENCE AND FUTURE PLANS

Over the past year, parts of this course have been presented to undergraduates and graduates in universities in the USA and India. The undergraduates in India (IIT Gandhinagar) asked the local professor in charge if they could formally evaluate the course so that they could express in writing that they appreciate the content of this course and that they would like to have more instruction along these lines. Out of a class of around 30, only one student borderline-failed the test at the end.

In the USA, approximately twenty first and second-year students at George Mason University were taught the computational complexity module in November 2011. It was possible to introduce O-Notation, discuss algorithmic foundations of information search, cryptography and the optimality question (P=NP?) without loss of understanding. At this level, it is most important to go through numerical examples in order to demonstrate, for example, the behavior of algorithms that have various O ratings.

Also in November 2011, graduate students at Carnegie Mellon University (CMU) were given eight hours of teaching related to modules of engineering tasks, computational complexity, data structures and database design. Since many already had prior exposure to much of the course material through fundamental courses already available at CMU, concepts were delivered at a faster pace. The consensus was that in spite of familiarity, they learned new things and that the material could be assimilated at an undergraduate level if given at a slower pace.

Future plans involve more teaching in the USA and time permitting, in the UK and in Asia. Also, slides will be revised to reflect modifications and additions to the new edition of the text book (Raphael and Smith, 2003). Finally, more exercises and solutions sheets will be available on www.ASCEGlobalCenter.org for free downloading.

CONCLUSIONS

An increasing number of universities are revising their computing curricula for engineers in order to reflect aspects such as already acquired skills of students, increasing needs for wide-band competence and agility requirements when new technology emerges. This course provides an opportunity to address these aspects through adopting a strategy of teaching fundamental computer science concepts that are relevant in engineering contexts. Such knowledge is expected to remain useful to engineers throughout their careers.

ACKNOWLEDGEMENTS

The author is grateful to Professors Jim Garrett, Carnegie Mellon University; Sudhir K Jain, Bireswar Das, IIT Gandhinagar; and Liza Durant, Deborah Goodings and Tomasz Arciszewski, George Mason University and for their support and their willingness to experiment with new teaching material. Prof. Benny Raphael, NUS developed early teaching modules and co-authored the text book while he was at EPFL. Finally, the author is grateful to the teaching assistants who helped improve course content, Y. Robert-Nicoud, S. Saitta and L. Rhode Barbarigos.

REFERENCES

Raphael, B. and Smith, I.F.C. (2003) *"Fundamentals of Computer-Aided Engineering"*, Wiley, 306p.