A COURSE IN COMPUTER-AIDED ENGINEERING FUNDAMENTALS

Ian F.C. Smith[1] and John Miles[2]

[1]Ecole Polytechnique Fédérale de Lausanne (EPFL), Structural Engineering Institute, Station 18
1015 Lausanne EPFL,Switzerland, Ian.Smith@epfl.ch
[2]Cardiff University, School of Engineering, The Parade, Cardiff CF24 3AA, UK

Abstract

Civil Engineers have over the last 50 years moved from being pioneers in the use of computers to being the least computer literate and the least progressive of all engineers. This paper looks briefly at the reasons for this transformation, linking part of the problem to standard civil-engineering computing education. Computing education of engineers should be structured so that it provides an appreciation of the basic aspects of computing and information science. A set of teaching modules is proposed in order to improve teamwork with computer scientists in the workplace, increase capacity for informed purchase decisions and provide more technical agility for future developments in information technology.

Introduction

Engineers have difficulties performing tasks that involve computing because they lack a fundamental knowledge of computer science and software engineering. This means that when they use standard computer packages, they fail to understand strengths and weaknesses because of a lack of fundamental knowledge of basic concepts such as data structures and computational complexity. Also, a lack of knowledge of the range of available techniques leads them to use inappropriate methods. For example engineers may attempt to search for good solutions using unsystematic repetitive runs of simulation packages rather than, for example, stochastic search algorithms. After generation of a small number of simulation results they thus claim to have found the optimum solution. Such lack of knowledge results in badly thought out and poor quality solutions for clients.

While teaching computational mechanics within engineering degree schemes is well developed, fundamental computing education is scarce at the undergraduate level. This paper looks at a new approach to computing education for engineers. Rather than, as occurs in many universities, create under-trained programmers and engineers who have a rudimentary knowledge of a few current or legacy software tools, this course aims to provide a fundamental understanding of important and fundamental topics in computing. Engineers should think before they type. Computing is both a science and an engineering discipline, not just a skill.

On July 14, 2005 the Global Center for Excellence in Computing [1] was formally established in Cancun Mexico by the American Society of Civil Engineers. Its mission is to provide global leadership in research, education and technology transfer and to encourage the best possible use of computing in civil engineering. Although initially seed funded by the American Society of Civil Engineers, one goal of the Center is to become self sustaining financially within the next three years. This will be done through revenue from a one week ASCE Certificate Course on the fundamentals of computing. Funds from teaching activities will be used to sponsor global research projects. This paper describes the outline of the first version of this course that is intended initially for practicing engineers.

Motivation

Although civil engineers were the first engineers to use computers in 1956, they have been slow to adopt new methods and technologies. One explanation for this could be described as the paradox of being first. Early use has lead to complacency regarding the importance of advances in information technology and this has subsequently lead to inappropriate use and unnecessary costs. Also, the control of computing education in most civil engineering departments has been in the hands of the computational mechanics community who see little need for anything but FORTRAN (C and Java for the more adventurous) and algorithms are limited to those associated with finite element analysis.

Thus to many academics, computing is a relatively simple subject consisting of the teaching of loops, input and output and conditional statements. Such a simple and well established topic is not the sort of material that should be taught in a high level degree scheme and this has damaged the image of computing in civil engineering resulting in insufficient allocation of appropriate teaching time within undergraduate curricula. Our young engineers are ill prepared to recognize opportunities and benefit from new technologies. In practice, the use of computing for high value-added activities in civil engineering is now the lowest of all engineering domains. Having been the first engineers to apply computing, we are now last in terms of our comprehension of the science and use of the technology.

Over the past decade, attempts to remedy this situation have met with mixed results. For example, arguing that civil engineers rarely program computers in their professional lives, some universities have eliminated programming courses from their study programs. In their place, courses in specialized areas, such as construction, have been given more time to introduce special purpose software tools for use in project courses.

While such training may make engineering students more immediately useful, uncontrollable events may make these efforts futile. For example, new software versions may be released, software companies could go bankrupt and employers may adopt other software for their work. Also this sort of training, where one is mechanically taught to use a tool rather than understand and appreciate the technology, is the sort of training one would expect to be given to technicians rather than professional engineers. Finally from a long-term perspective, such education is a weak preparation for future developments in information technologies.

How can civil engineers benefit the most from current and future information technologies? This course provides a solution. Civil engineers need to have a better knowledge of those fundamental aspects of computer science that are relevant to civil engineering. Such aspects are the basis of advances downstream when new information technologies are developed. In this way engineers become better prepared to judge current and future proposals. Furthermore, while most engineers may not program, many will be asked to participate in teams with computer specialists for tasks such as information representation, project support, knowledge management and purchase of computer infrastructure. Since teamwork is most effective when players have close technological cultures, in addition to showing engineers how they need to participate, this course will help build trust between team members and thus avoid costly mistakes.

The course, which is described below, is organized into modules that are derived from fundamental topics in computer science. Taught by civil engineers with much practical experience, examples are drawn from civil engineering cases and practical experience. At the end of modules, exercises give students the opportunity to apply what they have learned to civil engineering situations.

Teaching Plan

A one-week course has been created to be given to engineers working in practice at the end of this year. The total course time is 32 hours over 4.5 days (Monday morning to Friday noon). In its first version, it is a distillation of a course given at EPFL [2] and Cardiff University. The following nine modules will be offered using many civil-engineering examples. Apart from the examples, the material is core knowledge within computer science courses that may be given over 1-2 semesters. Only knowledge that is unquestionably relevant to computer-aided engineering has been retained. After each 2-3 hour package of lectures, students receive an hour to complete exercises that were largely inspired from civil-engineering tasks. All exercises are performed in the classroom in the absence of a computer. This serves to underline that many concepts are independent of hardware and software technologies and more importantly, that much thinking is beneficial when it is done upstream from computer use.

1. Identifying the type of problem (3 hours)
   - Fundamental logic in open and closed worlds
   - Importance of abductive tasks and their difference with respect to simulation and analysis tasks
   - Engineering tasks – identification of deductive, inductive and abductive tasks in engineering
   - Setting the scope for computer-aided engineering challenges of the future

Exercises: Matching engineering tasks to types of logic

2. The limits of computing - dealing with complexity, who / what makes the decisions (4 hours)
   - Why computers cannot do everything: estimating trends in execution time, why computers can do some things very well
   - Big O notation
   - Why bigger computers will sometimes not help: classification of functions
   - Tractability and algorithm optimality
   - Determining the complexity of tasks
   - P, NP and NP completeness
   - What do engineers do?

Exercises: Using O-Notation to estimate the effect of task size

3. Representing information (5 hours)
   - Garbage in → garbage out
   - Data structures – lists, graphs, trees, stacks, queues
   - Knowledge structures
   - OOP – classes, encapsulation, abstract, polymorphism, message passing, decomposition hierarchies, agents
   - Ontologies – creation, incremental growth, maintenance

Exercises: Cycles in message passing, ontology creation

4. Designing relational data bases (4 hours)
   - Relational model
   - Functional dependencies
   - Normal forms
   - Up-date anomalies

Exercises: Design of a robust database from functional dependencies

5. Optimization, search and exploration (6 hours)
- Although engineers can rarely optimize in an open world, some mathematical techniques help the search for good solutions
- Types of optimization problems
- Objective functions, goals and constraints
- Gradient methods
- Stochastic search – evolutionary algorithms, simulated annealing, probabilistic methods
- Multi-criteria optimization: Pareto and other approaches

Exercises: Gradient optimization, solution space creation, stochastic search

6. Learning from experience (4 hours)
- Company experience is valuable
- Case based reasoning – five stages – difference between CBR and information retrieval
- Statistical learning - ANN
- Inductive learning – ID3, Entropy
- Data mining

Exercises: Case based reasoning, ANN, Entropy

7. Decision support (2 hours)
- Why keep knowledge?
- Types of knowledge
- Rule bases – chaining
- Linking to analysis software
- Appropriate presentation of information
- Agents
- Challenges and risks

Exercises: Rule chaining, updating, agent information cycling

8. Graphics and geometrical modeling (2 hours)
- Why bad graphics leads to higher costs
- Geometrical representation: Parametric CAD, sweep, Bézier, boundary representations, winged edge representations, geometric solid modeling
- Displaying information: projections, shading, lighting, textures

9. Distributed systems (2 hours)
- What is wrong with mainframes?
- Why link computers together?
- Client-server architectures
- World Wide Web,
- Grid computing
- Peer-to-peer networks
- Computer Supported Collaborative Work
- Agents

Within all of these modules, several recurrent themes emerge. Firstly, software has to be designed so that it is robust in terms of modifications and changes. Since the most dominant constant in civil engineering practice is the fact that *everything changes*, representations and reasoning strategies must be selected so that typical modifications require a minimum of modifications elsewhere and that no information is lost. For example, relational data bases in their first and second normal forms should be avoided, changes should not induce exponential complexity in algorithms and gradient search strategies should not be exposed to changed objective functions that have multiple local minima.

A second theme is that even though practicing civil engineers may never carry out a full-scale computing implementation alone, they must participate with computer specialists in order to ensure that civil-engineering knowledge is modeled correctly. For example, only civil engineers can provide the domain knowledge necessary to formulate heuristics for NP complete problems, functional dependencies for relational data-base design, data structures for the right internal representation and the best design for client-server configurations. Leaving these decisions to computer specialists is a recipe for poor returns on investment. While testing may ensure that systems run satisfactorily at the beginning, high maintenance costs eventually cripple systems that do not adequately account for the way engineers work and the way engineering information has been represented and understood by the profession.

A third recurrent theme is that black-box computing, where engineers provide input and collect the output without knowledge of how the output was created, is not appropriate for full-scale practical systems. Engineers perform abductive tasks in open worlds where they are legally responsible for their decisions. They need to know exactly how, and under which assumptions, computer output is created. They must also be able to introduce information at intermediate stages. This means that, for example, data structures should be as close as possible to those currently employed in codes of practice, constraints on search spaces should reflect realistic design constraints, visualization should be as realistic as possible and user interfaces should be compatible with the knowledge and capacity of professional civil engineers.

A fourth theme involves justifying complication. There are many ways to do the same thing with different combinations of representations and reasoning strategies. Engineers need to choose the simplest possible combination for the task at hand. Greater complication has to be justified. For example, for an optimization task, if the objective function is monotonic, only boundaries need to be explored for optimal values. If there is only one minimum (maximum) within the problem constraints, gradient optimization methods are sufficient. Stochastic (evolutionary) methods are justified only when it is known that that there are multiple local minima (maxima) within problem constraints that define large search spaces. If the search space is small, exhaustive search is best.

All of these themes are treated in several places throughout course. In this way, the course modules are not understood to be independent capsules of unrelated bits of information science for engineers. Through such recurrent ideas, civil engineers are shown that the have an essential role in the development of computer systems in civil engineering. For large computer systems, they also see how their knowledge must be integrated into a team effort with computer specialists in order to ensure successful long-term results.

Conclusions

Even after decades of advances in IT, applications in civil engineering are often unsystematic combinations of commercial tools and macro programs that are developed in-house for personal use. There is also a woeful lack of knowledge regarding what representations and reasoning strategies are available and in what contexts these techniques are best suited. In spite of this, the importance of computing is increasing as engineering firms strive to maintain competitiveness. It is therefore important to offer fundamental computing courses to practicing engineers and to modernize engineering curricula. Faced with the unknown future of computing applications, knowledge of computing fundamentals is important for engineers to maintain technical agility and to avoid unnecessary costs due to poorly performing and inappropriate computer systems.

Acknowledgements

Professor Tomasz Arciszewski (George Mason University) was instrumental in the formation of the ASCE Global Center for Excellence in Computing and all members of the Executive Committee of the Technical Council for Computing and Information Technology, ASCE have given their full support. While they were at EPFL, Prof. B. Raphael (NUS, Singapore) and Prof. K. Shea (TU Munich, Germany) helped develop the postgraduate seminar that has eventually led to the development of much of this course material. Also, contributions from Prof. S. Fenves, Dr. B. Domer and Dr. Y. Robert-Nicoud are gratefully recognized.

References

[1] http://www.asceGlobalCenter.org/
[2] Raphael, B and Smith, I.F.C. "Fundamentals of Computer-Aided Engineering", John Wiley, 2003, 306p.