Why integrate computational thinking into a 21st century engineering curriculum?

C. Mohtadi[1]
MathWorks Ltd, Cambridge, UK
coorous.mohtadi@mathworks.co.uk

M. Kim[1]**, J. Schlosser**
MathWorks GmbH, Ismaning, Germany
<mischa.kim, joachim.schlosser>@mathworks.de

Conference Key Areas: Curriculum development, Education concepts specific for engineering education, Industry and engineering education

Keywords: Integrated Curriculum, Simulation, Project-Based Learning, Tools

INTRODUCTION

Engineering educators today are facing a number of challenges [1]. Students entering engineering courses are less skilled in science, technology, engineering, and mathematics (STEM) subjects [2] despite being brought up in an environment with all forms of computational and electronic devices. A key challenge faced by engineering educators is to prepare students with the necessary skills and knowledge to work in multidisciplinary design teams upon graduation, solving complex problems using computational tools.

To foster such a transformation, university courses need to integrate computational thinking [3] into all aspects of the engineering science curriculum. Computational thinking skills include reformulating seemingly difficult problems, reduction, embedding, transformation and simulation in conjunction with abstraction, and decomposition in tackling a large complex task. The experiences of a number of leading academic institutions demonstrate that when this is done systematically, students quickly learn mathematical concepts early on using symbolic and numeric software tools. Further, the students acquire a deeper understanding of programming and systems engineering with hands-on project-based learning linked with real hardware. Students also learn to think independently, investigate and explore environments, and apply tools used by practicing engineers.

Programming and numerical simulation platforms such as MATLAB® and Simulink® can act as catalysts to bring about the transition from traditional engineering courses to modern courses in which the requirements of engineering analysis as defined by accreditation bodies (e.g. Engineering Councils, EUR-ACE®) are met. Such platforms enable the integration of computational thinking and tools to promote a deeper understanding of engineering principles through spiral curricula. They also support the use of simple “Apps” to bring concepts to life before encouraging and empowering students to develop their own. By helping generate excitement, establish purpose early, and introduce skills that are reinforced later, the integration of computational thinking serves to engage students at progressively higher levels, enhance learning, and increase retention.

This paper explores these themes through a number of case studies from universities around the world, sharing some of their practices and key outcomes. Effective and counterproductive

[1] Corresponding Author
Coorous Mohtadi, coorous.mohtadi@mathworks.co.uk, Mischa Kim, mischa.kim@mathworks.de

patterns of using computational tools are discussed. Today's engineers need to find solutions for great challenges and computational thinking a key prerequisite.

1 ENGINEERING EDUCATION CHALLENGES

Much of engineering education requires a sound basis of mathematics as well as practical understanding and insight. Many recent changes in secondary education resulted in a perceived drop in the underlying skills and knowledge of the students in such areas as higher level mathematics and programming skills. This, combined with students coming from a larger geographical distribution, results in a much less uniform initial skill set among students in a course [2].

A second challenge is tied to the increased complexity of engineering systems (and systems of systems) that newly qualified engineers will be expected to analyse, design, and test in their career. It is no longer adequate merely to have an understanding of the underlying principles. A graduating engineer now must have a much deeper working knowledge of the processes involved in designing these complex systems.

Addressing the growing gap between the uneven skills of incoming students and higher expectations of graduating students within the limited duration of typical engineering courses is an increasingly difficult challenge for institutions of higher education.

1.1 Students' variable STEM skill set

A recent report from the House of Lords, Select Committee on Science and Technology on Higher Education in Science, Technology, Engineering and Mathematics (STEM) subjects [2] highlighted the skills gap that universities are seeing. It states, "*In 2006, the Royal Society argued that the gap between the mathematical skills of students when they entered HE and the mathematical skills needed for STEM first degrees was a problem which had become acute. Two reasons were suggested to explain the gap: first, lack of fluency in basic mathematical skills; and, secondly, the fact that some A level syllabuses allowed topics to be excluded which were relevant to some first degree courses. The evidence we received suggested that the problem remains.*

In addition to the skills gap at the school-HEI interface, we also received evidence that graduates were often found to lack the numeracy skills needed to succeed in the workplace, an issue confirmed by employer surveys conducted by the CBI which identified a shortage of students with adequate maths skills."

1.2 Complexity and interdisciplinary design

Complexity is an inherent part of most engineering disciplines. Moreover, the increase in complexity and the interdisciplinary nature of engineering education go hand in hand. Two key recommendations from the National Academy of Engineering for educating the engineer of 2020 [4] are:

- Whatever other creative approaches are taken in the four-year engineering curriculum, the essence of engineering–the iterative process of designing, predicting performance, building, and testing–should be taught from the earliest stages of the curriculum, including the first year.
- Engineering schools should introduce interdisciplinary learning in the undergraduate environment, rather than having it as an exclusive feature of graduate programs.

An associated challenge is inclusion of computational thinking early on in the design process. Computational thinking skills include reformulating seemingly difficult problems, reduction, embedding, transformation and simulation in conjunction with abstraction, and decomposition in tackling a large complex task. These skills are indispensible in modern engineering practice. The design of the various systems in a passenger aircraft [6] provides an illustrative example. Fuel systems in modern aircraft are substantially more complex than those two decades ago. Computational thinking underpins the design of complex systems by enabling teams to validate requirements early, communicate the functional specification to suppliers, and complement written requirements in conformance with appropriate standards. Computational models can be reused to create desktop simulators, commission hardware-in-the-loop test rigs, run a virtual integration bench, and demonstrate system functionality to customers.

1.3 Accreditation

A major challenge in designing engineering courses is satisfying national and international accreditation bodies such as UK Engineering Council and European Network for Accreditation of Engineering Education (ENAEE). EUR-ACE® [5] covers six areas: Knowledge and Understanding, Engineering Analysis, Engineering Design, Investigation, Engineering Practice, and Transferable skills. In most of the above categories, computational thinking will enhance an engineer's capacity and capability. Moreover, mathematical analysis, and computational modelling should be integrated into Engineering Analysis: "*Graduates should be able to use a variety of methods, including mathematical analysis, computational modelling, or practical experiments … [graduate should have] the ability to use their knowledge and understanding to conceptualise engineering models, systems and processes".[5]*

2 COMPUTING INTEGRATED INTO THE CURRICULUM

Integration of computing into the engineering curriculum is not a new idea [7]. However, in the past, the complexity and unsuitability of the software tools for first year engineering students was a fundamental barrier. The following sections present examples that illustrate the advantages of integrating computing into curricula.

2.1 Symbolic tools to teach mathematics and elementary programming

In many universities mathematics faculty members teach the concepts to engineers and scientists alike. Moreover, maths is one of the key subjects in teaching engineering. One way to foster students' early adoption of software tools is in using the tools to teach mathematics[8]. Symbolic tools are used to reinforce the mathematical concepts and to encourage independent learning. Symbolic computer algebra software packages such as the MuPAD® notebook provide a smooth progression from pencil and paper to computer-based tools using the same notation.

As an example, a first-year course in exploring mathematical and engineering concepts with appropriate software tools provides students with an early introduction to self-directed learning while building foundational knowledge they will need in their other math and science courses. Taught using MATLAB and Symbolic Math Toolbox™ with its MuPAD® notebook interface, a typical two-term course introduces students to the use of symbolic computation in modern mathematics. The course director feels all maths undergraduates must have experience using a computer algebra system, regardless of what branch of maths they might specialize in. An immediate benefit of taking this course in the first year is that students can use the software to solve problems and check their answers in other courses. Another advantage is that the students not familiar with numerical computation can start from familiar grounds of symbolic manipulation. In the longer term, exposure to software tools is beneficial for students in their third- and fourth-year projects, in their postgraduate research, and as a skill to highlight on their resumes.

Another university uses the MuPAD notebook to teach electrical engineering principles. Here the MuPAD notebook is used to convey such concepts as Fourier series, Taylor expansion, vector products, ordinary differential equations, and RLC circuits [9].

2.2 Software tools to help implement a CDIO™ based programme

CDIO™ (Conceive, Design, Implement, and Operate) is a student-centric methodology designed to deepen and widen engineering students' understanding through a series of well-conceived design exercises.

In an introductory mechanical engineering course [10], first-year students learn the basics of computation. Working in MATLAB, they define a set of equations that describe an engineering problem, solve the equations, and interpret the results by generating plots and graphs. In upper-level courses, students build upon the skills and techniques they have acquired, using the same software environments to complete increasingly complex assignments.

In a mechatronics course [10], for example, students use Simulink® and Stateflow® in the design phase to model and simulate a robot controller that controls multiple motors based on a variety of sensor inputs. In the implementation phase, the students generate code from their models using Embedded Coder® and deploy the code to an embedded target for hardware-in-the-loop simulations and real-time testing.

Students go on to use MATLAB and Simulink to complete engineering capstone projects in which they apply the theory that they have learned to design, build, and operate a production system.

2.3 Model-Based Design, problem-based learning, and industry collaboration

One challenge for instructors lies in helping students move rapidly from learning a theory to applying it. Often, university students learn a great deal about maths and algorithms without acquiring an appreciation for what it takes to put that knowledge into practice—for example making an algorithm fly in a real aircraft. Closing this gap is needed to produce graduates who are outstanding candidates for partner companies or for internal research teams.

The solution consists of complementing lectures with interactive demonstrations and tutorial-based simulators. The students employ software tools and basic programming skills to determine steady-state conditions and use trim routines. They use simulation tools when no simple analytical closed-form solutions exist. Later, they build models and deploy code to use with quadcopters for in-lab flight tests. [11]

2.4 Motivation and making mathematics real

A number of universities use project-based learning with hardware such as LEGO® Mindstorms® NXT [12], Arduino®, Raspberry Pi®, or Beagleboard® to motivate students, encourage their creativity, and present them with real engineering challenges [13].

The biggest barrier in using such hardware with first- and second-year students is that the hardware typically requires programming embedded systems using low-level tools. With higher-level tools, students move smoothly from the design and simulation environment to embedded systems without having to learn the intricacies of such devices. These details can be part of follow-on projects to complement the fundamentals acquired on the first project. With this approach, students can draw upon their mathematical modelling and analysis skills, learn how to structure algorithms and solutions in computer programs, and solve problems by computational thinking before getting into the details of low-level software tools and compilers. In addition, the students follow processes similar to those widely employed in industry.

2.5 Spiral curriculum, scaffolding theory, and engineering curriculum

With a spiral curriculum, students start with authentic design, build, and test tasks early in the first year. Instead of focusing for relatively long periods on specific narrow topics, a spiral curriculum exposes students to a wide variety of ideas repeatedly. Each time, the skills and depth of understanding reach a new level. Instructors encourage students to go beyond their usual comfort zone and stretch their abilities.

For example, students will learn basic concepts with mathematical modelling in the first year and revisit these concepts in greater depth in subsequent years.

As an example, educators can use simple “Apps” [14] to convey various educational concepts. Students originally use the “App” to learn about a concept, such as the solution to a differential equation. Subsequently they learn to extend the App to compute the solution using alternative methods, and finally they develop Apps of their own, which they can add to the library.

2.6 Using an integrated tool suite as a multipurpose, pedagogical, Swiss army knife

One approach of using MATLAB and Simulink in an integrated engineering curriculum is shown in Fig. 1. Rather than starting in the first year of study with abstract programming constructs, students are introduced to computing via simulation, and experience engineering concepts directly using hardware platforms such as LEGO® Mindstorms® NXT. This introduction fosters retention and helps bridge the knowledge gap in mathematical and engineering topics that are typically covered in later semesters. Programming skills are then honed in the early semesters in math and physics courses, which serve as prerequisites to enter the engineering sequence. As the students' comfort level in programming rises and the complexity of engineering problems increases, the computing component is extended to include a simulation platform such as Simulink. Engineering curricula typically culminate in a series of applied lab-based courses in which students are encouraged to demonstrate the problem solving skills they have acquired with full-blown, real-world hardware projects.

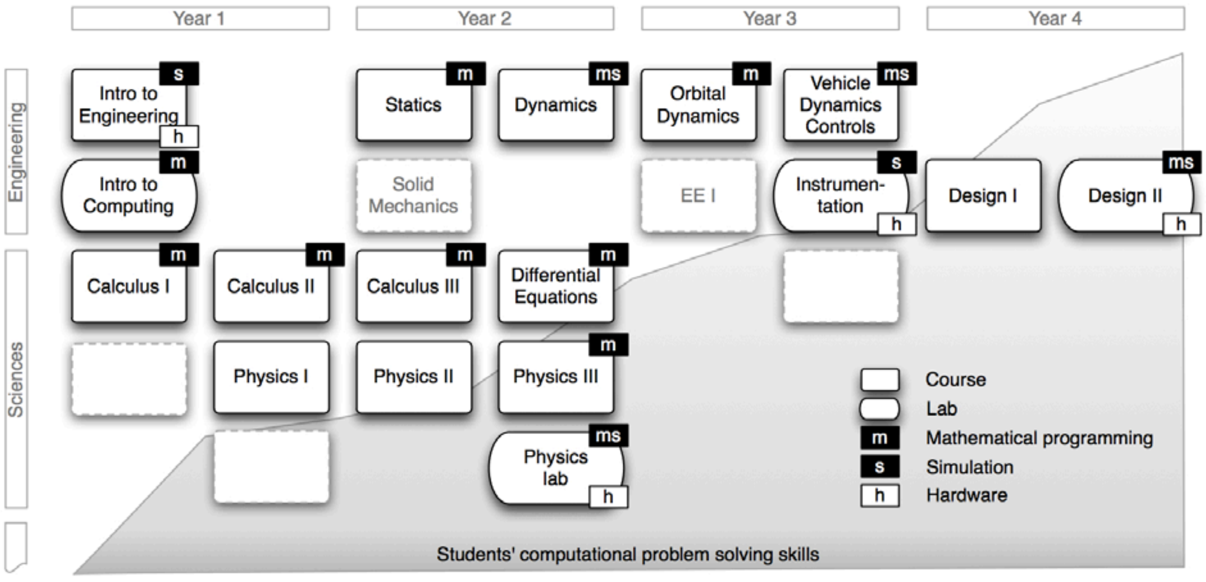

Fig. 1 An example integrated engineering curriculum using MathWorks tools and hardware

3 GOOD PRACTICES IN DESIGNING ENGINEERING CURRICULA

The primary aim of using computational techniques in teaching is to reinforce learning of key engineering concepts. However, tool competency can become a barrier in developing the concepts when using new software environments. With this in mind, it is important to follow a number of good practices that ultimately lead to higher levels of understanding.

Some typical problems with using software tools in education are:

- Expecting first year undergraduate engineers to motivate themselves to learn the software tools by themselves
- Teaching the tools in the first year but not asking students to use them until they encounter their projects in their third years or later
- Not using the tools in the context they are used in industry.

Many institutions have made significant strides in the use of software tools and new technology in their courses, lab assignments, and student projects. This section highlights some of the best practices from those institutions.

3.1 Good Practice 1: Reuse the same technology in multiple courses via scaffolding

Within one engineering department, multiple faculty members coordinate the efficient reuse of the same technology and related concepts across multiple courses in multiple terms and years as illustrated in Fig. 1. This can be implemented in both graduate and undergraduate programs. Through this *scaffolding*, students better understand the value of learning and applying software, and can reuse what they learned previously to expand their use of the technology for problem solving in other courses.

Moreover, course instructors and teaching assistants can use instructional resources made available by technology providers. When such resources are available, teachers do not have to develop all instructional materials themselves, and students do not have to learn the software tools entirely on their own [16].

3.2 Good Practice 2: Establish a solid foundation for first year students with tools they will reuse

Another good practice is for the faculty members to agree to give all first-year students a solid foundation to build upon by teaching a common *Introduction to Engineering* type course, oriented toward problem-solving and including software tools that will be widely reused in subsequent courses. The success of approaches like this can be found in many universities, for example see [15].

3.3 Good Practice 3: Coordinate tool use throughout an entire department's curriculum

Students learn at different rates and using different approaches. However, repetition is a powerful method of reinforcing concepts. It is a good practice to ensure that at least one course every semester or term is taught using the same software tools for technical computing and/or simulation. Under this approach, students' proficiency with these tools does not atrophy during semesters of no use, but rather grows stronger through reinforcement and scaffolding. As a result, instructors in higher-level courses in the third year and beyond can cover more concepts at a deeper level and can introduce students to Model-Based Design techniques used in industry.

3.4 Good Practice 4: Integrate tools thoroughly throughout curricula for all applicable departments

Thought leadership and high-level support from multiple engineering department heads and the dean of engineering enables the implementation of a coordinated, integrated curriculum across multiple engineering departments, and potentially math and science departments as well.

MATLAB integration throughout the entire engineering curriculum enables a holistic teaching-and-learning approach that attracts students, engages them, and helps them readily transfer skills within their coursework and beyond, into industry [16].

4 CONCLUSIONS

Most students today are comfortable using computers as they begin their studies, but few are comfortable applying them to solve engineering problems. To close this gap, computational thinking and the development of associated skills must be integrated throughout the engineering curriculum. Examples of this integration are cited here and a number of practices for achieving optimal results are included.

While the mathematical foundation for engineering education is relatively stable, the complexity and multidisciplinary nature of engineering problems today requires engineering programs to satisfy multiple learning goals at the same time. The ability to use software tools is not only a requirement for today's engineers, it also a solution that enables students to put the theory they have learned into practical use on real systems. Experience with a variety of tools can help students broaden their horizons, but this benefit must be weighed against the ability to explore engineering concepts in greater depth by using—and gaining experience with—the same tool environment in each year of their studies. Instructors worldwide have found that using the same set of software tools from the first year through graduate studies enhances the learning experience—particularly when those same tools are used by practicing engineers working on real-world problems in industry.

5 ACKNOWLEDGMENTS

The authors wish to thank T. Gaudette and J. Tung for their review of the text.

REFERENCES

[1] Mills, J.E., Treagust, D.F., (2003), Engineering Education – Is problem-based or project-based learning the answer?, *Australian Journal of Engineering Education*, on-line publication, http://www.aaee.com.au/journal/2003/mills_treagust03.pdf.

[2] House of Lords, Select Committee on Science and Technology (2012), Higher Education in Science, technology, Engineering and Mathematics (STEM) subjects, 2nd report of session 2012-2013. http://www.publications.parliament.uk/pa/ld201213/ldselect/ldsctech/37/37.pdf

[3] Wing, J., (2006), Computational Thinking, *Communications of the ACM*, Vol. 49, No.3, pp. 33-35.

[4] National Academy of Engineering, (2005), Educating the Engineer of 2020: Adapting Engineering Education to the New Century, National Academies Press, Washington D.C.

[5] ENAEE Administrative Council, EUR-ACE, (2008), Framework Standards for the Accreditation of Engineering Programmes, http://www.enaee.eu/eur-ace-system/eur-ace-framework-standards

[6] MathWorks, (2012), *Airbus develops fuel management system for the A380 using Model-Based Design*, http://www.mathworks.co.uk/company/user_stories/Airbus-Develops-Fuel-Management-System-for-the-A380-Using-Model-Based-Design.html?by=company

[7] Edgar, T., (2004), *Computing Through the Curriculum: An Integrated Approach for Engineering,* Proceedings of ASEE, Salt Lake City, Utah.

[8] MathWorks, (2012), Mathematic undergraduate students at the University of Oxford use MATLAB for symbolic computation and problem solving, http://www.mathworks.co.uk/company/user_stories/Mathematics-Undergraduate-Students-at-the-University-of-Oxford-Use-MATLAB-for-Symbolic-Computation-and-Problem-Solving.html

[9] Brown, M., Steele, C., (2013), *MathExplorer: Exploring UG Engineering Maths Using MuPAD,* MATLAB Virtual Conference 2013.

[10] MathWorks, (2012), *KTH Royal Institute of Technology's CDIO Program Improves Retention Rates and Turns Students into Engineers*, http://www.mathworks.co.uk/company/user_stories/KTH-Royal-Institute-of-Technologys-CDIO-Program-Improves-Retention-Rates-and-Turns-Students-into-Engineers.html

[11] MathWorks, (2011), *Technische Universität München Uses Model-Based Design to*

Drive Research, Problem-Based Learning, and Industry Collaboration, http://www.mathworks.co.uk/company/user_stories/userstory57690.html

[12] Aach, T., (2013), *MATLAB meets Mindstorms, Institute for imaging and computer vision, RWTH Aachen, Publications web page,* http://mindstorms.lfb.rwth-aachen.de/index.php/en/publications

[13] MathWorks, (2013), *Supported Hardware Resources*, http://www.mathworks.co.uk/academia/hardware-resources/

[14] MathWorks (2013), *File Exchange*, http://www.mathworks.co.uk/matlabcentral/fileexchange

[15] MathWorks, (2011), Northeastern University's High-Tech Tools and Toys Lab Teaches Freshmen to Think Like Engineers, http://www.mathworks.co.uk/company/user_stories/Northeastern-Universitys-High-Tech-Tools-and-Toys-Lab-Teaches-Freshmen-to-Think-Like-Engineers.html

[16] MathWorks, (2011), *Michigan State Integrates MATLAB into Engineering Curricula to Foster Student Proficiency in Problem-Solving Using Computational Tools* http://www.mathworks.co.uk/company/user_stories/Michigan-State-Integrates-MATLAB-into-Engineering-Curricula-to-Foster-Student-Proficiency-in-Problem-Solving-Using-Computational-Tools.html