Removing Lectures from a Computer Programming Course – A Quantitative Study

Christian Thode Larsen[12], Sebastian Gross[3], J. Andreas Bærentzen[1]

[1]Technical University of Denmark (DTU), Denmark, [2]Danish Research Center for Magnetic Resonance (DRCMR), Denmark [3]MathWorks GmbH, Germany

ABSTRACT

Computer programming is a discipline that is becoming increasingly important to today's engineering practice and society overall, and it is used extensively and intensively in several fields. Examples can be found in mechanical, electrical, or chemical engineering, and many other areas. As such, it is common that higher education institutions teach at least one basic programming course on the subject in every engineering degree program.

This means that programming is taught to a variety of engineering students with significantly different backgrounds and expectations. Consequently, it is important that all students — in particular those that had less prior exposure to computational methods and programming in their past — get sufficient time to gain experience with their programming tools.

Lectures are often chosen as the standard teaching method when designing a course structure. This also applies to programming courses. Two main reasons for this are the effectiveness of informing a large group of students in a very short period of time, and the efficiency of doing so while involving only a single teacher.

However, we believe that programming is much better taught and learned with a "hands-on" approach. Therefore, we argue that lectures can be safely removed from programming courses and replaced with extended lab exercise sessions where teaching assistants circulate and interact with the students individually when needed. This gives students additional time to build up experience with the programming environment consisting of the programming language, interface, and associated tools.

We support our argument with a discussion of both quantitative metrics and a summary of qualitative statements drawn from programming course evaluations and student feedback. These evaluations were gathered in courses over a series of semesters — before and while the course followed our suggested model.

We conclude from the observed data that removing lectures from the course not only increased overall student satisfaction, but also bolstered the learning outcome.

Furthermore, we show that an appropriate adaptation of better teaching material, after having removed the lectures, yielded further improvement in these categories.

KEYWORDS

Programming, MATLAB, Lectures, Teaching, Standards: 2, 5, 8, 11.

INTRODUCTION

Jeanette M. Wing coined the phrase *computational thinking* in 2006 and referred to it as a universally applicable attitude and skill set for everyone (Wing, *Computational Thinking*, 2006). She highlights that the process of transformation that started decades ago in engineering and science has long since spread to numerous other disciplines (Wing, *Computational Thinking – What and Why?*, 2010).

Computers, programming, and computational thinking are at the heart of countless technological, economical, and social developments and have become a part of everyday life. Barr and Stephenson clearly state "All of today's students will go on to live a life heavily influenced by computing, and many will work in fields that involve or are influenced by computing." (Barr & Stephenson, 2011). They leave no doubt that computer science education is vital for everyone and here to stay. The design of engineering and science degree programs, as well as neighboring disciplines, must address this growing need in particular.

The EUR-ACE Framework Standards for the Accreditation of Engineering Programs (ENAEE Administrative Council, 2008) list six outcomes of engineering programs. Those include not only knowledge and understanding, but also engineering design, practice, and analysis, investigation, and transferable skills. The learning outcomes defined in the CDIO syllabus 2.0 (Standard 2) (Crawley E. F., Malmquist, Lucas, & Brodeur, 2011) list 'technical knowledge and reasoning' as the first important aspect; however, many others follow. The list includes problem solving, experimentation, knowledge discovery, teamwork, design, implementation, and operation. The CDIO Standard 4 describes experiences gained from design exercises and problem solving. These introductions to the engineering world are meant to be both instructional and motivational in nature. Standard 5 drives this further to engage students in design-build experiences of larger scales. These aspects are meant to be cornerstones of the engineering curricula (Crawley E. F., Malmquist, Östlund, & Brodeur, 2007). The National Academy of Engineering (National Academy of Engineering, 2005) demands the iterative process of designing, predicting performance, building, and testing should be part of the curriculum from the beginning of the degree program. While all these mentioned competencies are fairly general, they are also very important in relation to programming.

At the same time, the changes in education are evident. We are moving from traditional lectures to more interactive communication. Classrooms are flipped and simple knowledge transfer is moved to video sessions (Lage, Platt, & Treglia, 2000). Lab classes go beyond experiments that prove taught theory in experiments to connect acquired knowledge with the real world beyond the wall of our education institutions (Thomsen, et al., 2010) (Thomsen, Scenario Based Learning in Electronic and Electrical Engineering UCL, 2013). Practical classes exposing the students to real tasks instead of unidirectional knowledge transfer are becoming more and more popular (Behrens, Atorf, & Aach, 2010) (Gross, Schlosser, & Schneider, 2014).

The provocative question "What's the use of lectures?" has been raised by a likewise named book in 1971 (Bligh, 1971). After results drawn from an advanced biology class, a work from the year 2005 suggests that even a partial shift toward a more interactive and collaborative class format can lead to significant increases in student learning gains. (Knight & Wood, 2005).

In this paper, we follow these cues and remove traditional lectures from a computer programing course.

The remainder of the paper is organized as follows: The second section describes the course structure and the implemented changes over a period of several years at DTU. The third section explains how the quantitative and qualitative data was acquired, and the fourth section highlights the results. The fifth section discusses the results with respect to target audience, knowledge retention, influence of structural changes to the course, and underlying effects. The sixth section suggests future developments for the course. The final section draws conclusions.

COURSE STRUCTURE AND IMPLEMENTED CHANGES

Today the DTU MATLAB course is run in two different versions each semester. One version takes place during each of the spring and autumn semesters for 13 weeks, and the other is taught for three weeks in January and June. The 13-week versions runs in parallel with other courses, while the three-week versions are concentrated one-course periods.

While the duration of the two versions is different, both versions utilize exactly the same structure, content, and number of sessions. The students work with the curriculum for approximately two-thirds of the sessions, while the remaining sessions are used for project work. The course is followed predominantly by first-year students from many different bachelor degree programs. A non-exhaustive list of programs includes chemistry, biotechnology, mathematics, physics, and innovation and design.

The MATLAB course has run in several stages of evolution for many years, as highlighted in Table 1. Before June 2010, lectures (1-2 hours) and lab exercises alternated. In June 2010 the lectures were removed in favor of longer lab exercises (4 hours) with extended supervision of the students by teaching assistants (3 hours). Until June 2012 the MATLAB course used a fairly inhomogeneous teaching note written by several authors from DTU.

Table 1. Course structure and implemented changes

Time	Structure	Examination (final project)	Teaching Material
'08 – '10	Lectures + lab class + projects	Code evaluation	Inhomogeneous teaching note
'10 – '12	Curriculum lab class + projects	Code evaluation	Inhomogeneous teaching note
'12 – '14	Curriculum lab class + projects	Code + written report evaluation	MATLAB book (Attaway, 2013)

During this period, the noncurriculum sessions consisted of three projects. Two of these were dedicated to learning, where the students would solve a programming task — given a fairly broad problem specification — as a team effort (two to three people per team). The students would receive feedback on the quality of their work by teaching assistants.

The final project constituted the exam, with each student writing his or her own program given a smaller and much more specific problem specification. Grading was based on evaluation of the exam project program code submitted by each student. Some evaluation criteria were program quality, efficiency, accuracy, structure, and commenting.

In June 2012 a number of changes were introduced. First, the teaching note was replaced by the book *MATLAB: A Practical Introduction to Programming and Problem Solving* by Stormy

Attaway (Attaway, 2013) in order to improve teaching material homogeneity. Second, all exercises — previously from the teaching note — were also replaced by those from the book. Third, one of the two team-based learning projects was removed to make room for a slightly extended curriculum due to the new book. Also, evaluation of the students' work was changed from graded feedback to 'on-the-fly' feedback during class.

Finally, the problem description in the exam project was made broader and more open, thereby allowing each student more freedom in his or her design and implementation choices. Also, the students were now required to report and discuss and critically assess their design choices, implementation, and program functionality.

Connection to CDIO Standards

The exercise sessions, as well as learning projects, rely heavily on active learning, which is the goal of CDIO Standard 8. The structure of all projects involves a design and an implementation stage, which is key in CDIO Standard 5. Likewise, many exercises in the curriculum involve concepts from Standard 5.

The course is structured around a list of (publicly available) learning objectives. This list also forms the basis for the exam evaluation criteria, and is used as guideline for the teacher's assessment of how many objectives were met, given the quality of report and code. As such, the course's learning objectives are intimately linked with CDIO Standard 2 and 11.

DATA ACQUISITION

Quantitative and qualitative evaluations of the MATLAB course have been collected for several years. We present assessments from the beginning of 2008 until summer 2014.

The quantitative assessments cover the students' perception of how much the individual student learned, to what extent active participation was required to follow the course, the quality of the teaching material, how well the teacher provided feedback on submitted work, and finally the overall quality of the course.

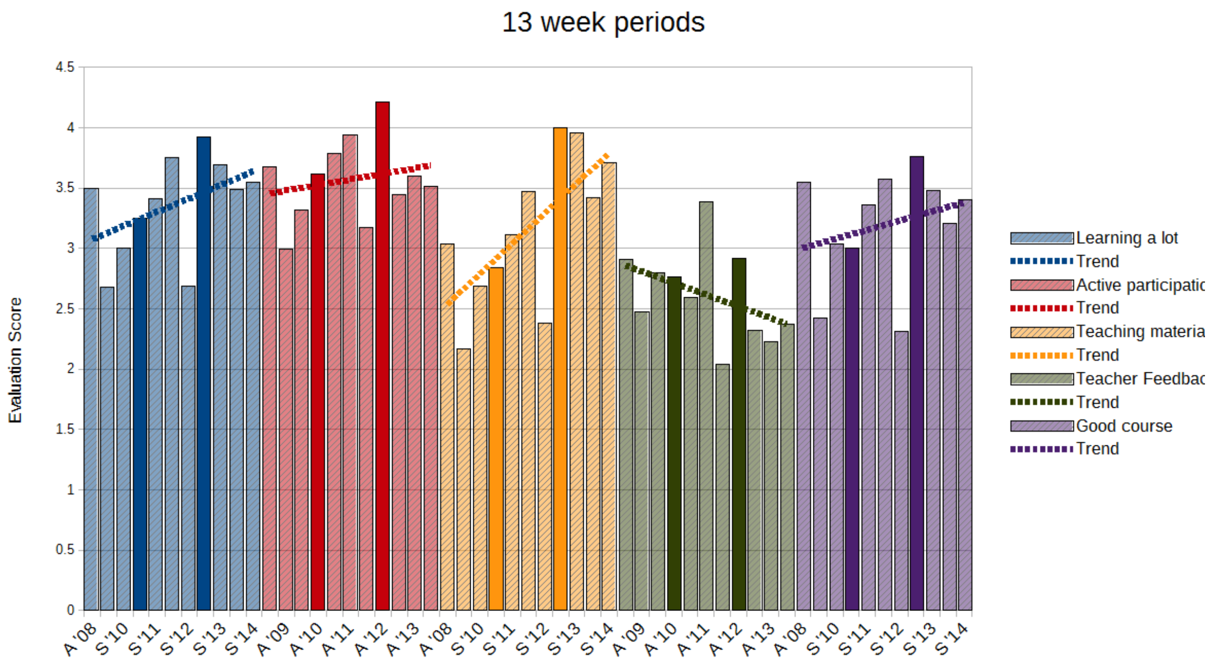
All quantitative assessments operate on a scale from 1 to 5, where 1 is the lowest possible score and 5 is the highest. All quantitative data is available online[1]. The qualitative data is confidential, for which reason we provide only a summary.

RESULTS

Quantitative evaluations

The average quantitative assessments for all periods where the MATLAB course ran between 2008 and 2014 have been presented in Figure 1. Each category has been depicted with its own hatching, transparent color. Also shown are the periods where the removal of lectures and the introduction of the MATLAB book took place (solid colors), as well as general trends in the different categories (dotted lines).

[1] http://www.kurser.dtu.dk/



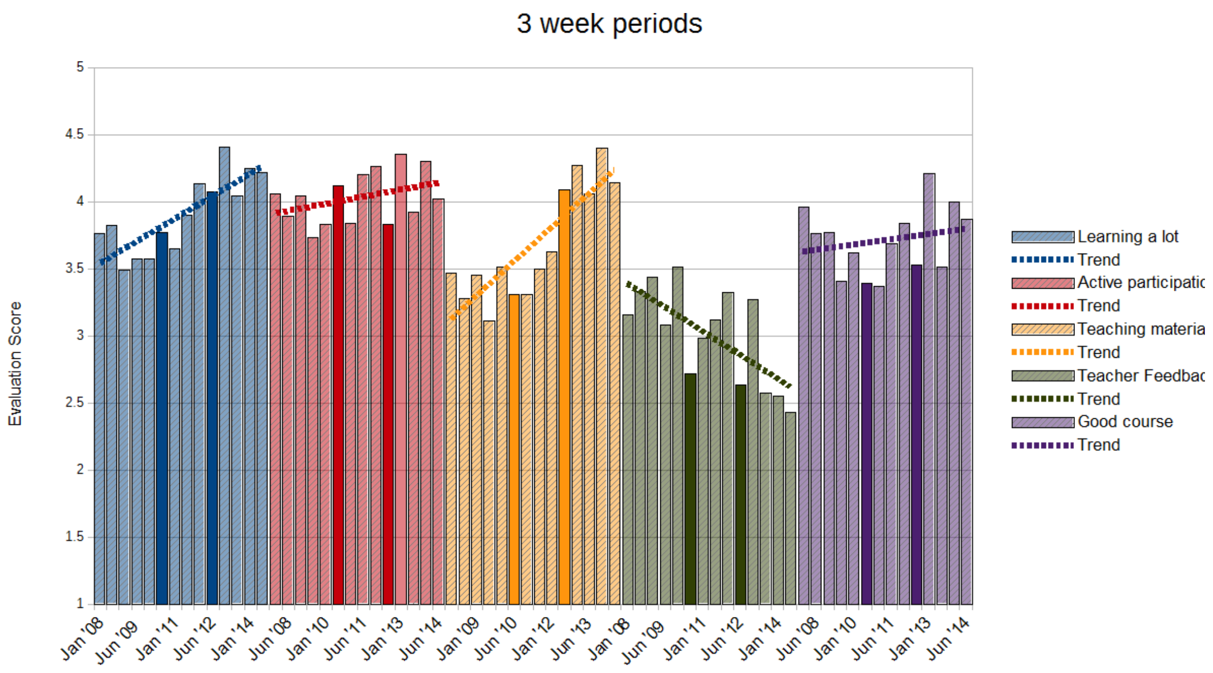


Figure 1. Evaluation scores from 3 week and 13 week periods. Each category has been illustrated with its own hatching, semi-transparent color. The time points where lectures where removed and the MATLAB book introduced, has a solid color. The dotted lines signifies the overall positive or negative trend in course development within each category.

When comparing the trend lines to the categories, there seems to be a tendency towards a climb in evaluations after the lectures were removed, except for the *teacher feedback* category. This suggests that the students reacted positively to this change.

In particular the category of *teaching material* stands out, revealing that replacement of the inhomogeneous teaching note (by the MATLAB book) was received very positively by the students. This suggests that good teaching material is very important to the students, perhaps even more so than the removal of lectures.

Also, interestingly, inspection of the *learning a lot* and *teaching material* categories, and to some extent the *active participation* category, suggest there is a link between how much the students feel they learned, how active they feel they have been in the course, and the material they have been studying.

Two periods stand out with significantly poor student evaluations: the spring of 2012 and the autumn of 2013. Possible explanations for this are that, in one case, the teacher (T1) may have lacked motivation due to ending employment. In the other case, the teacher (T4) who had no prior experience with the course went on a sudden leave of absence. This resulted in other people, also with no prior experience, having to take on the teaching responsibility without proper preparation.

Finally, it can be seen that the three-week periods consistently receive higher average scores when compared to their 13-week counterparts.

Table 2. Distribution of teachers, the number of evaluations received, and the number of participants during the different periods.

	Jan '08	Jun '08	Jan '09	Jun '09	Jan '10	Jun '10	Jan '11	Jun '11	Jan '12	Jun '12	Jan '13	Jun '13	Jan '14	Jun '14
Teacher	TX	TX	TX	T1	T1	T1	T1	T2-3	T2	T2	T2	T3	T3	T3
Evaluations	50	98	84	123	81	136	49	131	38	125	44	53	20	95
Participants	94	202	233	238	139	255	140	317	99	310	148	283	151	355

	A '08	A '09	S '10	A '10	S '11	A '11	S '12	A '12	S '13	A '13	S '13
Teacher	TX	T1	T1	T1	T1	T1	T1	T2	T3	T4	T3
Evaluations	118	104	26	79	56	53	52	52	52	63	43
Participants	202	156	51	166	113	149	104	161	152	151	150

As depicted in Table 2, a number of different teachers have been responsible for the MATLAB course. TX represents various teachers before T1 took over and introduced a more computer-scientific approach in the inhomogeneous teaching note. T2 introduced the removal of lectures as well as the new teaching material. T3 collaborated with T2 in order to take over the new structure. T4 was a substitute teacher during a single period in the autumn 2013.

Also shown in Table 2 are the number of participants and received evaluations during each period. In general, between 25% and 50% of the students evaluate the course. Note that the June 2013 and January 2014 courses suffer from fewer evaluations where, in particular, January 2014 is statistically weak with only 20/151 evaluations.

Summary of qualitative evaluations

Before the lectures were removed, many students would express displeasure with their presence, while few expressed satisfaction. Afterwards, a significant number of students reported appreciation for the increase in hands-on programming time, while few suggested that lectures would have been nice. A number of students felt that they needed to work harder to get through the course, but also that they learned a lot. This trend continued after introducing the MATLAB book and the new exam project. Only a few students requested more feedback.

DISCUSSION

The success or failure of any course results from a mix of many different factors. Now we discuss some of the factors we believe to be most important for our observed data, as well as for running a successful programming course.

Removal of lectures

It is perhaps not obvious how removing lectures can contribute to an increased appreciation of the course. Our hypothesis is that the removal of lectures made room for more hands-on programming, and this has led to greater learning — thus, greater course appreciation.

Influence of the teacher

Not surprisingly, the teacher and teaching assistants are key to running a successful course. Specifically, their experience in teaching and their ability to motivate and 'meet the student where he or she is', are important. Upon careful comparison between Figure 1 and Table 2, it becomes clear that the evaluations depend heavily on the actual teacher in charge of the course. As such, using and critically assessing student evaluations is instrumental in choosing the right teacher and teaching assistants for a course.

Restructuring a course

Whenever the structure of a course is modified, it is likely to run into unforeseen issues with teaching material, exercises, and projects. Typically, the teacher will not be able to completely address these until the second iteration of the course is run.

One example was the introduction of the new MATLAB book. With the book followed a whole new range of exercises, but they turned out to be too many and they were too extensive. For this reason, the students became discouraged. As a result, the exercises were trimmed and prioritized to help the students move along the learning curve more easily.

This example serves to illustrate that there is a tradeoff between stability and changing the course structure too often. Minor changes should be implemented whenever necessary in order to obtain obvious improvements. Major changes such as changing the course structure should be done rarely and only if there is an apparent need to do so. Furthermore, every new structure should be allowed time for refinement and tuning.

Targeting the audience properly

There are large differences between the students from different degree programs as a group, as well as between the individual backgrounds of students. Therefore, students from certain degree programs will be more inclined to respond positively to a course structure without lectures, which means that it leaves them with more responsibility for learning.

It is difficult to draw a clear trend from the data that supports this hypothesis since students mix between the different periods. In general, we have observed that students from technical degree programs such as electrical, mechanical or mathematical engineering respond more favorably when compared to interdisciplinary degree programs such as biotechnology or design and innovation.

Ideally, the course content should target each study line specifically, with more specialized examples and a refined course curriculum. This is, however, a resource demanding task to perform, which is obviously why it is rarely done.

Retention

Figure 1 revealed that the 3-week period received significantly better evaluations compared to the 13-week period. This may be due to short-term retention. Some students express that it is difficult to learn and remember a given topic from one week to the next. Far fewer difficulties have been observed during the 3-week periods. We believe that an intense period of focused learning with a lot of hands-on time to be the best way to ensure short-term retention.

This does not address the issue of long-term retention. Some students lack motivation 'because they are not going to do programming in the future'. Other students find that they do need programming later on but forgot what they learned during the course.

Ultimately, programming is a craftsmanship that needs to be learned and maintained. There seems to be no easy solution to this other than to have the students program regularly in as many courses as possible throughout their studies. However, this model is reasonable for only a subset of degree programs.

Declining feedback score

The removal of one of the team-based learning projects, and changing evaluation of the second one to an 'on-the-fly' process, is a possible explanation for the decline in feedback score. While the students have more confrontation time with the teaching assistants without lectures, they may not consider this to be feedback to the same extent as an evaluated (possibly graded) project. Another reason could be that the teacher is less visible.

Interestingly, we did not observe that the students felt, in general, that they needed more feedback. A possible explanation is that the feeling of having learned a lot outweighs the need for feedback. Even so, it follows that a better model for providing feedback could be employed. Possible ways to improve this could be, for example, regularly scheduled reviews (depending on session type) of the implemented exercises.

FUTURE WORK

In the future, we hope to address the one negative trend: teacher feedback. To some extent, feedback can be improved by using automatic code assessment software, but there is also a need for more direct communication between teacher and students. The challenge here is that the number of students is quite large. However, some time is freed up by not having to give lectures. Teaching assistants provide feedback. Finally, we are also experimenting with Internet fora as a means for communicating with students.

While we are very satisfied that the described changes led to an improvement in learning, the course is being fundamentally changed yet again — this time not to improve quality, but because the course is being used in several educations, all needing slightly different learning outcomes. For this reason we are investigating modularization of the course in order to make it more adaptable, such that all of our client engineering programs can be accommodated.

CONCLUSION

We have presented a number of changes that were introduced in a computer programming course over a number of years. Specifically, we have discussed how lectures were removed, how the teaching material was later replaced, and how the course structure was modified. Quantitative and qualitative data have been provided, which documents how the students evaluated the course before, during, and after these changes.

Unfortunately, the evaluations are clearly affected by many factors that are outside of this study and outside of our control: who the teacher is and the body of students following the course are the most obvious. The evaluations are also influenced by whether the course is taught during the 3-week or the 13-week period. Despite these shortcomings in the measurements, we believe that the data shows two clear trends. First, the students feel a bit more left alone: there is a decrease in how much feedback they feel they receive. On the other hand, there is a positive trend when it comes to all other statistics pertaining to how much the course is appreciated. This indicates that the changes are an improvement, and while the course is not in a finished form (and never will be), we regard the described changes as a positive step in the continuing development of the course.

REFERENCES

Attaway, S. (2013). *MATLAB: A Practical Introduction to Programming and Problem Solving* (3rd ed.). Butterworth-Heinemann.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? (ACM, Ed.) *ACM Inroads, 2*(1), 48-54.

Behrens, A., Atorf, L., & Aach, T. (2010). Teaching Practical Engineering for Freshman Students using the RWTH - Mindstorms NXT Toolbox for MATLAB. In E. P. Leite, *MATLAB - Modelling, Programming and Simulations* (pp. 41-65). SCIYO.

Bligh, D. A. (1971). *What's the use of lectures?* Hawfordshire, England: New Barnet.

Crawley, E. F., Malmquist, J., Lucas, W. A., & Brodeur, D. R. (2011). The CDIO Syllabus v2.0 - An Updated Statement of Goals for Engineering Education. *Proceedings of the 7th International CDIO Conference.* Copenhagen.

Crawley, E. F., Malmquist, J., Östlund, S., & Brodeur, D. R. (2007). *Rethinking Engineering Education: The CDIO Approach.* New York: Springer Verlag.

ENAEE Administrative Council. (2008). *Framework Standards for the Accreditation of Engineering Programmes.* EUR-ACE.

Gross, S., Schlosser, J., & Schneider, D. (2014). Integrating Introduction to Engineering Lectures with a Robotics Lab. *Proceedings of the 10 International CDIO Conference.* Barcelona, Spain.

Knight, J. K., & Wood, W. B. (2005). Teaching More by Lecturing Less. (E. Chudler, Ed.) *Cell Biology Education, 4*(4), 298-310. doi:10.1187/05-06-0082

Lage, M. J., Platt, G. J., & Treglia, M. (2000). Inverting the Classroom: A Gateway to Creating an Inclusive Learning Environment. *Journal of Economic Education*, 30-43.

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

Thomsen, B. (2013, 3). *Scenario Based Learning in Electronic and Electrical Engineering UCL.* (T. a. Workshop, Ed.) Retrieved 11 11, 2013, from http://www.youtube.com/watch?v=sh6x8h-3eEE

Thomsen, B., Renaud, C., Savory, S., Romans, E., Mitrofanov, O., Rio, M. Mitchell, J. (2010). Introducing Scenario Based Learning - Experiences from an Undegraduate Electronic and Electrical Engineering course. *Education Engineering (EDUCON)* (pp. 953 – 958). Madrid, Spain: IEEE.

Wing, J. M. (2006). Computational Thinking. (ACM, Ed.) *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2010). *Computational Thinking - What and Why?* Retrieved 10 22, 2013, from http://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

BIOGRAPHICAL INFORMATION

Christian Thode Larsen, M.Sc., has a bachelor’s degree in software technology and a master’s degree in digital media engineering from DTU. He has been involved with the MATLAB programming course for a number of years, first as a teaching assistant and later as the teacher. He is currently pursuing a Ph.D. degree in medical image analysis and machine learning at DTU Compute and the Danish Research Center for Magnetic Resonance.

Sebastian Gross, Ph.D., studied at RWTH Aachen University and received his electrical engineering degree in 2007 and his Ph.D. in image processing in 2014. He is now working as an education technical specialist at MathWorks. His main function is the support of teaching staff in tool application, curriculum design, and the general improvement of technical teaching at all levels of education.

J. Andreas Bærentzen, Ph.D., is an associate professor at DTU Compute. His research focuses on the representation of digital shape and, more generally, on computer graphics. His teaching is mostly on geometry processing and real-time graphics. He has been director of studies for the Digitial Media Engineering M.Sc.Eng. programme at the Technical University of Denmark. He currently serves as chairman of the study board at DTU Compute. He has supervised three Ph.D. projects, 22 MSc projects, and 16 BSc/BEng projects.

Corresponding author

Christian Thode Larsen
DTU Compute
Richard Pedersens Plads, Building 324
DK-2800 Kongens Lyngby
+45 28 58 07 87
christian.thode.larsen@gmail.com