TURNING PROGRAMMING INTO A RELEVANT TOPIC FOR NON-PROGRAMMING ENGINEERS

Erik Berglund, Dennis Persson

Department of Computer and Information Science, Linköping University, Sweden

ABSTRACT

In this paper we present an introductory course on programming for about 190 mechanical engineering, design, and product-development engineering students. These students use 3D-modeling software to develop physical products. Programming is one of the tools in their toolbox, and writing algorithms can both improve the efficiency of their work and transform their work process.

At the heart of the course, in line with CDIO Standard4, is a focus on real-world applications in an introductory programming course. Understanding why and how programming is a useful tool is considered to be of equal importance to learning fundamental programming concepts.

Here we present and discuss the course and how we plan to change it in the future. We report the results of student evaluations and our own experiences. Our results, thus far, show that the applied approach has been instrumental in turning programming into a relevant topic for these non-programming engineering students. Currently, however, there is also a relatively long period of frustration and students experience an inability to use documentation and online resources. Moving forward, we plan to add a crash course with a traditional focus to the first week of the class, before starting on the applied work. It is our belief that this will make students feel more secure, and as a result allow them to be more self-sufficient in overcoming the practical challenges they face in the course.

KEYWORDS

Introduction to programming, CDIO Standard 4, Blender 3D, Python.

INTRODUCTION

Finding ways to connect the real world for to introductory courses and to allow students to use fundamental concepts to build “real” things is central to the CDIO standards. The standards highlight the applicability of knowledge, even at an introductory level. For non-programming engineering students, programming can often be seen as theoretical, abstract and complicated. This can be discouraging, and the real-world applicability of basic programming concepts can be hard for students to see.

When tasked with constructing a new introductory programming course for 190 mechanical engineering and design and product development engineering students, this was an obvious challenge. These students are not computer-science engineers and thus are not motivated by the subject matter, and few have prior knowledge of programming. A show of hands during the introductory lecture in 2017 showed that less than 10% had done programming exercises before.

Our goal was, and continues to be, to construct a course that would teach the basics of programming while still being directly relevant to the students' future career. We wanted to find ways where each student could complete a project on their own at the end of the course that, in their mind, would represent a real task, while still maintaining the basic and abstract building blocks of programming at the center of the activity. Also, for many students, using code in 3D modeling and simulation software for the construction of physical products constitutes a new perspective on the work process. Thus, one goal of the course is to introduce how code can enable creativity in an engineer's work.

CDIO Standard 4 in particular is relevant to the design of this course. For example, Standard 4 emphasizes that introductory classes should "strengthen [students'] motivation for the field of engineering by focusing on the application of relevant core engineering disciplines". (CDIO Standards 2018). Using Blender 3D (www.blender.org), which is free and open source software for 3D modeling and simulation, we address programming in the domain of constructing physical products and using code as one of the tools available to achieve such goals. Students use Blender 3D in subsequent courses, and it is well-documented by a large community of YouTubers. The scripting language for Blender 3D is Python (www.python.org), which is one of the world's most used programming languages and also a common first language at universities.

The course also address aspects of Standards 5 and 7 in ways that are natural to the course. Standard 5 states that courses should emphasize "…engineering activities central to the process of developing new products and systems." And Standard 7 states "… students might consider the analysis of a product, the design of the product, and the social responsibility of the designer of the product, all in one exercise." (CDIO Standards 2018). For these students, programming is a new tool that enables efficient workflows, but it also changes the potential work process from solving engineering challenges to using code to control the computer, and allowing the computer to solve certain engineering challenges for them. There are also more sophisticated applications which lie beyond the scope of this course, such as the use of machine learning and artificial intelligence for generative algorithms that search for optimal physical constructions. Standard 7 also addresses the issue of engineering and its impact on the world. This course has a natural connection here, where one logical application of generative design is minimizing the use of materials while maintaining constructional requirements.

In this paper we discuss our course on the fundamentals of Python programming, in which we teach our non-programming engineering students how to use Python, in conjunction with the 3D modeling system Blender 3D, for the construction of physical products. We present an evaluation of the students' perceived attitudes and discuss how to change the course in the future to improve this applied introductory course. A main component of this plan is the addition of a crash course on more traditional discipline training right at the beginning, to ensure that the students have enough basic knowledge to become more self-sufficient in using online resources and search results that are not adapted to a more applied approach to learning to program.

THE ROLE OF PROGRAMMING IN PHYSICAL PRODUCT CONSTRUCTION

The construction of physical products using 3D modeling and simulation software, like Blender 3D, includes programming as one tool in the toolbox. Much like manipulating the model using a mouse and keyboard, engineers can write code that can create, manipulate, and analyze models, and moreover can automatically repeat such a process until a sufficient

solution has been reached. A significant new aspect of product design is algorithmic or generative design; see for instance Krish (2011). Currently, machine learning and artificial intelligence in relation to the construction of physical products is also very relevant.

Creating and evaluating parameterized 3D models, manipulating, deleting and minimizing materials, and running physical simulations – all of these things involve designers using code to search for physical constructions. This changes not only the efficiency of development but also has an impact on the creative process.

At the same time, a process that is a hybrid of hand-made and code-driven design may ultimately be more time-efficient than a fully-automated process. Real applications may include very few lines of code and yet may still be representative of real use-cases in industry.

RELATED WORK

Other work that relates to ours includes efforts to change standard approaches to how introductory programming courses are taught, and work on increasing integration of elements of the CDIO Standards and pedagogical elements like constructive alignment to achieve more or deeper learning outcomes, or to encourage more efficient teaching methods for better learning outcomes. A general approach to learning to program is to place a substantial amount of emphasis on practical work, "… on practice, practice and practice" as reported by Winslow (1996).

Prost (2016) reports positive motivational effects from adding degrees of freedom that allow students to make choices about parts of their tasks, but also notes that this challenges the teachers, making it harder and more time-consuming to prepare for this openness. Phae et al. (2014) and Martínex and Muñoz (2014) reported positive findings from organizing introductory programming classes into larger teams of up to 5 people, saying that it was both good for learning and more motivational. Here the assessment process was also changed in order to support students teaching each other. More social and reflective assessments were required. An alternative position was presented by Gaspar and Langevin (2012), in which traditional pair programming was replaced by a process of initial individual preparation followed by pair programming. Also, utilizing automated test systems, Gaspar and Langevin used an exchange of tests among student pairs as a means of getting students to generalize their solutions beyond "what works". Reng and Kofoed (2012) also reported on how inspirational events, field trips, and tasks related to image processing have been instrumental in changing the degree of motivation and quality of work produced by non-programming artists and creative professionals that need to have more programming skill for their future careers in the interactive media and games industry. Vo et al. (2017) identify issues associated with using applied and practical work in courses, including that (a) students and teachers may confuse a working system as equivalent with knowledge, (b) a good technical solution doesn't necessarily represent a real learning outcome and (c) there is a risk that failure in the task could be perceived as failure in learning.

Lots of practical work would seem to be a central feature of introductory classes, but it is also important to find good ways of encouraging students to become more oriented towards deep learning. We have initially worked very hard to find the balance between applied challenges and keeping the focus on programming fundamentals. In the future, we want to change activities and assessment methods to avoid shallow learning, and to address the emotional experience of being introduced to programming.

PEDAGOGICAL APPROACH AND COURSE DESIGN

Constructive alignment means finding alignment between learning goals, learning activities and assessment. As a consequence, a student-focused process is needed, in which teachers are part of the supporting environment (Biggs & Tang, 2007). Practical work has been identified as the key activity in enabling novices to become competent in programming (Winslow, 1996). Furthermore, Gagné and Deci (2005) argue that autonomy is an important motivational factor in the learning process.

The goals of our course are to:

- Understand how programming can be used in the production of physical products as one of the available tools in 3D modelling software
- Understand basic concepts in programming like variables, lists, loops, conditions and functions

Additional goals, in the current embodiment of the course, include:

- Working partly by hand and partly in code to see how code, as a tool, compares to other tools
- Using random generation and physical simulation as tools for construction to experience the potential of algorithms for solving problems or generating candidates for a creative process
- Visualizing the algorithmic process so that students can see what the code can do
- Running large numbers of experiments on models, and then validating and sorting the models to see the scope of the algorithmic potential

To achieve balance between autonomy and a tutoring-based style of education, the course is based on the students working on a series of construction challenges in a supporting environment. This includes:

- Labs, with detailed step-by-step instructions about both how to solve the challenge and how to write the code. The labs vary in complexity, but the final lab is as complex as the project. These are like interactive lectures.
- Tasks, with detailed instructions about how to solve the challenge, but not how to code the individual steps. Knowledge about coding, learned in the labs, is applied and repeated here.
- A project, where the students are given only a high-level challenge and must both break down the challenge and code it.

Students work in pairs and are primarily assessed by demonstrating their solution in person. Examination is oriented around being able to explain how the solution works, and grading is based on approving solutions that are adequate in principle. Formative feedback is also provided on how to improve for the next challenge. The project is presented at a closing seminar with a slide-show presentation. Essentially, if a student can demonstrate sufficient ability for the last lab, the tasks, or the project, then they have learned the basics of programming. To address the goal of understanding when and why to write code, we tailor the challenges to illustrate how code can help students achieve things they cannot practically do by hand.

We aim to provide alignment between real-world applications and programming fundamentals by generating, evaluating and comparing large numbers of models. This leads to the need to run loops, evaluate conditions, sort and check data variables, and to keep track of candidates over several steps in the process.

A supportive environment is provided, which consists of lectures, flipped-classroom lectures, tailored material, online support forums, screencast videos, and links to online material. Multiple weekly sessions with assistants in computer labs are also provided, which is very common in our courses at the department. This is currently the most important but also the most troublesome supporting activity. Students are allowed to work on their own time and simply demonstrate their abilities if they are able.

Project 2017: searching for legs

In 2017, the project in the course was to focus on finding leg positions for an asymmetrical tabletop that the students themselves create following a screencast video of a manual process in Blender 3D. Valid leg positions (3 or 4) are positions such that the table doesn't fall over even with weights placed at strategic positions. Figure 1 shows screenshots of such a project.

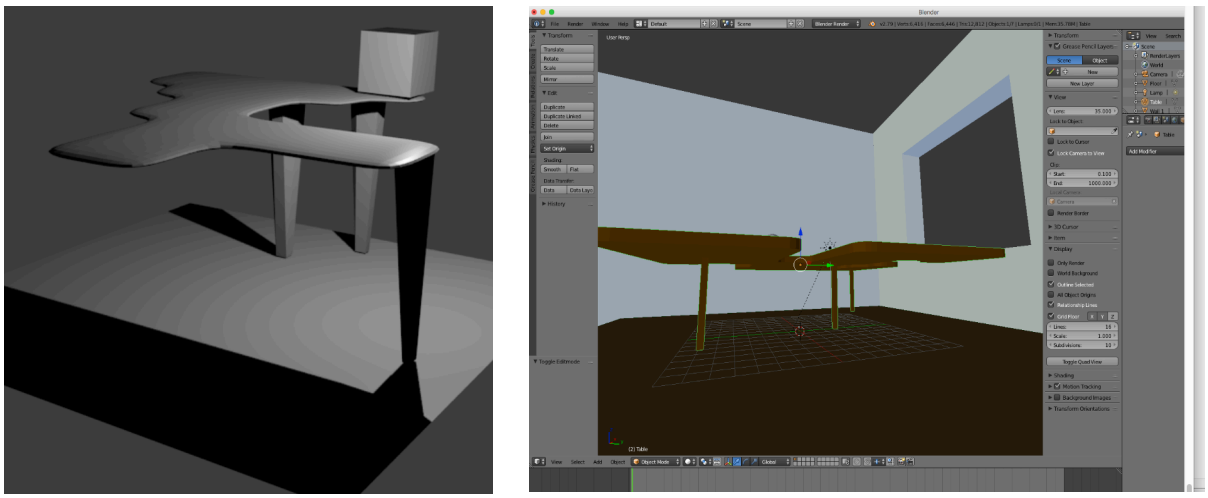

Figure 1. The legs of the tables are generated and validated by code in the search for stable leg-positions for a man-made asymmetrical table. Physical simulation and validation of hundreds of possible candidates are often needed to achieve 10 working tables.

The goal is to identify possible portions of the model for 3 or 4 legs:

- Locate and randomly select possible leg positions on the tabletop
- Add legs and simulate to see if the table is stable
- Simulate with strategic weights and test for stability
- Iterate and find 10 working leg-combinations, often generating more than 100 tables to find solutions that work
- Compare tables that are stable in terms of their leg positions and sort them for a human decision-maker (e.g., eliminate very similar solutions)

The students work partly by hand, to create the asymmetric tabletop and to find positions for weights that fit the table. They then write code to run the process of generating, evaluating and comparing tables that use random selections of legs. The process requires a minimum of about 160 lines of code, so this assignment is not overwhelming.

Using Blender 3D and Python as a platform

In our opinion, Blender 3D is a very good platform for introductory programming in general, but it is particularly good for this group. Blender 3D is a massive system, but the subset of data and functions we needed is very limited, and we find that the code written in our challenges is very much focused on the fundamentals of Python programming, rather than specific Blender concepts. Also, there is a strong conception of general algebraic and 3D concepts like vectors, locations, dimensions, and normals, as well as to physics and physical simulation with forces and torques, which our engineering students learn about in other courses. Other programming concepts such as algorithms, data variables, and flow of execution are also clearly represented.

One of the really interesting possibilities with Blender is that we can visualize an algorithm as it generates and manipulates the 3D scene to show how the algorithm works. The developer has fine-grained control over how the scene updates, and Python can easily be paused if the process runs too fast to be visible.

Blender also has a built-in Python editor, and code can be run directly inside Blender with the click of a button, making it easy to use without setting up the coding environment. One problem is that the editor doesn't autosave, so students may lose code if they are not careful.

We experienced memory-related problems, leading to Blender crashing, that were caused by the students but were too hard for them to understand. Data needs to be deleted in a particular way, and storing data has already been deleted can lead to severe problems. Because of how the Blender data model works, there can be an invisible build-up of memory usage, causing execution to become increasingly slower over time. This can be avoided in the future by changing our challenges so that deletion is not part of the process, and an initial clean-up can be copied-and-pasted into the students' programs at the start of execution.

Course Evaluation

In 2017 we evaluated the course using a questionnaire, and the students also evaluated the course in class workshops (5 classes). Our interest was in their attitudes towards programming in general and towards programming as a tool for them specifically. We are well aware of their actual abilities based on their work, and this is also presented here as an Examiner's reflection.

Student enquiry and evaluation

Out of the 190 students that took the course, 120 students participated in a questionnaire with 19 questions about their attitudes towards programming, their ability to code before and after the course, and their attitudes regarding the subject matter in relation to their future careers. The questionnaire didn't ask about sex or age, but for this student group (190 students of which 120 answered), about 30% were women, and the age range was 19-23. Questions where formulated to evaluate the student's opinions about their abilities before

and after the course and their attitudes towards programming as a subject and their opinion about its relevance for their future career. Questions were of the type,

> "Did you think programming was hard before the course",
>
> "Is programming relevant in your future career?" and
>
> "Would you be able to correct simple errors on your own?"

Students were asked to answer each question with one of the following:

> (a) Yes, agree strongly (b) Yes, to some extent (c) No, not at all (d) Don't know

On average, the data showed that they had little previous experience and thought programming was hard before the course. They learned a lot and felt they would be able to write simple programs and fix simple errors, but were less confident about their ability to write error-free code at the level required by the project completely on their own. They still felt that programming was hard, but they thought it was relevant to them and wanted to learn more.

From the students' own workshops, the students expressed a general appreciation for the subject matter, and for the applied, open and adaptive nature of the course. They felt that the course was very relevant for the program, and that the level of difficulty was reasonable. They complained about not getting enough direct assistance, needing more concrete lecturing and material, and feeling lost and uncertain about requirements. They also raised the problem of not knowing enough about how to use documentation, online resources and search results. They stated that it was hard to appreciate lectures at the beginning of the course due to a lack of basic understanding of programming fundamentals.

The Examiner's reflection

During the two years in which we've run this applied introductory course, the course has successfully provided relevant learning outcomes with both of our goals in mind. In our opinion, the students learn about the same amount as other student groups that take non-applied courses. In 2017 (unlike 2016) our challenges also clearly demonstrate the ability to use algorithms to transform work processes. Students have learned basic programming for real tasks, writing small programs on their own and independently fixing normal introductory code errors.

Also in 2017, we feel that we have achieved better alignment between applied real-world tasks and maintaining focus on fundamental programming models. The generation of multiple models that are simulated physically, evaluated for validity, and ultimately compared with one another with regards to physical criteria, leads to code that is full of fundamental loops, conditions and variable management, and even sorting and development of non-trivial sorting functions. In fact, by allowing the students to manually manufacture the components we can spend more time focusing on programming fundamentals. In 2016 we worked more with managing the camera, adding and changing materials, printing images, and so forth, but this led to code with a large number of Blender-specific function calls stacked on top of one another, and emphasis on algorithmic work was sacrificed.

By adding more challenges related to finding a working candidate with minimal material, we both increase the focus on algorithms and address environmental aspects in engineering, as stressed in CDI Standard 7 (CDIO Standards 2018).

The students complain about needing more concrete and direct help, about not being able to work on their own, and about feeling frustrated and insecure. This has led to increased pressure on assistants and long waiting times in the computer labs, which in itself creates more frustration. While they receive a great deal of support, more so than many computer-science engineers, the students have an insatiable appetite for direct help, which may indicate that we have another problem. Helping them with their challenges in computer labs, as we do at our department in most of our courses, does not seem reduce this appetite. It seems that we have a trial-and-error situation in which students tend to "shake the box" until it works, as Gaspar and Langevin (2012) put it. Perhaps we achieve our learning goals because the students shake the box the same way many times. But the experience that programming is frustrating and hard is a real problem.

For 2017 we created a much larger amount of specially tailored documentation and online material on the course home page, including screencast videos. This did not really improve the situation. In fact, I suspect that the problem cannot be fixed by providing more documentation, more precise and concrete instructions, or even more screencasts (though I have more hope for screencasts). I think this needs to be managed with less direct help and more social, reflective, deep-learning activities on programming in Blender 3D, rather than focusing exclusively on solving suitable challenges.

One student team expressed that:

> Since we were often late to the computer labs we would frequently not get a seat, and this lack of help was the reason that we understood so much at the end of the course.

A few students expressed that they were able to complete the project in a day, because they had really acquired the necessary skills. This is our desired result, but for many others it took much more time and was still frustrating on a fundamental programming level. It's clear that many students had not achieved the level of understanding that we would like them to have by the time they started the project.

Ultimately the students came out with about the same knowledge in 2017 as in 2016, at the intended level of the course. In my personal opinion (having taught programming courses since 1997) they achieved about the same level as many other students. But the students are frustrated, require a lot of direct support, and have a troublesome journey, which reduces their opinion of programming as an enabling technology. There is also cause for concern that this could negatively impact their future efforts and achievements, as Lishinsky et al. (2017) show that negative associations with performance in the past can lead to negative performance and emotions in the future.

Also, one thing that was optimal in 2017 was doing serial physical simulation, see for instance the project description in section 0. This led to the need to delete objects (to clean up before the next iteration) and results being removed and only stored as data. Nothing was visible to students if they didn't actively code for that visibility. In 2018, we should work with parallel simulation and avoid deleting objects during the script execution, which means that the students will come out at the other end with a sea of models that have all been simulated, and with the ability to see all of it over and over again, even to publish it as a video. This is an easily fixed problem that will make a big difference for the course, and for these non-programming engineering students, by giving them a better sense of the value of code.

In terms of working partly by hand and partly in code, we still find that students tend to use more code than necessary, and that they get focused on solving problems only using algorithms. Several challenges in 2017 were more easily solved by using more modelling, and the challenges also illustrated the use of components to support the programming and to create conditions. We believe that changing the structure of some of the learning activities and making them more reflective will help the students to see the whole problem and not just the programming aspects. Here, we also see potential to increase the autonomy by directing students towards creating more steps manually and then running code to address the challenges.

DISCUSSION AND FUTURE COURSE DESIGN

For the future, we want to introduce a crash-course in the first week of the course to get everyone up and running very, very quickly. Not because the students don't learn what they need to learn, but because the students feel frustrated for too long. That feeling needs to be avoided by actually learning a large volume of information quickly in the beginning, which is in line with the findings of Lishinsky et al. (2017) that negative experiences in programming have a negative impact on students' future learning.

We plan to do this in seminars with relatively small groups, with scheduled activities that require active participation, in which the purpose is to experiment, discuss and understand. Also, we are inspired by the findings of Phae et al. (2014) and Martínez and Muñoz (2014) that teams of up to 5 could provide better learning outcomes than individual and pair work, so we want to try larger student groups.

A potential outcome of a traditionally-focused crash course is being able to give the real-world challenges with less specific instructions and more open-ended requirements. Currently we have highly specific instructions with many details that students can misinterpret.

The second thing we want to do is introduce more time between asking for help and receiving help. Deep-learning-oriented help takes time to create, and in the computer lab there is too much pressure on assistants to help quickly and move on. If assistants have more time to answer, the quality should go up and students should get more value out of the help. This, we believe, is more easily done via an online forum than in personal meetings in computer labs. Personal meetings are still needed, but we are planning for flipped-classroom sessions where the students' problems are discussed rather than offering them direct assistance during programming.

To keep our applied approach, we can also develop challenges that include more hybrid development with more hand-made parts. This means that we can have real tasks and more autonomy, without affecting the amount of code required, and even find challenges where very, very small amounts of code still constitute a real application of programming for the student group.

CONCLUSION

We have applied a very real and practical approach to an introductory course in programming to a group of non-programming engineering students, as is stated as a goal in CDIO Standard 4, and which is related to many other CDIO standards such as 5, 6 and 7. Our experiences are very positive, and so is our student group, but learning programming

fundamentals comes late in the course, which has led to feelings of frustration and an inability to use documentation, online resources and search results.

As a result, we want to shift the focus from application to discipline in an early segment of the course, with deep-learning activities such as experimentation and discussion in seminar groups, before turning to more applied programming exercises. Though we find that the applied approach works very well to motivate and teach programming to non-programming engineers, it should be supplemented by an initial, quick infusion of traditional teaching to avoid a prolonged sense of frustration.

REFERENCES

Biggs, J. & Tang, C. (2007) *Teaching for Quality Learning at University Maidenhead*: Open University Press/McGraw Hill.

CDIO Standards (2018, Jan), *The CDIO Standards 2.0*, http://www.cdio.org/implementing-cdio/standards/12-cdio-standards

Gagné, M., & Deci, E. L. (2005). Self-determination theory and work motivation. *Journal of Organizational behavior*, *26*(4), 331-362.

Gaspar, A., & Langevin, S. (2012). An experience report on improving constructive alignment in an introduction to programming. *Journal of Computing Sciences in Colleges*, *28*(2), 132-140.

Krish, S. (2011). A practical generative design method. *Computer-Aided Design*, 43(1), 88-100.

Lishinski, A., Yadav, A., & Enbody, R. (2017, August). Students' Emotional Reactions to Programming Projects in Introduction to Programming: Measurement Approach and Influence on Learning Outcomes. *In Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 30-38). ACM.

Martínez, C., & Muñoz, M. (2014). ADPT: AN ACTIVE LEARNING METHOD FOR A PROGRAMMING LAB COURSE. In *Proceedings of the 10th International CDIO Conference, Universitat Politècnica de Catalunya, Barcelona, Spain*.

Phuong, A. P., D NGUYEN, M., NGUYEN, L. Q., NGUYEN, T. M., & Bao, N. L. E. LEARNING COMPUTER PROGRAMMING IN CDIO'S TEAM SETTINGS. In *Proceedings of the 10th International CDIO Conference (CDIO 2014), June* (pp. 15-19).

Probst, C. W. (2016) ADDING CDIO-COMPONENTS TO (NON-)CDIO COURSES *Proceedings of the 12th International CDIO Conference*, Turku, Finland, June 12-16

Reng, L., & Kofoed, L. B. ENHANCE STUDENTS'MOTIVATION TO LEARN PROGRAMMING THE DEVELOPING PROCESS OF COURSE DESIGN. In *Proceedings of the 8th International CDIO Conference.(Queensland University of Technology, Brisbane, Australia*.

Vo, Nhan-Van, Duc-Man Nguyen, and Nhu-Hang Ha. (2017) A CASE STUDY OF CDIO IMPLEMENTATION IN THE COURSE OF HACKING EXPOSED AT DUY TAN UNIVERSITY. *Proceedings of the 13th International CDIO Conference, Calgary, Canada, 90-100.*

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin*, *28*(3), 17-22.

BIOGRAPHICAL INFORMATION

Erik Berglund, Ph.D. in computer science and an Associate Professor at the Department of Computer and Information Science and has taught programming in applied courses since 1997 at the university of Linköping.

Dennis Person, Ms. C. is a Teaching assistant at Department of information and computer science and instrumental in the development of the course discussed in the paper.

Corresponding author

Dr. Erik Berglund
erik.berglund@liu.se
Linköping university
Department of Computer and Information Sceince
SE-581 83 Linköping
+46 28 10 00