MASTERY LEARNING IN INTRODUCTORY PROGRAMMING: RUNNING A PROJECT ALONGSIDE A TEST LADDER

Guttorm Sindre[1], Gabrielle Hansen[1,2]

[1]Excited Centre of Excellent IT Education, Dept of Computer Science
[2]SEED, Center for Science and Engineering Education Development
Norwegian University of Science and Technology (NTNU), Norway

ABSTRACT

Many students struggle to learn introductory programming, especially computing non-majors. Teaching designs where all students are expected to keep the same pace, will be too demanding for some students, who are left without any sense of mastery – and at the same time boring for other students who are quick learners or have previous knowledge of programming. Hence, self-pacing could be an interesting paradigm for programming courses. The current paper reports on the transition of the introductory programming course for first-year STEM teacher students at the NTNU, from a traditional lecture/exercise/exam design to a learning design inspired by mastery learning, with a series of automated tests in parallel with an individual programming project. The course design showed some positive and promising results in terms of a very low failure rate and good student satisfaction across a wide range of progress paces and ambition levels. At the same time, there were also some negative issues. While most students started early with the tests, many struggled to get started with the project, and the grade average was poor.

KEYWORDS

Programming, Mastery-learning, Assessment, Project-based learning: 2, 5, 8, 11

INTRODUCTION

Learning to program is challenging for many students (Bennedsen & Caspersen, 2019; Matthíasdóttir & Loftsson, 2020), and especially for computing non-majors (Wiedenbeck, 2005). The so-called learning edge momentum (Robins, 2010) implies that programming concepts build upon each other in such a way that students who fall behind the nominal progress of the class are likely to fall further behind rather than catch up. Hence, mastery-learning with self-pacing inspired by Keller's Personalized System of Instruction (Keller, 1968) might be an interesting intervention, and recent adaptations of mastery learning in introductory programming courses have reporting promising results despite some challenges with student procrastination (Ott, McCane, & Meek, 2021; Purao, Sein, Nilsen, & Larsen, 2017).

In Norway, there are many 5-year master programs for students seeking to become teachers. NOKUT evaluated these programs and found a need for improvement, especially related to

motivation and sense of belonging in these study programs. At the NTNU, one of these programs is for students wanting to become high school teachers in STEM subjects. A general initiative was started to improve this program. The most radical change was to design an introductory programming course specifically for this student group, rather than having them take the standard CS1 course together with many other STEM programs. Goals for the new programming course were: (1) improved sense of belonging, by giving the class one course specifically for them in the first semester, instead of just having huge auditorium lecture courses together with other programs. (2) increased professional relevance for teacher students, by looking at the usage of programming related to teaching of STEM courses in high school. (3) piloting a radically different course design, which could more easily be done in a small class of 50 students, than in one of the standard CS1 courses with 500+ students. Moreover, students who themselves are on a path to become teachers, could benefit from having been exposed to widely different course designs during their studies, rather than too many courses with a standard set-up of plenary lectures and end-of-course exams.

A central aspect of mastery learning is to have a series of tests, but such automated tests would typically focus on short code snippets and rather generic problems. Hence, it was considered necessary to have a programming project alongside this series of tests, so that the students could learn to write code. The project was also the component meant to ensure the increased relevance of the course, inspiring the students to design a program relevant for their major study discipline and future profession. The research questions we are investigating in this paper are as follows: (RQ1) How did the students perform in the course, and especially on the project running alongside an automated test series? (RQ2) How did the students experience the course in general, and the project in particular? For the first question, we look at log data showing the students' progress, as well as inspect their delivered code. For the second question, we look at students' perception as expressed through a questionnaire survey.

The rest of this article is structured as follows: Section 2 presents the design of the new course. Section 3 explains the research method for evaluating the course, whereupon findings are presented in section 4. Section 5 provides a discussion of the results, both compared to related work and providing ideas on how the course could be improved for the next offering. Finally, section 6 makes some concluding remarks.

COURSE DESIGN

Inspired by mastery learning / Keller's PSI, the course was divided into modules. The content of modules is indicated in Table 1.

Table 1. Course modules

	Key concept(s)	**Also covered**
I	Variables, assignment	Names, arithmetic operators, precedence, input, print
H	Data types, functions	Lists vs. numpy arrays, simple plotting in matplotlib
G	Conditions, branching	f-strings
F	Loops	Indexing in sequences, augmented assignment
E	2d data, double loops	Simple usage of files
D	Exception handling	String methods, files, slicing
C	Sets, dictionaries	More about list methods, mutability
B	Functions as parameters	Mitigating rounding errors
A	Recursion	More difficult problems across the curriculum

Each student could then choose their own pace and ambition level through these modules. The lowest passing grade in the Norwegian system is E (similar to the ECTS system), which would correspond to a D in the American system, and the top grade is A (there are only the letters E, D, C, B, A – no A+ or A-). Hence, the modules directly corresponded to grades. To pass, a student would need to do the 5 first modules (I, H, G, F, E), and each grade upwards from that E would require mastery of yet another module. Mastery of a module had to be documented by passing a test (starting at module I) and a project (starting at module G).

The tests were fully automated, with one test per level I, H, …, A. Each test typically consisted of 8-10 tasks, together covering the concepts of that module. Commonly used question genres were multiple choice, multiple true/false, pairing, Parsons' problems, and various code completion tasks (code with gaps to be filled). The pass threshold was initially given as 90%, and a failed test could be retaken the next week with no penalties. Summative tests for passing a module were conducted under supervision. In addition, there were practice tests for formative usage. Both types of tests drew question randomly from the same question banks, with approximately 20 variants per task, so that students would rarely get the same questions with repeated attempts – this to encourage understanding of the concepts rather than mere memorization of answers. With practice tests thus being identical to supervised tests (except for different outcomes of the random drawing) there was a high level of transparency to the tests, where practice tests could be used formally for gradual improvement before a supervised test. Seeing where they lost points on a practice test, students could then look at videos and notebooks explaining those concepts to improve their scores. Similarly, if failing a supervised test, students could look at the results and see where they needed to improve before reattempting the test the next week.

The project was done individually, with incremental deliveries through the semester – again open for different paces among students. Although team projects can have many affordances (Pee & Leong, 2005; Säisä, Määttä, & Roslöf, 2017), such as collaboration skills, students learning from their peers, we considered it too risky within a course design of self-pacing. There have been courses using team projects in mastery learning courses, such as (Jazayeri, 2015), but then the project was towards the end of the course, and only for students who had passed the previous mastery tests in nominal time, so the slower students would not be exposed to any project. Our individual project could more easily be done in parallel with the module tests and available for all students, regardless of pace and ambition level. Every student was free to define the objectives of their project and what features the code would contain, within some broad requirements: (I) The program should be intended for pedagogical usage within the student's major discipline. Hence, a student aiming to become a biology teacher should make a program to be used in the teaching of some topic within high school biology, whereas another might do something within high school math, physics, or chemistry. (II) For each module, the code must demonstrate purposeful usage of the key concepts included in that module. Hence, to reach the grade of E, the project code had to include concepts covered in modules I through E, cf. Table 1. A student who then wanted to improve the project from E to D would need to add some code to demonstrate purposeful usage of exceptions, string methods, slicing, and file handling, cf. Table 1. (III) The program must run without error and give some understandable output to the user. With each delivery, the students attached a self-evaluation checklist indicating how requirements were met (e.g., in which code lines various concepts had been used). It was decided to start project deliveries at level G, since at levels I and H students might know too little programming to write coherent code. Also, this gave students time in the beginning of the semester to think about what type of project they would want to make.

As for **grading**, there was no percentage score for the tests and project, instead students climbed a pass/no-pass mastery ladder, and if test and project were at different levels at the end of the semester, the lowest grade would result. Hence, it would be pointless for a student (at least from a grading perspective) to achieve a high level on the project while staying at a low level on tests – or the other way around. It can be noted that a student satisfied with the lowest passing score would only learn the concepts covered in the first 5 rows of Table 1, thus for instance not learning about exception handling, sets, dictionaries. The rationale for this design was that it was considered better to ensure that all passing students had good mastery of a limited number of programming concepts, so that they could actually do something useful with them, rather than giving them superficial understanding of a broader range of concepts, which might have resulted if we had tried to cover all concepts in a basic way within modules I-E and then looking at more advanced usage in further modules.

Teaching and learning resources. There was no defined textbook for the course, but Jupyter Notebooks and videos had been made beforehand for the topics covered in each module, and some example projects had been developed, at grade levels E, C, and A. The practice tests were also a key learning resource. In addition, the course had weekly seminars (10-12 every Thursday), with compulsory attendance of at least 11 of 14 seminars. During seminars, students would sit together at group tables, typically organized according to level of progress and what type of learning activity the student wanted to pursue. For instance, students practicing for the E-test would sit at one table, F-test at another table, G-test yet another table, and students wanting to work on their projects during the seminar at yet other tables. Teaching staff (teacher and TAs) would help students along the way during seminars.

RESEARCH METHOD

Student performance (RQ1) was evaluated by looking at grade statistics, plus aggregate observations about student progress on tests and project made by the teacher during the term. For privacy reasons we could not use more detailed log data from the LMS and digital exam system in the analysis. Students' perception and satisfaction with the course (RQ2) was investigated by a questionnaire survey. This survey was administered in the classroom in the beginning of the seminar in the 11th teaching week, and 44 of 48 students in the class responded, giving a response rate of more than 90% (non-responders being students who happened to be absent from that seminar). The survey was anonymous and thus in compliance with requirements by SIKT (The Norwegian Agency for Shared Services in Research). This was achieved using the national questionnaire platform Nettskjema, which explicitly avoids capturing indirect electronic identifiers from the respondents. Since the questionnaire investigation was anonymous, it is not possible to correlate student grades with the questionnaire answers, and hence impossible to see, e.g., if satisfaction was correlated with achievement.

RESULTS

Findings on RQ1: Student performance in the course

A positive result for student performance was the zero failure-rate. Table 2 shows the grade distribution resulting from our course in 2023 compared to what the STEM teacher students of 2022 got. In 2022, with a more traditional course design, the failure rate was 11% for students who did attend the exam but scored too poorly. However, there were some other less

satisfactory aspects of the performance. The grade average was only D, since more than half the class only achieved the lowest passing grade E. As shown in Table 2, less than half the class achieved better than E. Notably, the 2022 cohort of the same study program, who had a more traditional CS1 course, also had a grade average of D, but then at least on the positive side of D, so the 2023 average was 0.3 grades lower.

Table 2. Grade distributions %, 2022 (trad. exam) vs. 2023 (mastery learning)

Grade	A	B	C	D	E	F
2022	5.6	2.8	**30.6**	**33.3**	16.7	**11.1**
2023	**6.4**	**8.5**	8.5	14.9	**61.7**	0

In the first 4-5 weeks of the semester, the average speed of the class up the test ladder would have yielded a grade average closer to C than D if it had persisted, which would have been an improvement over the year before. However, then the average pace slowed down. The F-test and E-test turned out to be notably more difficult than the previous tests, some students needing several attempts on these. Also, the project work lagged the test-taking. Especially in the first half of the semester, many students had a slow start to the project. However, students had a wide range of different paces – which the self-pacing course design was also intended to allow for.

As for the projects, students were free to choose what kind of program to make, only that it had to be related to their discipline. Already in the first semester, these students choose between 5 different specializations within the program: Math + Physics, Math + Chemistry, Math + Biology, Chemistry + Biology, and Math + Informatics. Each student was required to make a project related to their specialty subject(s) and its teaching in high school. Hence, every student would have two choices, for instance, a Math + Physics student could either make a program related to the teaching of math or to the teaching of physics. Chemistry and biology were the most popular topics, while few students chose math, even if most of the class could have done so since math is included in 5 of 6 study directions. Also, few chose physics, and nobody chose informatics. Regardless of discipline, the structure of the delivered code was mostly poor. One symptom of this is the limited use of functions, as most students had the bulk of their code in the main script. There was no absolute requirement to use a particular structure for the program, but students had been recommended to decompose their code by means of functions, as the curriculum for the course was procedural programming in Python. Many students missed better teaching support on how to structure their code.

There was a wide range of exact curricular science topics covered by the programs, such as in biology: genetics, ecology, plant growth, epidemics; in chemistry: molecules, reactions, acids and bases; in physics: ballistics, free fall with air drag, bouncing and elastics; in math: equations, fractions, geometry. Also, there were some different genres of programs: calculation and visualization, data support for experiments or field trips (entering and analyzing data), simulation of natural phenomena, and quizzes. Even if they struggled, many students seemed to find it motivating to write a program with a pedagogical purpose related to their future profession.

Findings on RQ2: Student experience with the course

Figure 3, upper left, shows how students responded to the question *How satisfied are you with this course?* on a 5-point scale from very dissatisfied to very satisfied. Only a small share of the students chose the negative options, and most responses were positive. Compared with other courses they took in parallel which had a more mainstream design with lectures and end-of-course exams (upper right), 64% felt this course was better, while only 16% felt it was worse. Lower right shows responses to the statement *I have experienced mastery in this course*, again with a majority for the positive options. One aspect that students were less satisfied with, was the teaching and learning resources related to the start of the project, which many found difficult (lower right), with a clear majority responding very challenging or challenging.

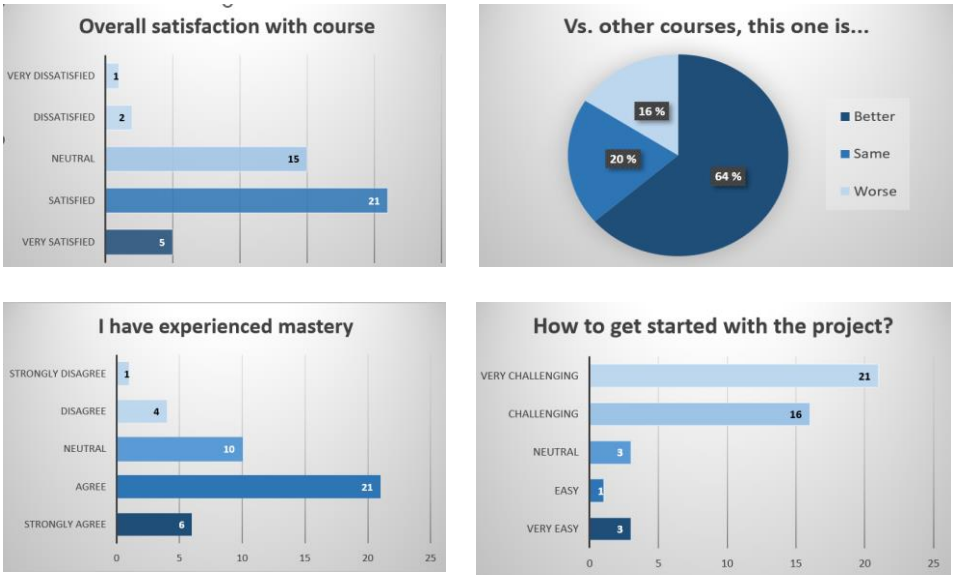

Figure 3. Course satisfaction as such (upper left) and vs. other courses (upper right), perception of mastery, and ease of starting project (all: dark = better, light = worse)

It can also be interesting to look at the students' opinions on the usefulness of various learning activities and resources. This is shown in Figure 4, with the columns left to right being very low to very high usefulness. The one thing that really stands out in this diagram is the very high perceived usefulness of the practice tests. No students have given negative answers about these, and a clear majority (37/44 respondents) have rated them as very useful. Other resources also get good scores, except the demo projects, where there are more negatives than positives.

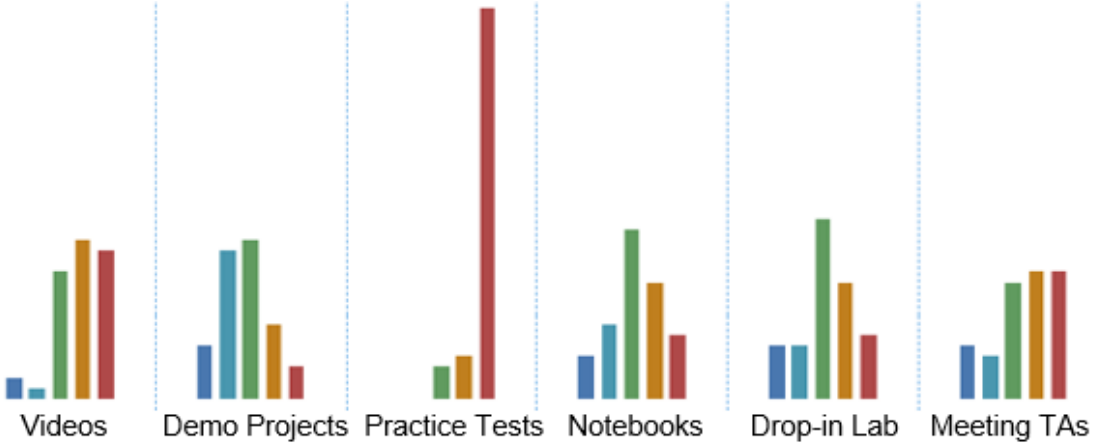

Figure 4. Perceived usefulness of six different learning resources in the course, each with responses from very low (left, dark blue) to very high (right, red)

Students were also asked several other questions in the questionnaire, but for space reasons we cannot present all the results here. One of the questions was about which learning resources they used in connection with the project work (multiple selections allowed), the results were tutors (82%), peers (73%), teacher videos (55%), demo projects (41%), practice tests (32%), and other (30%) – which might have been other internet coding resources such as StackOverflow or YouTube videos, or generative AI like ChatGPT. Since only 32% reported usage of practice tests in connection with the project, while these are the clear "winner" in the overall usefulness questions shown in Fig.4, it seems reasonable to assume that the practice tests were primarily considered useful for passing the supervised tests, and less (but not zero) for doing the project.

DISCUSSION

Related work

Mastery learning has been used in many universities and disciplines throughout the years. Key ideas for mastery learning emerged in the 1960's, in particular Bloom's *Learning for Mastery (LFM)* and Keller's *Personalized System of Instruction (PSI)* (Bloom, 1968; Keller, 1968). One earlier example of a PSI-inspired approach to introductory programming is (Purao et al., 2016), which like our course allowed for a high degree of student self-pacing. Unlike our course, they did not decide grades solely based on the passing of modules but by an end-of-course exam, and they had several smaller programming exercises rather than an incremental project. Indeed, many applications of mastery-learning in introductory programming have been hybrid, combining a mastery ladder with a traditional end-of-course exam and some plenary lecturing (Garner, Denny, & Luxton-Reilly, 2019), and few have combined mastery learning with a project. An exception is the previously mentioned approach by (Jazayeri, 2015) including a team project, though differing from ours in that the project was only towards the end of the course, for those who had passed the other modules, hence the weaker students would not get any project. The course reported by the paper (Toti, Chen, & Gonzalez, 2023) had a course design resembling ours in that the passing of modules was directly linked to grades. They had project as the very last of 12 course modules, which would thus only be taken by students going for the top grade.

Interpretation of Findings

For RQ1 about performance, the results were mixed. On the positive side, the failure rate was much smaller than previous years. On the negative side, there were fewer students achieving grades C and D, and instead more achieving E, so that the overall grade point average was weaker than the year before. Partly this may have been because some students decided to stop at E, thus having secured a passing grade in the programming course while they were more worried about some of the other courses they were taking in parallel. Some ended on E because that was the best they could manage within the time and capacity they had available, or because they believed it was the best they could manage, having struggled with the F and E tests and fearing that the D test was even more difficult. Others may have easily been able to take more tests, having passed the E test already by mid-semester – but had made less progress on the project, so then took a break from testing to work more on the project for a while.

For RQ2 about satisfaction, students seemed overall quite satisfied with the course, and most of them considered it better than other courses taken in parallel. However, this finding must be taken with some caution. The questionnaire survey was conducted in week 44, i.e., three weeks before the end of the teaching period. At this point, many students had already secured a passing grade in our course, while in other courses with a final exam, they obviously had not. Hence, some may have been more worried about, and less satisfied with, other courses for that reason. The very high level of satisfaction with the practice tests must also be taken with some caution. In a way, this is a self-fulfilling prophecy. Since the practice tests were identical to the supervised tests (drawing questions randomly from the same question banks) they will obviously have been perceived as highly relevant in preparing for the tests, which again linked directly to grades. The videos (second most popular learning resource) were also structured in a way that was closely aligned with test tasks. What can be derived from this is that students appreciated the high level of transparency inherent in this test ladder design. This transparency made it clear what you had to learn to obtain a certain grade, thus making it possible to take the lowest passing grade E in a controlled manner. With a final exam, on the other hand, some students who would be satisfied with an E will end up preparing to a higher level to have a margin of error at the exam.

Another positive take-away from the course is that even with self-pacing causing the class to be spread over several different grade levels already by mid-semester, and although projects were individual, students report collaborating a lot with their peers both in preparing for tests and in project-work. A key factor for facilitating this was probably the seminars with compulsory attendance, which helped students find other students who were at the same testing level (for collaborating about test preparation), or who, despite different project topics, had similar challenges with those projects (e.g., how to use a loop, read data from file, or plot a graph). Seminars also contributed to a sense of belonging in the class and study program.

CONCLUSIONS

From the results, in particular the overall student satisfaction and low failure rate, a course design with a mastery ladder directly linked to grades, and with an incremental project in parallel with a series of pass/fail mastery tests, can be viable. However, this first pilot offering of the course also had some issues which need to be improved. Some of the tests were too difficult when the pass requirement was 90%, in particular the F and E tests. The learning resources for the project were insufficient. The demo projects may have given the students good examples of what a finished project might look like, at levels E, C, and A. However, what

they needed most help with was how to get from nothing to level G, and then onwards to F and E. Hence, before the next offering we will develop better learning resources and more scaffolding for the project, in the form of templates they might start with and adapt to various topics, rather than staring coding from an entirely blank editor window.

FINANCIAL SUPPORT ACKNOWLEDGEMENTS

This work was conducted by the Excited Centre for Excellent IT Education, funded by the Norwegian Directorate for Higher Education and Skills (HK-dir).

REFERENCES

Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM inroads, 10*(2), 30-36.

Bloom, B. S. (1968). Learning for Mastery. Instruction and Curriculum. Regional Education Laboratory for the Carolinas and Virginia, Topical Papers and Reprints, Number 1. *Evaluation comment, 1*(2), n2.

Garner, J., Denny, P., & Luxton-Reilly, A. (2019). *Mastery learning in computer science education.* Paper presented at the Proceedings of the Twenty-First Australasian Computing Education Conference.

Jazayeri, M. (2015). *Combining mastery learning with project-based learning in a first programming course: An experience report.* Paper presented at the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering.

Keller, F. S. (1968). Good-bye, teacher…. *Journal of applied behavior analysis, 1*(1), 79-89.

Matthíasdóttir, Á., & Loftsson, H. (2020). *Improving the Implementation of a First-Semester Programming Course.* Paper presented at the Proceedings of the 16th International CDIO Conference.

Ott, C., McCane, B., & Meek, N. (2021). *Mastery learning in cs1-an invitation to procrastinate?: Reflecting on six years of mastery learning.* Paper presented at the Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1.

Pee, S., & Leong, H. (2005). *Implementing project based learning using CDIO concepts.* Paper presented at the 1st annual CDIO Conference.

Purao, S., Sein, M., Nilsen, H., & Larsen, E. Å. (2017). Setting the Pace: Experiments With Keller's PSI. *IEEE Transactions on Education, 60*(2), 97-104.

Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education, 20*(1), 37-71.

Säisä, M., Määttä, S., & Roslöf, J. (2017). *Integration of CDIO skills into project-based learning in higher education.* Paper presented at the Proceedings of the 13th International CDIO Conference.

Toti, G., Chen, G., & Gonzalez, S. (2023). *Teaching CS1 with a Mastery Learning Framework: Impact on Students' Learning and Engagement.* Paper presented at the Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1.

Wiedenbeck, S. (2005). *Factors affecting the success of non-majors in learning to program.* Paper presented at the Proceedings of the first international workshop on Computing education research.

BIOGRAPHICAL INFORMATION

Guttorm Sindre holds MSc and PhD degrees in Computer Science (1987, 1990). He has been a full professor at NTNU since 2003, where he served partly as Head of Dept, partly as deputy Head of the CS department 2009-13. He was the leader of the Excited Centre for Excellent IT Education from 2016-21, is currently deputy leader of Excited, and study program board leader of Informatics at the NTNU. Sindre has teaching experience across a wide range of IT topics, from first-year introductory programming to PhD level research courses, as well as supervising several master and PhD students. His research has focused on computing education, especially how to teach introductory programming, how to mitigate threats to assessment integrity, and (before the Excited centre) on software requirements engineering and software security.

Gabrielle Hansen is a senior researcher at the Excited Centre of Excellent IT Education. She holds an MSc in Psychology and a PhD degree in Pedagogy with a thesis focusing on the use of feedback in higher education. Before joining the Excited centre, she has been a researcher at the Sør-Trøndelag University College (HiST) and then at the SEED Centre for Science and Engineering Education at the NTNU, in both cases working on educational improvement interventions and the coaching of teachers during such processes. In addition to research on feedback and assessment, she has also done research on the usage of Student Response Systems in lectures.

Corresponding author

Guttorm Sindre
NTNU Norwegian University of Science and Technology
Excited Centre of Excellent IT Education
Department of Computer Science
IT building, Sem Sælands vei 7
7491 Trondheim, NORWAY
guttorm.sindre@ntnu.no