USING GITHUB CLASSROOM IN TEACHING PROGRAMMING

Jens Bennedsen, Till Böjttjer, Daniella Tola

Dept. of Electrical and Computer Engineering, Aarhus University, Denmark

ABSTRACT

Teaching students programming can be done in many different ways. One is to use Test-Driven Development (TDD) where the students can receive immediate feedback on their implementations, to correct them before submitting their assignments. The article describes a study performed on first-semester bachelor students in computer engineering in an introductory course on programming. Various tools were used to support the students learning, namely, GitHub Classroom, Visual Studio Code, and repl.it. The article discusses the pros and cons of using TDD together with the mentioned tools for an introductory course in programming. The results are based on a questionnaire filled in by the students to understand the outcome from the students' perspective, and also based on the experience from the teachers' point of view. The results were mainly positive from both the teachers' and students' points of view, with a few aspects where there were trade-offs and things that can be done differently.

KEYWORDS

Programming, GitHub, Professional Skills, Test-Driven Development, Software Development
CDIO Standards 2, 8, 11

INTRODUCTION

Becoming a professional software engineer (or other professional careers) requires that you ensure that the software you create is correct. We have – for many years – observed that our students in the first couple of years find it difficult to focus on both the creative and constructive process of "programming" and ensuring that their product ("the program") is correct.

Many modern software development methods prescribe that one should create the test of elements of the program before implementing the functions (Beck, 2003). This is known as Test-Driven Development (TDD), something that applies to software and is relevant to many other engineering disciplines.

We have started to use GitHub Classroom to support the students in their "programming journey" (GitHub, 2022). We are not the first ones to do so (see, e.g. Hsing and Gennarelli (2019)), but we are, as far as we know, the first to structure our use of it following the "Use – modify – create" (Lee et al., 2011) principle for structuring course activities.

The article discusses and evaluates one way of implementing Test-Driven Development (or one

could call it Test-Driven Programming since the focus is not the entire development of a product from conceiving to implementation) using GitHub Classroom. It is based on quantitative data from a questionnaire sent to all students at the end of their course.

The article is organized as follows: firstly, it frames the work within the general knowledge area of introductory programming. Then it describes the research design, leading to an analysis of the data. Lastly, future work and the future development of a programming course based on (among other things) GitHub Classroom and TDD is described.

RELATED WORK

This section starts by summarizing general trends in learning to program and then focuses on others' work using GitHub in their introductory programming courses.

Trends in Learning to Program

Software development competencies have become more in need by the industry over the last many years (Istiyowati, Syahrial, & Muslim, 2020; US News, 2021). One of the core software development competencies is programming. However, many students experience challenges when learning to program (see e.g. Corney, Teague, and Thomas (2010); Guzdial (2010)). In her PhD thesis, Kaila (2018) states *programming is a very difficult skill to learn, and even more difficult skill to master. After introductory courses, various students typically still have difficulties in reading the program code and writing simple programs. Moreover, the dropout rates in introductory programming courses are typically quite high (p. 1)*. In various CDIO conferences, scholars reported on their challenges and experiences with teaching programming (see e.g. Martínez and Muñoz (2014); Matthíasdóttir and Loftsson (2019, 2020)).

Various approaches have been proposed for supporting students' learning to program. Some approaches focus on a structuring principle for the course (e.g. objects first (Cooper, Dann, & Pausch, 2003) or creative computing (Xu, Wolz, Kumar, & Greenburg, 2018)). Other focus on different tools for helping the students learn to program; for an overview see Naps et al. (2002); Sorva, Karavirta, and Malmi (2013). As an example, Sorva et al. analyses 46 different visualization systems built until the article was published in 2013; many more have been added after that (see e.g.(Staugaard, 2020) for a list of additional systems). The guiding principles for the course used in this research is described in section "The Course".

Giving feedback to students learning to program is a difficult and time-consuming task. Different approaches have been suggested to ease the task for the teachers (e.g. the use of automated feedback systems (Muuli et al., 2017; Thangaraj, 2021), the use of automatic calculation of different metrics of quality of code (Zaw, Hnin, Kyaw, & Funabiki, 2020))). The main objective of all these approaches is, that the teacher should spend time on giving feedback that "matters" and not on trivial things like syntax issues, indentation etc.

Test-Driven Development

As described in the Introduction, many modern software development methods have tests as a central part of specifying the functional requirements for a given piece of code. In general *Test-*

Driven Development is associated with extreme programming (Beck, 2000) and was initially described by Beck (2003). It is an iterative development process with the following steps:

- **Add a test**: When a new feature is needed in the program, it is specified by test case(s) such that if the test passes, the specifications are met.
- **Run all tests**: The systems should fulfil all but the newly added tests (which should fail for expected reasons).
- **Write code that fulfil the tests**: If some tests are not met, the code must be revisited.
- **Refactor if needed**: Modify the code so that it fulfils the quality standards. When doing so, ensure that the tests are still being met.

The course in question did not focus on the refactoring part and, in most cases, the code that the students should write was standalone, not a part of a big system (and thus, there were not a large pool of tests before the new feature (bullet one) was introduced).

Use of GitHub in Teaching Programming

Glassey (2019) surveyed eight publicly available version control tools including GitHub Classroom to help teachers select a solution for their courses. Technical features and pedagogical aspects of the tools were illustrated including 1) Repository creation and distribution to the students, 2) Team creation for a project or peer assessment, 3) Batch cloning of repositories of students repositories for assessment and evaluation.

Angulo and Aktunc (2019) studied the benefits and challenges of using GitHub for courses in a software engineering program for multiple years. Specifically, GitHub was used for teaching an Object-Oriented Programming and Design course and Java and Applications course. Initially, students were familiar with Learning Management Systems e.g. Blackboard but had no prior experience with GitHub. The authors report minimal challenges when introducing GitHub. Nonetheless, after a demonstration of the main functionality students became comfortable users of GitHub within 2 weeks. Further, students were able to collaborate (branching and merging) on group projects throughout the semester while maintaining the transparency of individual contributions for the teacher. The authors explain that creating and managing various GitHub repositories becomes challenging with an increasing number of students and assignments. They plan to adopt the GitHub Classroom application due to the simplicity of publishing and collecting assignments.

Glazunova, Parhomenko, Korolchuk, and Voloshyna (2021) focus on teaching collaborative software development through GitHub Classroom on the example of 29 Computer Science and Engineering students. The teacher combined the Learning Management System and GitHub to share theory, instructions and results, and allow the students to implement project tasks. The interviews showed that students favoured three features of GitHub; collaborative development of software, ease of bug tracking and accessibility of the code editor.

Diehl and Brandt (2020) present the use of GitHub Classroom to provide an interactive C++ development environment and introduce students to the concept of Git. They surveyed a group

of 10 students. Students reported that they particularly enjoyed the interactive notebook feature for creating and testing their C++ code.

RESEARCH DESIGN

This section describes the research design. It starts with our research hypothesis, then gives the context of the research (i.e. the course, the participants etc.).

Research Hypothesis

One of the core ideas of CDIO is an integration of the student's technical skills and their professional skills. In many cases, there is a tension between the professional tools and processes students use when they work in industry and the tools used.

As described in Related work, many find learning to program difficult. Furthermore, much time is spent on feedback on low-level problems (the program cannot compile, the program fails the simplest tests etc). Our research hypothesis is, therefore:

> *Beginners find it easier to learn to program using TDD with GitHub and such tools makes it possible for the teachers to focus on giving higher-level feedback*

Research Context

This section describes the course, the tools used and the participants.

The Course

The research is done in an introductory programming course in the first semester of a bachelor of engineering program at Aarhus University, the second-largest public university in Denmark. The course is 10 ECTS (that is $1/3$ of the time in the semester should be spent on this course).

At the end of the course, the participants will be able to (Aarhus University, 2022):

- Describe and discuss commands and control structures of imperative programming;
- Describe the relationship between iteration and recursion;
- Describe and discuss structuring mechanisms in different programming styles;
- Implement their own programs using different programming styles;
- Explain the concept of imperative and functional programming;
- Describe assertional techniques for reasoning about programs;
- Reason informally about programs and relate this to tests.

Our guiding principles in the course used for this research could be described as:

- **Simplicity first**: Starting with the simple programming constructs and gradually enhancing the complexity.
- **Use-modify-create**: The students firstly see a programming construct, then they modify existing code and lastly they create new code.
- **Specification-Driven Development**: Before creating code, the students read, modify or create a specification. The specification includes pre- and postconditions as well as test cases.
- **Program is a verb**: We focus on the programming process (program as a verb) not just the program itself (program as a noun).

During the course, the students had to hand in 12 assignments. The assignments were graded (pass/fail) by two teaching assistants (the second and third author), and the student had to pass all 12 assignments to take the final exam. The course is divided into two face-to-face activities: Lectures and Programming café. In the café students can get help with their assignments.

The Tools Used

The main tools used during the course are: Replit (2022), GitHub (2022), and Visual Studio Code (2022). A test framework was only included for the assignments distributed through Github Classroom.

GitHub Classroom is a tool that allows teachers to create a template repository containing code, which can be distributed to students via a link. Repositories for each student is created when they log into GitHub and activate the link. The students submit their assignments by sharing the link to their GitHub repository with the assignment code.

The first tool learnt by the students was repl.it. Repl.it is an online IDE and compiler. It was mainly used for the students to get acquainted with the basics of programming. The students created their programs in repl.it and shared the link to their assignments with the teachers.

After four weeks of using repl.it, the students were introduced to VS Code and Github, where they had to download and install these programs, a C-compiler and other related programs on their own device. We selected VS Code for this course based on the fact that it is an open-source IDE, with the possibility to install extensions and to easily tailor it to one's needs. It was also the most used IDE in 2021, according to a developer survey performed by Stack Overflow (2022).

The assignments were created and distributed using GitHub/Github Classroom with skeleton projects. A project included test cases for each of the functions the students should implement as well as header and C files. An example repository can be found at `https://tinyurl.com/assgnment`. The students could run the test cases and get immediate feedback on their implementation. Errors in the implementation would show in the test results, allowing the

```
[ctest]     Start 1: usage_test
[ctest] 1/1 Test #1: usage_test ......................   Passed   40.48 sec
[ctest]
[ctest] 100% tests passed, 0 tests failed out of 1
[ctest]
[ctest] Total Test time (real) =  40.50 sec
[ctest] CTest finished with return code 0
```

Figure 1. Example output from a test run. Here the test PASSED.

students to narrow down the problem and correct the code. In later stages of the course, the students were allowed to modify and create their own test cases.

The usage of the *Use - Modify - Create* concept in this study is illustrated in figure 2. The students learnt to *use* and understand the predefined test cases that were made to determine if the students had implemented their assignments correctly. The idea was that the students should both get automatic feedback and learn that defining test-cases is a nice way of specifying the functional requirements. After the students were more experienced, minor errors were put into the tests cases, and the students were told to find and fix these as well as extend them. This is where the *modify* part comes into play, as the students need to learn how to modify the test cases, to ensure they are correct. The final step was for the students to create their own test cases from scratch. The difficulty increased gradually from the *use* step, to the *modify* step, and to the *create* step.

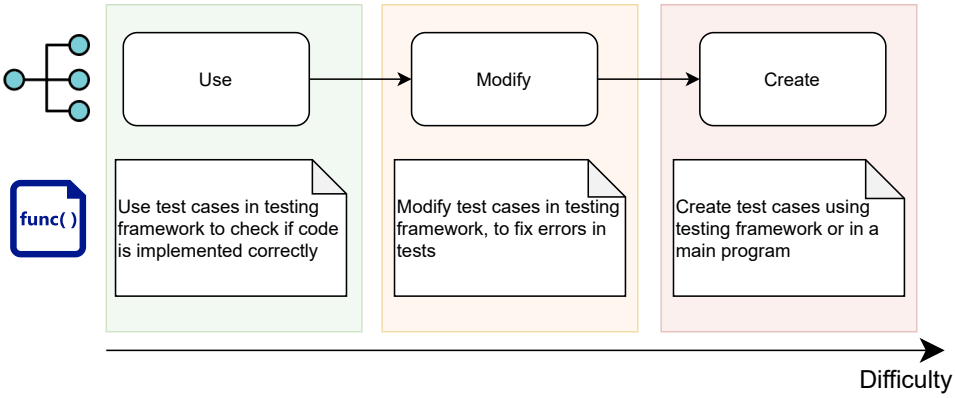

Figure 2. The main elements used in this study following the Use-Modify-Create structure, where the difficulty increases with each step.

The Participants

35 students participated in the course. Most of the students came from high school; a few of them have started another study program before this. Of the 35 students, only three were female. Approximately half of the students had programmed before (in many different programming languages/systems).

Data Collection

The data for this research was collected using a questionnaire at the end of the semester containing both closed questions (on a five point likkert scale) and open-ended questions. The questionnaire is sent out automatically to all students who participated in the course. The main purpose of the questionnaire is the quality assurance process of the university, but teachers can add both scale and free text questions to the questionnaire. To get a higher response rate, time during a lecture was allocated to allow the students to reply to the questionnaire.

The authors are the lecturer and the two teaching assistants for the course. Evaluation of their experiences is done through discussions among them.

ANALYSIS

This section analyses the data from the students (responses to the questionnaire) and describes the teachers' experiences. The closed questions was used for quantitative analysis. A generalizable study with statistical testing of the hypothesis requires a larger sample size.

The students' perspective

As described in Data Collection the questionnaire was distributed to 35 students. 20 students responded; the response rate was 60%. One student answered part of the questionnaire.

In general, the students found the outcome of the course significant (18 out of 21 answered either "very great outcome" or "significant outcome"). They found the course well organized (18 out of 21 either "agreed" or "mostly agreed" with that statement), and relevant for their studies on the whole (all either "agreed" or "mostly agreed" with that statement).

The students found it somewhat difficult to get the course infrastructure installed (compiler, git etc.) as can be seen in figure 3. Especially students who use a mac found it difficult. Some of the students were not present when the setup was introduced, and some found that the supporting material was not detailed enough (we made a video and a text document explaining the setup).

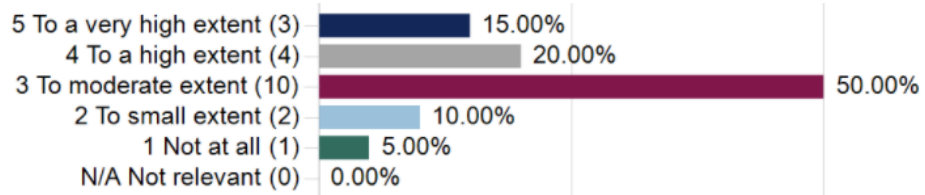

Figure 3. How challenging has it been to get the course infrastructure (compiler, git etc.) installed?

Some of the students found the transition from repl.it to GitHub challenging. It required a few weeks to get used to the new way of handing in. A few students had difficulties for quite some time and we (the teachers) could not help, since the problems were related to mac and none of the teachers had experience with a mac. As one of the students wrote *The main part of the*

time spent on the hand-ins was not spent on programming but on solving problems with the compiler and GitHub (translated from Danish). Difficulties with the transition may be resolved by introducing a clear schedule for the exercises and transition during the first lecture and by describing the motivation for the transition.

One of our rationales for introducing GitHub was that the students should learn a professional tool. When asked "Do you think the tools you have learned in the course (especially GitHub, VS Code) will be relevant to your academic/professional career? How?" only a few of the students could make a connection to their future career.

All in all, it is difficult to answer "accept" or "reject" on the students' part of our hypothesis. There have been some practical issues with the tools but it seems like the students have found them useful in their learning.

The students wish for better distribution of the difficulty of the exercises; this was a reappearing comment throughout the questionnaire. One student asked for larger freedom in creating the program structure. However, our aim was to guide the students during the initial exercises by providing a problem definition through the LMS and header files, data types and function skeletons through GitHub Classroom. Only during the last exercise the students were given an empty repository and were asked to create C files, header files and test cases.

The majority of the students reported that good opportunities for feedback and counselling regarding their academic performance were given (80% agree or mostly agree).

Noticeable disagreement was given when asked if the academic qualifications for participating in the course were good (45% agree or mostly agree). However, after participation in the course, the majority reported that their programming skills are sufficient to complete the course (75% agree, mostly agree or answer neutral when asked if their skills are above what is expected).

Thirteen students answered whether they felt the test cases helped them to see if their code was implemented correctly. Six students answered positively, 2 students answered negatively, and 5 students answered neutrally. There had been a few issues with some of the test cases making it difficult for some of the students to work with. The students that answered neutrally mainly viewed the test cases as a help, but did not like the fact that there were some issues with the test cases and had difficulties understanding them.

The students were asked to comment on what programming environment(s) and programming languages they used before enrolling in the course. Only one of the students had programming experience and worked in various languages. Two students used VS Code before enrollment. The majority of the class did report little experience (6 students) to no prior knowledge (14 students).

The teachers' perspective

There is a trade-off between using test cases for scoring assignments, and manually understanding and checking if the assignments are implemented correctly. It is more time consuming to correct the assignments manually, but the feedback given to the students is more precise and

helpful, which is exactly what is needed in the beginning of such a programming course. Correcting the assignments using the test cases minimizes the amount of time spent on checking the code, but also reduces the precision of the feedback given.

Many of the students found interpreting the output of running the tests a little difficult (see an example output in figure 1). It is not a nice and user-friendly output like many know from apps or other programs, so in retrospect, we should have spent more time introducing this part. In general, we should have spent more time introducing the "programming process" using the tools: when you have made a small part of the implementation, run the tests, interpret the results (and be aware that tests for non-implemented parts will fail) and modify your code based on the analysis of possible causes of failing tests.

One of the advantages of using GitHub and also repl.it, seen from a teacher's perspective, is that when the students have errors that may be difficult for them to understand how to solve, it is possible for the teachers to upload changes to the repository, to guide them in the right direction. This was especially useful if students had made errors related to the setup of the project, making it difficult to compile.

One of the disadvantages of using the test cases is that sometimes the students would have errors in one file, which resulted in the project not being able to compile. Some parts of the students' assignments could be implemented correctly, but due to errors in other parts of the assignments, the project could not be compiled altogether. This meant, that the teacher would have to either fail the students or fix the compiler issues themselves which in some cases was time-consuming.

The integration of GitHub Classroom and the LMS is missing, and therefore added an extra step for the students and teachers when handing in and correcting assignments.

DISCUSSION, FUTURE WORK AND IMPROVEMENTS

Learning to program is a process. In the beginning, many students struggle with syntactic issues (like a missing ";" at the end of a line), and test-cases do not help here. It is therefore important that the feedback (or feed-forward) in the beginning recognises this and is very detailed. Later in the course, feedback can focus on structure and less on details. In the next run of this course, we will make this even more happen.

Compiling and running code on different computers and operating systems can give different results, depending on the compiler used, the compiler settings and the computer architecture. When compiling the students' assignments, the teachers used the same compiler that the students had installed, to ensure the output of the compilation was as similar as possible. Even though this was done, there were still issues with running the code on different operating systems or computers with different architectures. One of the issues was a segmentation fault occurring on the teacher's computer but not on the student's. To avoid issues related to this, it would make sense to increase the compiler error and warning levels to the highest, to achieve as similar results as possible.

To run the test cases when correcting the assignments, the program must be able to compile.

If it could not compile, the teacher could choose to either fail the student or try fixing the issue. The reason it would make sense to fix the issue is that the students could have implemented most of the assignment correct but due to a minor issue, the program could not compile. In such a case, the student might have enough of the assignment correct to pass, but due to the compile errors, it is difficult to determine this since the test cases could not be run.

More focus will be put on the transition from repl.it to GitHub Classroom, to ensure the students understand every step and to avoid confusion. A potential assignment for the transition could be, that the students must copy their code from an assignment created in repl.it, and hand it through GitHub. In this assignment, the only new aspects the students will need to learn is how to use GitHub and how to hand in their assignment using GitHub.

CONCLUSION AND RECOMMENDATIONS

There is no silver bullet (Brooks & Kugler, 1987) in learning to program. It is a difficult and challenging task. The use of Test-Driven Development and tools supporting this process can be beneficial, but it is only one component as described and analysed in the paper.

Introducing technology such as GitHub requires that the students are comfortable with the tools and can understand their use. One recommendation from our research is, therefore, to remember to introduce the tools and especially the benefits for the students when using the tools. In our case, that could have been done better. On the other hand, the tools are easy to use and helps free up time to focus on.

The tools that we used in this study are not all necessary for working in such a manner, but they make it easier for the teachers and the students to work in this way. GitHub Classroom makes it easy to distribute the assignment by reducing the number of steps required, i.e. the students don't need to create their own GitHub repository and copy the assignment files into it, instead, they can just open a link that automatically creates a repository and clones the assignment for them.

Although we believe using the tools as we did in this study is a great way for the students to learn, it is important to understand that there is a significant amount of time the students spend on learning to use the different tools. This can be seen as a small overhead in the method used in this study.

Many institutions have a "bring your own device" policy – including Aarhus University. Using infrastructure that must be installed and with many different options require support. In our case, it would have been better if one of us had experience with macOS and could help students using Mac computers.

FINANCIAL SUPPORT ACKNOWLEDGEMENTS

This work has not been financially supported.

REFERENCES

Aarhus University. (2022). *Programming for computer engineering.* Retrieved 2022-01-10, from https://kursuskatalog.au.dk/en/course/107799/Programming-for-Computer-Engineering

Angulo, M. A., & Aktunc, O. (2019). Using github as a teaching tool for programming courses. In *2018 gulf southwest section conference.*

Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Brooks, F., & Kugler, H. (1987). No silver bullet. In J. P. Bowen & M. G. Hinchey (Eds.), *High-integrity system specification and design* (p. 11-27). Springer.

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in introductory computer science. In *Proceedings of the 34th sigcse technical symposium on computer science education* (p. 191–195). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/611892.611966 doi: 10.1145/611892.611966

Corney, M. W., Teague, D. M., & Thomas, R. N. (2010). Engaging students in programming. In *Conferences in research and practice in information technology, vol. 103. tony clear and john hamer, eds.* (Vol. 103, pp. 63–72).

Diehl, P., & Brandt, S. R. (2020). Interactive c++ code development using c++ explorer and github classroom for educational purposes. *Proceedings of Gateways*, 5.

GitHub. (2022). *Github classroom.* Retrieved 2022-01-07, from https://classroom.github.com

Glassey, R. (2019). Adopting git/github within teaching: A survey of tool support. In *Proceedings of the acm conference on global computing education* (pp. 143–149).

Glazunova, O., Parhomenko, O., Korolchuk, V., & Voloshyna, T. (2021). The effectiveness of github cloud services for implementing a programming training project: students' point of view. In *Journal of physics: Conference series* (Vol. 1840, p. 012030).

Guzdial, M. (2010). Why is it so hard to learn to program? In A. Oram & G. Wilson (Eds.), *Making software: What really works, and why we believe it* (p. 111-124). Sebastopol, CA, USA: O'Reilly.

Hsing, C., & Gennarelli, V. (2019). Using github in the classroom predicts student learning outcomes and classroom experiences: Findings from a survey of students and teachers. In *Proceedings of the 50th acm technical symposium on computer science education* (pp. 672–678).

Istiyowati, L. S., Syahrial, Z., & Muslim, S. (2020). Programmer's competencies between industry and education. *Universal Journal of Educational Research*, *8*, 10-15.

Kaila, E. (2018). *Utilizing educational technology in computer science and programming courses : theory and practice* (Doctoral dissertation, University of Turku, Finland). Retrieved 2022-01-20, from https://www.utupub.fi/handle/10024/144535

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., ... Werner, L. (2011). Computational thinking for youth in practice. *Acm Inroads*, *2*(1), 32–37.

Martínez, C., & Muñoz, M. (2014). Adpt: An active learning method for a programming lab course. In *Proceedings of the 10th international cdio conference.*

Matthíasdóttir, Á., & Loftsson, H. (2019). Flipped learning in a programming course: Students' attitudes. In *Proceedings of the 15th international cdio conference.*

Matthíasdóttir, Á., & Loftsson, H. (2020). Improving the implementation of a first-semester programming course. In *Proceedings of the 16th international cdio conference.*

Muuli, E., Papli, K., Tõnisson, E., Lepp, M., Palts, T., Suviste, R., ... Luik, P. (2017). Automatic assessment of programming assignments using image recognition. In *European conference on technology enhanced learning* (pp. 153–163).

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., ... Velázquez-Iturbide, J. A. (2002). Exploring the role of visualization and engagement in computer science education. In *Working group reports from iticse on innovation and technology in computer science education* (p. 131–152). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/960568.782998 doi: 10.1145/960568.782998

Replit. (2022). *Replit.* Retrieved 2022-01-11, from https://replit.com/

Sorva, J., Karavirta, V., & Malmi, L. (2013, November). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education*, *13*(4). Retrieved from https://doi.org/10.1145/2490822 doi: 10.1145/2490822

Stack Overflow. (2022). *Stack overflow developer survey 2021.* Retrieved 2022-01-11, from https://insights.stackoverflow.com/survey/2021#most-popular-technologies-new-collab-tools

Staugaard, J. T. (2020). *Teaching object-oriented programming to novices by connecting reality and code using visualisation* (Unpublished master's thesis). IT University, Denmark.

Thangaraj, J. (2021). Automated assessment & feedback system for novice programmers. In *Proceedings of the 26th acm conference on innovation and technology in computer science education v. 2* (p. 672–673). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3456565.3460021 doi: 10.1145/3456565.3460021

US News. (2021). *100 best jobs.* https://money.usnews.com/careers/best-jobs/rankings/the-100-best-jobs.

Visual Studio Code. (2022). *Visual studio code.* Retrieved 2022-01-11, from https://code.visualstudio.com/

Xu, D., Wolz, U., Kumar, D., & Greenburg, I. (2018). Updating introductory computer science with creative computation. In *Proceedings of the 49th acm technical symposium on computer science education* (p. 167–172). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3159450.3159539 doi: 10.1145/3159450.3159539

Zaw, K. K., Hnin, H. W., Kyaw, K. Y., & Funabiki, N. (2020). Software quality metrics calculations for java programming learning assistant system. In *2020 ieee conference on computer applications(icca)* (p. 1-6). doi: 10.1109/ICCA49400.2020.9022823

BIOGRAPHICAL INFORMATION

Jens Bennedsen is a Professor in engineering didactics. He received an M.Sc. degree in Computer Science from Aarhus University in 1988 and a Doctor Philosophiae degree in Computer Science from Oslo University in 2008. His research area includes teaching introductory programming, educational methods, technology and curriculum development methodology. He has published more than 80 articles at leading education conferences and in journals. He has previously served as co-leader of the CDIO European region. ORCID 0000-0003-3014-7567

Till Böttjer is a PhD candidate at the Department for Electrical and Computer Engineering. He received a M.Sc. degree in Mechanical Engineering from Aarhus University in 2020. His research focus are core technologies for implementing Digital Twins in industrial practice such as modelling and simulation, system monitoring etc.

Daniella Tola is a PhD candidate at the Department of Electrical and Computer Engineering at Aarhus University. She received a M.Sc. degree in Computer Engineering from Aarhus University in 2020. Her research focus is in making robot system integration easier by developing configuration tools and digital twins for these types of systems.

Corresponding author

Jens Bennedsen
Aarhus University
Dept. of Electrical and Computer Engineering
Finlandsgade 22, DK-8200 Aarhus N
Denmark jbb@ece.au.dk