An Experience with and Reflections on Live Coding with Active Learning

Anders Schlichtkrull
Department of Computer Science, Aalborg University Copenhagen, Denmark

Abstract

In this paper I report and reflect on a concrete experience with changing an introductory programming course from being based on "classical lectures" to being based on live coding with active learning. The experiment is built on learnings found in the literature and the pedagogical theories of scaffolding, think-pair-share and teaching as facilitation of learning. I reflect on the students' reaction to the experiment, the difficulty of the active learning, how to keep time, coverage of learning objectives, the degree of improvisation and student involvement. The experiment was well received by the students, and I report also on the feedback. My hope is that educators who want to introduce live coding with active learning will be able to draw inspiration from my preparation of, execution of and reflections on the experiment.

2012 ACM Subject Classification Applied computing → Education

Keywords and phrases Live coding, active learning, teaching programming

Digital Object Identifier 10.4230/OASIcs.ICPEC.2023.14

Acknowledgements I thank my colleagues Kurt Nørmark and Jens Myrup Pedersen for providing me inspiration and ideas for changing the course and also for comments on an earlier version of this text. I thank my colleagues Johannes Bjerva, Søren Qvist Eliasen, Andrés Masegosa, David Maya-Drysdale and René Rydhof Hansen for discussions. I thank Aalborg University for giving me the opportunity to do this experiment as part of its University Pedagogical Program.

1 Introduction

In the fall of 2020, I taught the 5 ECTS course "Imperative Programming" (IMPR) with ∼35 students at the first semester of the Software BSc education of Aalborg University Copenhagen (AAU CPH). For the lectures, a large seminar room was booked. In 2021 I would teach this course again – this time with ∼55 students. The course material consists of a book [4], a set of video lectures for each week and a set of exercises for each week. The videos were prepared by my colleague Kurt Nørmark for his version of the course which runs on Aalborg University's campus in Aalborg.

In 2020, each of my lectures in the seminar room consisted of some of the following parts:

- Classic lectures where I go through a number of slides to teach a curriculum.
- General feedback on a hand-in.
- Answers to questions from the videos.
- Live coding where the students see me write a program live in the seminar room.

I will focus here on the classical lecture and live coding. In my lectures, the "classic lecture" part took up a majority of the time. However, it frustrated me that I would often end up repeating the points that had already been taught in the video lectures. Some students also pointed this redundancy out to me in their feedback. On the other hand, I enjoyed particularly the live coding part because this gave the students a chance to see the concepts and theory that they had learnt about in the videos being applied to construct a program. Several students also told me that they learnt a lot from seeing a program being constructed.

One can also ask the question: What value can the lectures bring which videos cannot? With a traditional view of lectures where the active lecturer delivers knowledge to the passive students like a jug pouring water into a mug, the answer would be “Not much”. However, if we abandon this view, we see opportunities. When the students and teacher are in the same room it gives opportunities for active learning, immediate feedback and interaction.

With this in mind I decided to conduct an experiment in which my lectures would largely abandon the classical presentation of slides and instead expand the live coding. Additionally, I would try to have the students be active during the lecture and give them immediate feedback. One can discuss whether the word “lecture” can describe this approach, but for the lack of a better word I will keep using it for the remainder of this paper.

The rest of the paper is organized as follows: Section 2 states the problem considered. Section 3 presents the related work about live coding that the present paper builds on. Section 4 presents the concept of the experiment. Section 5 explains which pedagogical theories the experiment is based on. Section 6 explains how I prepared, presented, executed and continuously adapted the experiment. Section 7 reflects on the experiment. Section 8 presents my ideas for further experimentation. Section 9 discusses the feedback on the experiment that I got from the students. Lastly, Section 10 draws a conclusion on the experiment.

2 Pedagogical problem

Based on the above motivation I have formulated the following problem: “In the context of the course IMPR at AAU CPH, how can I teach programming using live coding and active learning?” I addressed the problem by experimentation in IMPR during the fall of 2021. In this paper I explain the experiment and reflect on how it went, and how I could go further with it.

3 Live coding

I am not the first to teach programming using live coding, and I draw inspiration for my experiment from many sources [5, 20, 15, 3, 12, 2], but in particular from Nederbragt, Harris, Hill and Wilson [9], Raj, Patel, Halverson and Halverson [13] and an example from the digital education platform Future Learn [7].

Nederbragt, Harris, Hill and Wilson [9] list ten quick tips for live coding. I list the tips here in italic, and explain the ones that are not self-explanatory:

1. *Go slowly.*
2. *Mirror your learner’s environment.* The setup that you use on the computer should be as similar as possible to that of the students.
3. *Be seen and heard.* You should not sit down behind a screen and hide. It is better to stand up and be seen and heard.
4. *Use the screen(s) wisely.* You should make sure that what you put on the screen can actually be seen, and therefore you should ensure that the font is big enough and the contrast strong enough e.g.
5. *Avoid distractions.* A main point here is to close programs that might give disturbing notifications during the lecture.
6. *Use illustrations – Even better, draw them.*
7. *Stick to the lesson material.*

8. *Embrace your mistakes.* When (not if) you make mistakes while live coding, you should explain the mistake to the students. In this way they will actually learn what to do when they end up in the same situation instead of just being confused.
9. *Get real-time feedback and provide immediate help.* The idea here is to let the students give real time feedback on whether they manage to follow along. Another point is to have a good ratio between students, teaching assistants and teachers.
10. *Turn learners into co-instructors.* Let the students be part of the process of writing the program by letting them give suggestions. You should also give the students a chance to discuss and put what they have learnt into words.

The authors also provide links to a video of a real world example of live coding [21], a (constructed) example of a teacher breaking most of the rules above [10], and an example of the same teacher following these rules [11].

Raj, Patel, Halverson and Halverson [13] explain the role of live coding in introductory programming. They point out a number of advantages:

- *Incremental coding.* Programmers do not sit and write a whole program before they run it. Rather they will change back and forth between writing the program and trying it out. Thus, the program is built incrementally. In slide presentations this will not be clear to the students because they will just see a perfect program "magically" appear on a slide.
- *Breaking down the problem.* Incremental coding can be done by breaking down the problem into smaller parts that can be solved individually. Live coding can illustrate this to the students.
- *Making thinking visible.* With live coding, the teacher has the chance to make thinking visible by expressing what thoughts go into building the program.
- *Modelling, Scaffolding and Fading.* Live coding presents an opportunity to do Modelling, Scaffolding and Fading. The authors see the modelling stage as the part where the teacher writes a program. They see the scaffolding stage as the part where the students have to solve an exercise based on what was learnt. They see the fading stage as when the students have to do home-work exercises without scaffolds.

The digital education platform Future Learn gives a small example of how one could do live coding to teach lists [7]. They give examples of how one can run the program often, ask questions to students, illustrate concepts and talk about errors in the program. I have also drawn inspiration from this example.

4 Concept

I present here my concept for the reorganized lectures of the fall of 2021:

- Introduction with slides
- 1st half
  - Live coding by Anders
  - Exercise in pairs for the students
  - Discussion of a solution
- 2nd half
  - Live coding by Anders
  - Exercise in pairs for the students
  - Discussion of a solution
- Small presentation with slides

The total time for this is ∼2 hours. I gave the students 15 minutes to solve the exercises in pairs. After the lecture follows ∼2 hours of group exercises in the same way as in 2020.

5 Pedagogical theory

I will now explain the pedagogical theory behind the experiment. A recent survey by Rodrigues, Monteiro and Osório [14] identifies methodologies for teaching programming. Of these my experiment draws on active learning and peer programming, but also takes inspiration from the flipped classroom (via video lectures) and is situated in an educational program oriented towards Problem-Based Learning. It also relies on teaching in the lecture room – not as traditional classes with a slide deck, but rather by applying the learnings from the aforementioned papers on live coding and a number of other didactic tools explained below.

One can see teaching as facilitation of learning rather than mere delivery of knowledge. As written in the beginning, I see live coding with active learning as a way to do this. Of course, there is still knowledge that has to be delivered such as "What is a type?" and "How does an if-statement work?" but using live coding with active learning I explore a different way to facilitate the students' learning of this knowledge than delivering it through a presentation.

Another pedagogical theory is that of scaffolding [22, 19], and as Raj, Patel, Halverson and Halverson [13] explained, live coding presents an opportunity to apply this principle. In the exercises in pairs the students have to program something that is similar to what they have just seen me program. Thus, the program I have just written can be seen as a scaffolding for the exercise. However, in contrast to Raj, Patel, Halverson and Halverson's way to do live coding [13] my concept fades the scaffolds away in several stages rather than "just" in a set of home exercises. When the students arrive at the group exercises there will be less of a scaffold – these exercises are typically more challenging and are not preceded by a very similar example. There is still a scaffold however, namely the course material and the availability of the teacher and teaching assistant. Additionally due to the Aalborg PBL (Problem-Based Learning) model [16, 17, 18, 6, 1] used on the education program, the students will also use what they have learnt in their semester project. The 10 ECTS semester project module on the first semester consists of the students finding and delimiting a problem and programming a solution to it as a program in C. The project is done in groups of size approximately 7. Each group has a supervisor who is either a teacher employed by the university or hired externally. The problems that the students face in their semester projects are less structured than exercises, and the only scaffolding is the students' supervision meetings.

A last pedagogical tool I will consider is "Think, pair, share" [8] in which lectures are broken up by pair exercises in which the students are given a question to think about, then pair up to talk about and lastly to share with the other participants. The methodology is generic and so I had to adapt it to the specific topic of live coding.

6 The experiment

From the third week of the semester and forward, I conducted my experiment. For each week this consisted of preparing and executing that week's live coding and exercises. In this section of the paper, I first explain my experiment as conducted in the third week and then I will explain how the experiment continued during the following weeks. In the following section, I will then give my reflections on the experiment.

6.1 Preparation

The third lecture's topic was logical expressions, if statements and switch statements. In my preparation I took a look at the learning objectives for that lecture and chose which ones of them I wanted to cover. I decided to cover logical expressions and if statements, but to leave out switch statements from the live coding. My reasoning was that it would simply be too much to also cover the switch statements and thus this part of the learning objectives would instead be covered "only" by the book, the videos and the exercises. Hereafter, I came up with 2 programs that illustrated respectively logical expressions and if statements. I then tried to write these programs as a practice. I also wrote a manuscript that contained the "plan" for writing the programs, i.e., how I would break it down into smaller parts and also suggestions for questions I could ask the students. Lastly, I made a minimalistic set of slides.

6.2 Presenting the experiment

In the "Introduction with slides" for this first run of the experiment, I presented and motivated the experiment to the students. I motivated live coding as a modern way to teach programming with the following advantages:

- The students see not only programs but also the process of constructing them.
- The students learn the thought process used to construct a program.
- The students see programs with mistakes and how these are repaired.

Furthermore, I argued that one of the learning objectives of the course is to program with the learnt knowledge, and that live coding could be a way to facilitate the students in achieving this objective.

After this presentation of the experiment, I asked the students if they were up for trying this and they agreed (by nodding and/or saying yes).

6.3 Execution

During the live coding I used large font sizes and also streamed the screen to the students' computers using Microsoft Teams. I did the streaming without sound, because the concept is that this is something we do in the seminar room. The motivation for streaming the screen was that everyone should be able to see what I code no matter where they sit in the room.

As shown in section 4, the concept of the lectures consists of two largely identical "halves" and therefore I will only explain the first of these. The first "Live coding by Anders" consisted of me programming a rock-paper-scissor game. During the development I asked the students questions and they also asked questions for me. The questions I asked were rather open, namely collecting suggestions on how we could split the program into smaller parts that we could then solve. Several students came up with suggestions.

In the first exercise in pairs, the students had to write an expression characterizing a draw in the game. During the 15 minutes where the students were working on this, I replied to their questions when they asked. After the exercise in pairs, I asked if anyone wanted to share a solution by uploading it to Microsoft Teams. Several pairs raised a hand, and I picked one. When they had uploaded it, I walked through it so that the rest of the students would also understand it. Then I pointed to a number of things that could be improved and showed how this could be done.

At the end of the lecture, I did an evaluation using an online form.

6.4 Continuing the experiment

In the following weeks I continued and adapted the experiment. I made a number of changes such as:

- adjusting the difficulty of the exercises in pairs
- using the black board and slides for illustrations
- doing mistakes in the programming on purpose
- asking more closed questions compared to open questions.

The changes were based on reflections that I will discuss in the following section.

7 Reflection on the experiment

I will now give my reflections on the experiment.

7.1 The students' reaction

The students understood the motivation for the experiment and were ready to "play along". Several students were also happy to give suggestions to my questions of how we could write the program. The students asked questions when there was something they did not understand, and they also remarked when they saw me make mistakes in my programming. During the exercises in pairs the students worked on them actively. The feedback I got from my survey was positive. It seemed that I had hit a good level of difficulty for the exercises in pairs, and feedback for the live coding was positive.

7.2 Bonus exercises in pairs

When preparing the exercises in pairs, I had to consider the difficulty. Since the exercises have a length of 15 minutes, they have to be quite easy to be possible to finish within this time limit. I asked the students for feedback on the difficulty of the exercises. The feedback indicated that I managed to hit a spot where most student thought the difficulty level was just right, some thought it was too easy and some thought it was too hard. I also got personal feedback from some students that it was too easy for them. A challenge when teaching introductory programming is that students do not start with the same experience. There are students who meet up with no experience in programming, while others have experience from either hobby programming or from having computer science as a subject in high school. In order to also give these students a challenge I decided to introduce also bonus exercises. The idea was to include an optional objective for each of the exercises. My experience was, as hoped, that some students only did the normal exercises while others could also manage to do the bonus exercises. The only disadvantage I see with having bonus exercises was that when we discuss the solutions together, students would often submit solutions that solved also the bonus exercises, and perhaps this could confuse the ones who did only the normal ones.

7.3 Difficulty

Another observation was of a subset of the students that had followed along with my live coding, completed the exercises in pairs during the lecture, but then at the group exercises really struggled to apply what they had been taught. At the group exercises I got the impression that they had managed to solve the exercises in pairs largely by copying what I had done and adapting it, but without really getting an understanding of what was going on.

I therefore wanted to change my exercises in pairs to avoid this problem. For this reason, I increased the difficulty of the future exercises in pairs. I made sure that each exercise had a small "twist" on what I presented in the live coding. E.g., in the lecture about structs, my live-coding example had structs where all values of fields were given as user input, whereas for the exercises in pairs, I required some of them to be calculated.

7.4 Keeping time

A challenge is to keep the time plan for each lecture. I set off 15 minutes for the students to finish the exercises in pairs and 5 for us discussing a solution. However, during the 15 minutes I would sometimes get a question in the last minute, and a dilemma was whether to help the student or to continue the lecture. If we were ahead of schedule and my impression was that most of the other students could use more time to work on the exercise I would just go ahead and help the student. In case we were not on schedule I would have to tell the student that I did not have time to help.

Another dilemma is that keeping the time can sometimes clash with Nederbragt, Harris, Hill and Wilson's [9] tip of going slowly. Sometimes I simply did not have the time to go as slowly as I would have liked. Of course, I could have extended the lecture at expense of the group exercises, but this also has drawbacks.

7.5 Learning objectives

Related to the dilemmas of keeping time is the question of how many learning objectives to cover during a lecture. In the classical lectures with slides that I did in 2020 I could cover a lot! With slides I could introduce new concepts, example programs etc. at a high pace at the click of a button or the blink of an eye. However, if the students see many things at the blink of an eye how much do they really learn? Live coding forced me to cut down a lot on how many learning objectives I could cover in each lecture. The consequence was that I covered less, but I did it more thoroughly.

This was not without consequences though. As mentioned for the lecture on logical expressions, if statements and switch statements, I decided to not cover switch statements in the live coding. As a consequence, some students were confused in the lecture on loops, when I used a switch statement. Because of this experience, I decided to start every lecture by listing to the students which part of the week's learning objectives were covered by the live coding, and which parts I expected them to learn from the videos, book and group exercises.

7.6 Sticking to the plan

Another challenge was when the students answered my open questions about how the program could be split into parts. The challenge was that they sometimes came up with a suggestion that was good, but that I had not prepared. Then I was in the dilemma of deciding between following what I had planned or the suggestion. I decided to compliment the student for their idea but to still follow my plan. Even if a student's idea is good from a programming perspective, it might not illustrate certain parts of the language that the students have to learn. In these cases, my choice to carry on with my plan seemed to me to be the right one. In another case however, I decided to carry on where it might have been better to follow the student's idea because in that particular case there would have been no loss with respect to what had to be learnt, and the student's idea was actually better than mine.

7.7 Mistakes

Based on Nederbragt, Harris, Hill and Wilson's [9] tip of embracing mistakes, I did exactly that. Whenever there was a mistake, I would explain to the students what the mistake was and how we could fix it. The only downside of doing this is that it can take time to do, but I would say that it is worth it. I also experimented with doing mistakes on purpose. In a lecture on random numbers, I tried to make a program where I "forgot" to call the srand function on purpose. This meant that students would not only see a slide telling them to remember this function, but that they also saw in practice what happened if they forgot it. At the group exercises I was happy to see many students remember to call that function, but there were also students who still forgot it. I think this shows that for some students it is not enough to be told about the problem and to see someone else experience it. They need to experience it themselves. Therefore I would say that this shows the value of the group exercises.

7.8 Illustrating problematic programs

In 2020 I included in my lectures a part where I would, with consent, discuss during the lecture some, anonymous, program that the students had handed in. I tried to pick a program that was not perfect, and then I showed the students what could be improved. In the new concept I instead discussed the programs that the students did during the lecture. A problem here was that often only the students with a strong grip of the material would share their solution, and there would often not be much to say about these, since they were close to perfect. In the later lectures, however, some students were brave enough to volunteer with less perfect programs. These were more interesting to discuss because there was more to say about them. I will also point out that when I received these kinds of programs, I made sure to compliment them – in all cases the students were on the right track and I believe that the students should be rewarded for sharing a solution like this, no matter what state it is in.

7.9 Students as co-instructors

Nederbragt, Harris, Hill and Wilson's [9] give the tip of having students be co-instructors. In my lectures I saw this happening especially during the exercises in pairs. When the students work in pairs, students who understand the topic can help clear things up for those who don't, and additionally they get a chance to put their knowledge into words. From a logistical point of view, it also makes a lot of sense. In the lectures I am one teacher for ∼50 students. In the group exercises we are one teacher and one teaching assistant for ∼50 students. If I can facilitate the course such that students help each other, then the ratio is much better. I saw this happening during the exercises in pairs. However, there were also some students who preferred to work alone here, even though that was not the intention. In some of the lectures I tried talking to them about it during the exercises in pairs. Some then decided to find someone to work with, while others decided to keep working alone. Among those who decided to find someone to work with, I then saw them in the next lecture working alone again. But if this can help some of them to work in pairs then it is probably worth it.

7.10 Scalability

The following year, 2022, I taught the course again, but I was allowed to teach the course, in the same time slot, for also another educational program with ∼40 students. Thus ∼95 students started on the course that year. Because of this I was allowed to have two student teaching assistants for the group exercises instead of only one. My experience was that the course scaled to the larger size without problems. In particular, the active learning part of the course gives all students the chance to be active during the lecture rather than only the ones who are brave enough to ask during the lecture or to answer one of my questions to the audience.

8 Possibilities for further experimentation

I see a number of possibilities for further experimentation with teaching introductory programming using live coding and active learning. I will explain and reflect on them here.

8.1 Variations on the exercises in pairs

In the exercises in pairs, I used a concept where the students work in pairs for 15 minutes. One can easily try out variations where they work for longer or shorter time in bigger or smaller groups. Another interesting idea could be to have the pairs meet other pairs to discuss their solutions. They could then look at pros and cons or to perhaps build a solution consisting of the best ideas of both solutions. The challenge with these opportunities is that one has to keep in mind both the time that they cost at expense of other activities and to which extent the facilities in the seminar rooms allow them to be possible. If the students sit on rows, then how can two pairs realistically meet up? In a seminar room that could perhaps work by having half of the pairs turn their chair, but in an auditorium it would not work.

8.2 Extending live coding with active learning

In the 2021 iteration of the course, I did 2 hours of live coding with active learning followed by 2 hours of group exercises. An idea could be to extend the live coding to cover the full 4 hours. A challenge when doing live coding is that it takes time and I struggle to cover several learning objectives; more time could make it possible to cover more. I have also experienced that some few students leave after the live coding, and this could be a way to circumvent that problem. I do also see a number of negative aspects of this idea. Firstly, it removes the amount of time that is spent in the project groups and thus time where the group is bonded together. I think this would be a shame. Of course, one could have the students work in their groups during the lecture instead of the pairs, but unfortunately the facilities in the seminar rooms are not fit for that. Secondly, if we see the group exercises as a part of the course where the scaffolding is (partially) removed then one could also fear that abolishing this part would hinder the students' learning.

8.3 Going off script

As discussed during the reflection, I largely avoided going off script during the live coding. I would like to explore what happens if I tried changing this habit. I suspect that the more I involve the students in the live coding, the more they will be active, and thus the more they will learn. One can also ask what the point is for me to ask them what we can do next if I am not willing to follow their suggestions.

8.4 Illustrating problematic programs

As discussed during the reflection, it was more interesting to discuss a problematic program than a "perfect" one. Therefore I am wondering what opportunities there are for getting students to submit programs that are less than perfect. An idea could be to encourage it verbally during the lecture. Or a way could be to have all pairs submit a solution anonymously, and then I could look at them during a pause and pick out an interesting one to discuss. In this way it would not require bravery on behalf of the students to submit a solution.

9 Feedback from the students

In several of my lectures I asked the students for feedback on the lecture they had just experienced. I asked for quantitative feedback on the difficulty of the exercises in pairs and qualitative feedback on the lecture as a whole. In each of these I got replies from between 19 and 34 students. The general picture on the difficulty of the exercises in pairs was that I hit a good difficulty – there was a good spread between the students finding it easy, in between and difficult. In Appendix A, I list some of the concrete thing that students liked about the live coding as well as some things they thought could be improved.

In addition, I also asked for feedback on the course as a whole in the last lecture. Here I got 33 replies. In the quantitative feedback, the students answered that they learnt a lot, were encouraged to participate actively, were motivated by the activities and liked the course. They also claimed to get a lot out of my (limited) presentation of slides, the live coding and the exercises during the lecture. All replies except 2 claimed to get a lot out of my feedback on a pair's solution to an exercise during the lectures. In Appendix B, I list some of the concrete thing that students liked about the live coding as well as some things they thought could be improved. In Appendix C, I have put bar charts for the quantitative feedback.

All in all, the feedback on this experiment from the students has been very positive – much, much more than I would have hoped for. When running the course again in the fall of 2022 for a larger audience the feedback was again very positive.

10 Conclusion

In the fall of 2021, I taught programming using live coding with active learning. I see this as an improvement over my previous approach where the emphasis was on teaching with slides. However live coding with active learning presented a number of challenges with respect to e.g. finding the right difficulty for the exercises, keeping time, coverage of learning objectives and illustrating problems in programs. The experiment was well received by the students. I see many opportunities for further experiments on the concept. It is my hope that teachers in programming can draw inspiration for their own teaching from my experiment, reflections and experiences.

References

1 Peter Dolog, Lone Leth Thomsen, and Bent Thomsen. Assessing problem-based learning in a software engineering curriculum using Bloom's taxonomy and the IEEE software engineering body of knowledge. *ACM Transactions on Computing Education (TOCE)*, 16(3):1–41, 2016.

2 Alessio Gaspar and Sarah Langevin. Active learning in introductory programming courses through student-led "live coding" and test-driven pair programming. In *International Conference on Education and Information Systems, Technologies and Applications, Orlando, FL*, 2007.

3 Alessio Gaspar and Sarah Langevin. Restoring "coding with intention" in introductory programming courses. In *Proceedings of the 8th ACM SIGITE conference on Information technology education*, pages 91–98, 2007.

4 Jeri R. Hanly and Elliot B. Koffman. *Problem solving and program design in C (8th edition, global edition).* Pearson, 2015.

5 Luke W Johnston, Madeleine Bonsma-Fisher, Joel Ostblom, Ahmed R Hasan, James S Santangelo, Lindsay Coome, Lina Tran, Elliott Sales de Andrade, and Sara Mahallati. A graduate student-led participatory live-coding quantitative methods course in R: Experiences on initiating, developing, and teaching. *Journal of Open Source Education*, 2(16):49, 2019.

6 Anette Kolmos, Flemming K Fink, and Lone Krogh. *The Aalborg PBL model: progress, diversity and challenges*. Aalborg University Press Aalborg, 2004.

7 Future Learn. Live coding. `https://www.futurelearn.com/info/courses/secondary-programming-pedagogy/0/steps/68423`, 2020. Accessed: 5. December 2021.

8 Frank T. Lyman. The responsive classroom discussion: The inclusion of all students. *Mainstreaming digest*, 109:113, 1981.

9 Alexander Nederbragt, Rayna Michelle Harris, Alison Presmanes Hill, and Greg Wilson. Ten quick tips for teaching with participatory live coding. *PLOS Computational Biology*, 16(9):1–7, September 2020. `doi:10.1371/journal.pcbi.1008090`.

10 Lex Nederbragt. A video introduction to live coding part 1. `https://www.youtube.com/watch?v=bXxBeNkKmJE`, May 2016. Accessed: 3. December 2021.

11 Lex Nederbragt. A video introduction to live coding part 2. `https://www.youtube.com/watch?v=SkPmwe_WjeY`, May 2016. Accessed: 3. December 2021.

12 John Paxton. Live programming as a lecture technique. *J. Comput. Sci. Coll.*, 18(2):51–56, December 2002.

13 Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson, and Erica Rosenfeld Halverson. Role of live-coding in learning introductory programming. In Mike Joy and Petri Ihantola, editors, *Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 22-25, 2018*, pages 13:1–13:8. ACM, 2018.

14 Gabryella Rodrigues, Ana Francisca Monteiro, and António Osório. Introductory programming in higher education: A systematic literature review. In Alberto Simões and João Carlos Silva, editors, *Third International Computer Programming Education Conference, ICPEC 2022, June 2-3, 2022, Polytechnic Institute of Cávado and Ave (IPCA), Barcelos, Portugal*, volume 102 of *OASIcs*, pages 4:1–4:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/OASIcs.ICPEC.2022.4`.

15 Marc J Rubin. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 651–656, 2013.

16 John R. Savery. Overview of problem-based learning: Definitions and distinctions. *Essential readings in problem-based learning: Exploring and extending the legacy of Howard S. Barrows*, 9(2):5–15, 2015.

17 Maggi Savin-Baden. What are problem-based pedagogies? *Journal of Problem-Based Learning*, 2020.

18 Virginie Servant-Miklos. Problem-oriented project work and problem-based learning: "Mind the gap!". *Interdisciplinary Journal of Problem-Based Learning*, 14(1), 2020.

19 Rob Wass and Clinton Golding. Sharpening a tool for teaching: the zone of proximal development. *Teaching in Higher Education*, 19(6):671–684, 2014.

20 Greg Wilson. Software Carpentry: lessons learned. *F1000Research*, 3, 2014.

21 Greg Wilson. Software Carpentry teaching demonstration. `https://vimeo.com/139316669`, 2015. Accessed: 3. December 2021.

22 David Wood, Jerome S. Bruner, and Gail Ross. The role of tutoring in problem solving. *Child Psychology & Psychiatry & Allied Disciplines*, 1976.

A Qualitative feedback from feedback form at the end of a number of lectures

Here is my summary of some concrete things that students mentioned they liked about the approach:

- Following along and understanding the live coding without prior coding experience.
- Live coding and active learning can be seen as "programming by practice".
- Getting a chance to experiment with the code.
- Working in pairs gave a chance to learn from someone with more experience.
- The exercises in pairs as a way to be included in what is happening.

Here is my summary of some concrete suggestions for improvements:

- The live coding could be more detailed.
- One of the early lectures was quite easy.
- The tempo can be too high, which can encourage just copying what the lecturer was typing without understanding it.
- One could more often involve students in coming up with what to do next when live coding.
- Show an animation of what was happening *while* the lecturer was coding instead of doing it afterwards.

B Qualitative feedback from feedback form at the end of the course

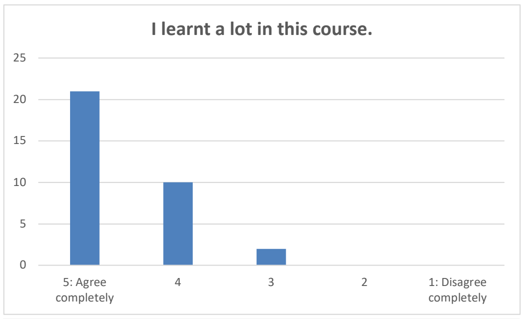
In the qualitative feedback, the students were positive about the live coding. Here is my summary of some concrete things that students liked about the approach:

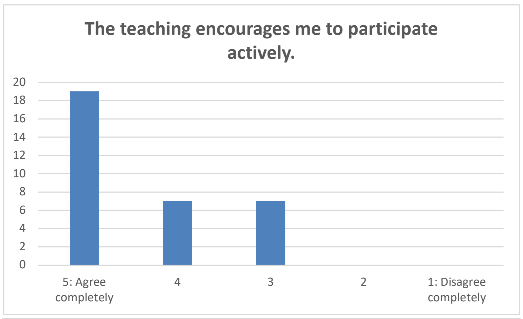
- The live coding involves the students.
- The live coding shows the students how to build a program from scratch and not only the theory of programming.
- The live coding is engaging.
- The live coding made the content clearer and easier to implement.
- The live coding showed how to use the content in practice.

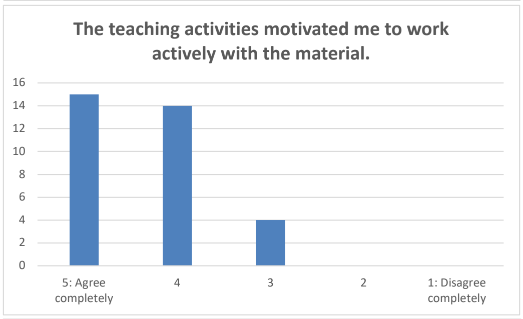
Here is my summary of the suggestions of what students thought could be better in the live coding:

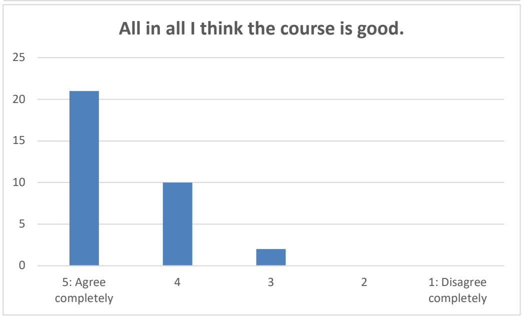
- The lecturer could explain more of the thoughts that go into the programming.
- The tempo during the live coding can be too high for them to type along.
- The tempo during the live coding can sometimes be too high for students to understand what is going on.

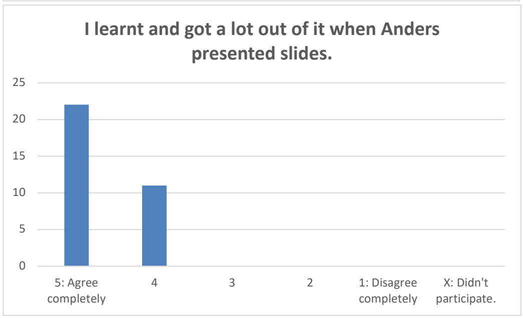
C Quantitative feedback from feedback form at the end of the course

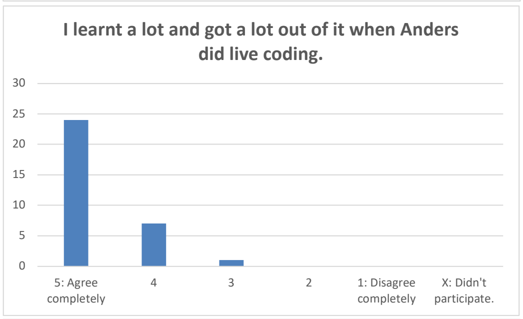

I learnt a lot in this course.
5: Agree completely
1: Disagree completely

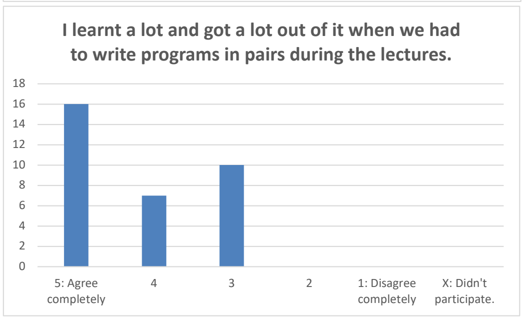

The teaching encourages me to participate actively.
5: Agree completely
1: Disagree completely

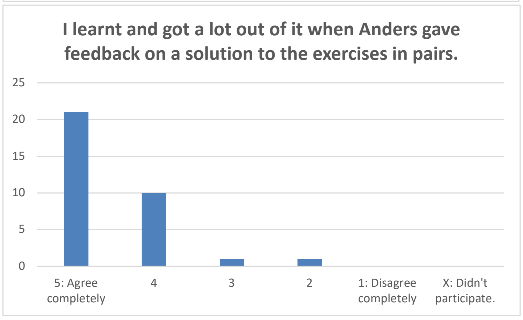

The teaching activities motivated me to work actively with the material.
5: Agree completely
1: Disagree completely


All in all I think the course is good.
5: Agree completely
1: Disagree completely


I learnt and got a lot out of it when Anders presented slides.
5: Agree completely
1: Disagree completely
X: Didn't participate.


I learnt a lot and got a lot out of it when Anders did live coding.
5: Agree completely
1: Disagree completely
X: Didn't participate.


I learnt a lot and got a lot out of it when we had to write programs in pairs during the lectures.
5: Agree completely
1: Disagree completely
X: Didn't participate.


I learnt and got a lot out of it when Anders gave feedback on a solution to the exercises in pairs.
5: Agree completely
1: Disagree completely
X: Didn't participate.