Social dimensions in the lab session when novices learn to program

1st Kristina von Hausswolff
Department of Information Technology
Uppsala University
Uppsala, Sweden
kristina.von.hausswolff@it.uu.se

2nd Maria Weurlander
Department of Education, Stockholm University
Department of Learning in Engineering Sciences, KTH
Stockholm, Sweden
maria.weurlander@edu.su.se

***Abstract*—This full research paper reports on a study of social dimensions when students learn to program. The aim of this study is to investigate students' experiences of social dimensions in learning to program as novices in a pair-programming lab setting. Data was collected by means of individual interviews with seven students mid-way through the course. A questionnaire to 77 students gave a background of the class as a whole and was used to select students for the interviews. The interview data were analyzed using an inductive content analysis method and interpreted theoretically from a pragmatist and transactional perspective on learning. Our results show different ways that the social dynamic between the students in a pair affected 1) the emotions experienced, 2) the extent to which students actively wrote code and interacted with the computer environment, and 3) how students perceived their competence. Interviewed students report that failure and success in a programming task result in an emotional roller coaster, and that in this turbulence, the social context is of utmost importance and weighs in when students consider if they are going to pursue a programming profession.**

***Index Terms*—novice programming, pair programming, practice, pragmatism, social dimensions**

I. Introduction

Computer programming is being introduced in curricula in many countries including Sweden. It has been argued that programming and coding skills are important in our increasingly digitalized society [1]. Many undergraduate education programs nowadays include a course in introductory programming for students majoring in other subjects. However, many students taking these courses are novices with limited experience of programming. It is also well known that many students find learning programming difficult. Robins [2, p. 327] writes that "programming languages are complex artificial constructs. Like the grammatical rules of natural language, they consist of a relatively small number of elements that can be combined in infinitely many productive ways" and "it is tempting to assume that programming is simply harder than most other topics to learn" [2, p. 332].

Educators and researchers have developed and evaluated various ways to improve teaching and learning in computer science (CS). There is an agreement amongst CS educators that students need practical work, hands-on at the computer, in order to learn the skills and understand the key concepts [3]–[5]. A recent review argues for the need of more in-depth research exploring novice students' experiences of learning programming in the context of introductory CS courses [5], commonly referred to as CS1.

We investigated social dimensions in the lab situation when CS non-major students encounter programming for the first time. Students working in pairs during the lab sessions is a common and important scenario in computer science education. A common learning goal of a CS1 course is that students are able to solve programming tasks by themselves, writing code. This learning process is situated in a social context with peers, teaching assistants, and teachers. By social dimensions we mean the interplay between students, students and the educational environment, and students' prior experiences and beliefs about themselves as learners.

Contextual aspects, including social dimensions, have a strong influence on learning, and in order to improve education we need to understand these aspects. The aim of this study is to better understand how social dimensions come in to play when students encounter programming as novices in a pair programming lab setting. To that end we have also used pragmatism as theoretical framework.

A. Research context

The context in which this study was conducted is an introductory programming course in a five-year engineering program at a Swedish university. There were no requirements or expectations of previous knowledge in programming in order to take this course. The main programming language used in this course was Python. The students who participated in the course did not have computer science (CS) as a major subject. The course (8 credits) was mandatory for these students, and ran part time during 18 weeks, in parallel to one or two other mandatory courses during their second semester. Although these students are CS non-major students in the first year, they can choose a specialization trajectory during the second year that would make them CS majors.

The content of the course was typical CS1 content: basic programming structures such as selections, loops, functions, and managing lists and files. The course also included object-oriented concepts such as class and inheritance. The Python part of the course consisted of fifteen lectures, six mandatory computer labs, and exercises without computers. The class of 40 to 50 students was divided into lab groups of about 10

students, each assigned a teaching assistant (TA). The TAs were more senior students or PhD students with a programming background. After eight weeks, the students had a written exam. For the remaining part of the course, students worked on a larger individual assignment. The assignment was chosen from a large pool of assignments and all of the students in the same lab group had different assignments. Although the assignment required individual work, students were allowed to discuss them and help each other. Students also had to attend computer lab sessions to show in-progress results to the TA, who could provide help if needed. Each student had to hand in their source code, and also run their program and answer questions in person to pass the examination.

The study reported here is part of a larger research project with the aim to explore novice students' learning programming and the importance of practical hands-on experiences [6]. The initial six lab sessions in the introductory course described above were the focus in this study, since we were interested in the social dimensions of collaboration and "practical" part of learning to program. During the computer labs, the students worked in groups of two, a pair programming inspired setting [7]. In pair programming two persons work together to solve a problem at a computer. They alternate between being the "driver", who types the code, and the "navigator", who is responsible for detecting errors and helping to solve the problem. Each student pair collaborated and discussed the problem at hand, while only one student at a time interacted with the computer. The students were instructed to shift the keyboard holder every 20 minutes. Although such educational situations are always unique—a specific content, a specific language, a specific teacher—similar pair-programming situations are common in university and upper secondary schools. The instruction to work in pairs made it possible for us to explore the students' experiences of both writing code (being the driver) and sitting beside the lab partner (navigating).

II. Background

A. Novices learning to program

There is a substantial body of research demonstrating that novices run into a wide range of difficulties when learning programming, and it is generally considered hard to learn to program [2]. Despite great efforts from educators and researchers the problems seem to prevail. Experiences of novices in introductory programming courses may affect their attitudes toward the subject negatively. Novice students have been found to experience that their skills for learning CS were insufficient, and they tended to use "ask for help" problem solving strategies [8]. This was in contrast to more experienced students who felt they had sufficient skills to learn CS, used more exploratory problem solving strategies and considered programming as another type of interaction with the computer. A study by Peters and Pears [9] found that students' experiences in a CS1 course led to a mainly positive change in attitudes of computer science. However, some students hold the view that "since they do not want to become programmers, they do not need to learn more CS than the basics in programming that they learned in the course, and hence they will not take more courses in CS." [9, p. 5]. These findings [8], [9] point to an interesting aspect in novice students' experiences: the way they identify themselves in relation to CS and programming. Moreover, in a recent review Luxton-Reilly et al. [5] conclude that "Although introductory programming courses are considered to cause significant levels of student anxiety and frustration, and student engagement levels in computing are benchmarked as among the lowest of any discipline [...], there are very few publications focusing on the student experience, particularly with respect to pastoral care." [5, p. 87]. The purpose of the present study is to shed more light on student engagement by exploring social aspects of learning programming.

B. Pair programming in education

Pair programming is a common collaborative learning method in computer science education. A recent literature review concludes that pair programming is beneficial for students' learning [10]. Students working in pairs seem to pass exams to a greater extent and they also seem to be more confident with their work compared to students who work alone. Moreover, it seems that pair programming may be particularly beneficial for weaker students, perhaps due to a higher attendance in the lab sessions [11].

Social aspects also come into play when students work in pairs. Students seem to enjoy pair programming more than working alone [10]. Working together in pairs also gives students teamwork experience, which is important since the ability to work well with others is a required social skill in many workplaces. However, there were also findings suggesting that "higher skilled students sometimes became frustrated when things were going wrong" [10, p. 144]. On the other hand, some students find pair programming to reduce frustration and increase their confidence [12], suggesting that students differ in how they perceive the usefulness of pair programming. When the students in a pair had significant differences in knowledge and programming skills, this could negatively influence the teamwork and the perception of pair programming [13].

Furthermore, a study by Bowman et al. [14] found that students with a more experienced partner thought that their partner did more of the work and they themselves less. This may be due to the fact that less experienced students let their partners take the lead, or that the more experienced students took the driver role. Pair programming also involves how students treat each other when they collaborate. The collaboration pattern within a pair can be characterized by equity or inequity, where in the latter, one student dominates the work perhaps in order to finish the task quickly [15]. When both students in a pair are engaged in the dialogue and make substantive contributions to producing code, solve problems, express uncertainty, and resolve it, the collaboration is more effective [16].

III. Theory

A. Pragmatism, intersubjective knowledge, and a transactional theory of learning

Our theoretical standpoint is a pragmatic view of knowledge and learning. As described by Biesta and Burbles [17], a pragmatic view sees learning a process of inquiry and builds on an ontology of transactional realism and an intersubjective theory of knowledge. Actions together with already established knowledge inform the learner, and intersubjective knowledge is created through communication. Dewey's transaction is "the point of contact" between an environment and an organism. Reality exists but is only perceived as a function of the transaction. Transactions create experiences, and knowledge becomes a form of experience with which we approach the world. The organism tries to establish a balance with the environment and develops patterns of possible actions, habits. Habits in combination with communication is how knowledge is gained, according to Dewey. He denotes this process *inquiry* [17, p. 58].

There is a growing body of research within several educational disciplines (subject didactics) that uses a pragmatic approach. Pragmatic philosophy and theory has been found beneficial to understand educational practices, e.g. Östman, Van Poeck & Öhman in sustainability education [18] and Wickman in science education [19]. To our knowledge, this approach is novel in computer science education.

In this pragmatic approach the purposes of education are discussed as a way of understanding different elements that go into an educational experience for a student. The three purposes, or functions, of education are *qualification*, *socialization*, and *person-formation* [20], [21]. Qualification, the most important function according to Biesta [21], is learning both knowledge and skills, for example to be a programmer, and to become a citizen in society. Socialization, is to be a part of an already established order to acquire norms and values, e.g. those that are a part of being a programmer or a Swedish citizen. Van Poeck & Östman argue that values are a part of every discipline, and that norms and values are acquired alongside the learning of knowledge and skills. This is called companion meanings or companion norms.

The third function, person-formation, is described as the cultivation of the self in an educational setting. They describe two forms this can take: *identification* and *subjectification*. Identification is how well the students feel that they "fit in" in a subject or profession, e.g. whether students can relate to role models in the field. In programming for instance, it seems that women have a harder time identifying with the programming profession which results in fewer women in the profession. Person-formation as subjectification means to mature in an independent way in the profession or in relation to the subject field. Biesta explains that subjectification is best described as the opposite of socialization, developing as a unique subject. In relation to Biesta's original functions of education, Van Poeck & Östman add identification as part of the umbrella concept person-formation.

Scholars in the pragmatic tradition have described learning as meaning-making [22]. This builds on the principle of continuity which implies purpose. Individuals are continuously in motion using habits already acquired, often neither consciously recognized nor reflected upon. When individuals are interrupted in this unreflected motion there is a reason to change direction, to focus attention on the interruption. They try to bridge the gap between what they already know to what they aim to understand by creating relations. This includes doubting, questioning, and reaching out for possible resources. In pragmatic meaning-making, there is an emphasis on the connection between cognition and bodily feeling, expressed by Von Poeck and Östman: "all our knowledge, concepts etc. have the origin in bodily feeling" [18, p. 136] referring to both James [23] and Dewey [24]. To take emotions and feelings into account when dealing with thinking and cognition makes it possible to discuss emotions and values that accompany cognitive learning. It gives the educational researcher a theory with which to connect actions and feelings to acquired knowledge and accompanied values. Johnson says "There is no cognition without emotions, even though we are often unaware of the emotional aspects of our thinking. The idea that meaning and understanding are based solely on propositional structures is problematic because it excludes (or at least hides) most of what goes into the way we make sense of our experience." [25, p. 9].

B. Hands-on and practical thinking in programming

Previous research has pointed to the beneficial effects of hands-on in programming education research [4], [26]. There seems to be a consensus among students as well as teachers that hands-on is important [3]. Not only the application of theoretical knowledge is important, the learning goes both ways—also from practice to conceptual knowledge, as shown in [27]. A spiral approach to teaching programming has been advocated for a long time [28] where learning is built up by moving between syntax programming and semantics in terms of different concepts. To learn programming hands-on by writing code has been described as *practical thinking* while programming [29], [30], using a pragmatic perspective on learning. Dewey specifically directs focus to the importance of habits when thinking, which underpins the concept of practical thinking, building upon by the Deweyan rejection of the dichotomy of practice and theory. Eckerdal [27] examines the complex relationship between doing ("practice") and reading or hearing about ("theory") in programming education and concludes that they intertwine in small loops. In a Deweyan framing this would be a sequence of inquiries linked together forming understanding in the end.

We will argue that characteristic for this type of learning is the possibility to test code ideas in the lab setting, based on hunches and not yet expressed thoughts. Which opens the possibility to think, act, and get immediate computer feedback, in small loops of inquiry. The thinking is not expressed in statements such as hypotheses in natural language but depends on small movements in the environment while carefully

observing the response. This *thinking-while* is what we call practical thinking. Novices get familiar with the computer environment by doing movements. These movements involve ideas or impulses of actions, not easily described in natural language but, when sitting by the keyboard, expressed by direct action, writing a line of code at a particular place in the code base, running it, and get a response from the programming environment. This small-loop inquiry gives rise to new ideas or impulses of action. The concept of practical thinking as movements is closely connected to understanding this new language, new environment, and actions taken by the individual. In this orientation, or learning process, all students experience feelings of confusion and incapability, but also of accomplishment and being proud. This connects to the social aspects of the learning to program which is the focus in the present paper.

IV. Method and research design

We conducted a pre-study in the spring of 2017 and a follow-up study in 2018. We collected data using a questionnaire and by interviewing students, on both occasions. The answers to the questionnaire were used for selecting students for interviews. The selection was based on their willingness to participate, their level of prior experiences of programming och differences in time spent on writing in the lab sessions. The interview questions were based on information from the questionnaire, on observations during the lab sessions, and on all other information during the course such as lab instructions, lecture notes, and handouts.

In order to understand the context in more detail, the first author collected all the teaching material, interviewed the TAs who led the lab sessions, and conducted several observations during the lab sessions. This information was used to better understand the situations that students described in the interviews.

A. Data collection and selection of participants

The questionnaire was distributed during the final pair programming lab session. In 2017, 32 of the 51 enrolled students answered (63%), and in 2018, 45 out of 51 enrolled students answered (88%).

1) Some statistics from the questionnaire: In total, 56% were men and 43% were women (one stated 'other'). 34% had prior experiences of programming (among them 46% male). 17% stated that their time at the keyboard in the lab was less than half on average. Students that stated that they had the keyboard less than 50% of the lab session where mostly novices. 36% of all the students said they used the keyboard more than half of the time, while 47% stated about half of the time.

2) Selection of participants for interview: Because of the research focus on novices, only students that never had programmed prior to taking the introductory course were selected for an interview. But due to practical reasons, we decided to include two students with minor experiences with programming, see Table I. The interviews were conducted 2-4 weeks after the questionnaire was collected, when the students were in the middle of their larger assignment but before they handed it in. All interviews were semi-structured and the questions focused on how students experienced learning to program in the lab setting, with a special focus on hands-on aspects of learning. Because of the instructions to pair program, all students had experienced both working hands-on with the keyboard, and sitting beside without directly interacting with the computer.

B. Method of the analysis of the interviews

The seven interviews were analyzed using a content analysis method in two steps [31]. The first step was to analyze the manifest content and the second step was to use pragmatism as a theoretical framework to interpret and deepen the understanding of the content. In this paper, the focus is on three related themes in particular, which all relate to social dimensions of learning to program.

1) Step one: The interviews were analyzed inductively focusing on the manifest content of the transcripts [31], [32]. The transcribed interviews were coded in N'Vivo and the same quotes could be coded with several different codes. The first unit of analysis was sentences in the transcripts that included a reoccurring word. Some words were expected, such as "learning to program", depending on the questions asked. But some seemed to reappear without a cue, such as "frustration" and "pride". After the initial coding, the codes were examined for overlap and some refinement was performed. A structure emerged when examining the quotes and the codes several times, thereby dividing the codes in three different groups.

The first group of codes described an emerging relation between the student and the computer/program environment, evolving during the course (denoted the student-computer environment relation group). The second group contained experiences of programming that involved the whole setting, the social dimensions in the programming experience that included experiences of belonging as well as thoughts on continuing to program as a part of the education (denoted the social perspective group). Between those groups were experiences that emerged from a person's interaction with the computer environment, that proceeded to establish a relationship to programming that involved relationship to the broader setting. This third group (denoted an in-between group), made important connections between the first two groups. It is the second group of themes, the social perspectives, that is examined in-depth in this paper.

2) Step two: In this second step, the initial analysis is reinterpreted using the pragmatic theoretical framework and theoretical concepts. New connections and some rearrangement of the categories were developed resulting in the themes reported in the result section of this article. The analyses were conducted by the first author and discussed with the second author at several occasions, both in looking at the relationship between the data and the categories, and refining the theoretical concepts and the empirical findings together. Some intersubjective validation was thus achieved. All the

stages of the analysis process were recorded in N'Vivo for the purpose of documentation and transparency.

V. Ethical considerations

Students were informed in writing of the aim of the study, the purpose of data collection, and their rights as participants. Their participation was voluntary and they signed a written consent form agreeing to participate. Information that could lead to identification of the educational program or the university was not included in this text, in order to ensure students' anonymity. Names presented in the text are pseudonyms. Quotes are chosen to represent a holistic interpretation of the interviews.

VI. Results

We identified four themes in the social dimension group.

A. Emotional roller coaster and other people as comforters

All the interviewed students expressed feelings of an emotional roller coaster during some part of the learning process. Oskar expressed this:

> You are really pleased with your own creations [...] and at the same time you easily feel so stupid [...] when you get stuck and can't move forward, and you don't know what to do or what the problem is. [...] So, it is a huge roller coaster. More than other subjects, that's for sure.

In this emotional roller coaster the students felt comfort in having a partner to work with and not being forced to face the programming assignments alone. This was especially present for those with no prior programming experience and during the first lab sessions. Students felt that they learned less when they sat beside while their partner had the keyboard, but for social reasons they preferred this learning setting. Sandra expressed this:

> Yes.. eh... I don't think individual (labs) would have been good. Because, I notice now, when we work with the individual assignment, when I get stuck I get really stuck, and it is very difficult to move forward when you work alone. And when things are going well, it is much more fun to be able to share that with someone.

Most of the students preferred sitting together when working on their individual assignment, even if that was not required. Most of the students mentioned that the togetherness was both comforting and fun, and described enjoyment in showing accomplishment and sharing the joy of programming. However, one student said that he did not show other students his accomplishments and kept it to himself, being afraid of bragging.

Another student lost his partner due to a student that quit the course and was forced to work alone the two last lab sessions. He described feeling lost and devastated, getting stuck and being all alone. He felt overwhelmed and almost afraid of interacting with the environment.

TABLE I
Participants of the interview study

Name	*Sex*	*Programmed before*	*Time at the keyboard*
Anna[a]	Female	No	Less
Sandra[a]	Female	No	More
Lisa[b]	Female	No	Less
Per[b]	Male	Yes, minor	More
Olle[b]	Male	No	More
Lars[b]	Male	No	Equal
Oskar[b]	Male	Yes, minor	Equal

[a]2017 [b]2018

On the other hand, one student with previous programming experience expressed that he preferred to work alone. He felt that it could sometimes help to verbally describe problems to other people, but that is was the verbalization that was the important component and it would be the same effect to talk out loud to a "rubber duck".

B. Time at the keyboard and social abilities when pair programming

The students in this study were involved in 16 different pair combinations. All the students reported learning by writing code, but some of the students did not write 50% of the time during the lab sessions. The overall reason seemed to be that their partner dominated the keyboard. It could be due to the partner knowing what to write and did not hand over the keyboard. Anna blamed herself how this played out in this social setting:

> It was very much my fault. [...] I think, the role you take when you work with someone.. [...] I noticed that they wanted to sit at the computer, and so it is on me to say "we should switch places now". [...] But somehow, it was like [...] social codes or something, or you don't want to or don't dare to.

Anna also mentioned that their programming ability came into play along with a desire to just finish the assignments, without focus on learning. Anna felt she learned more when writing and that she fell behind when not being able to write code as much. As a consequence she had a hard time at the start of the individual assignment. Her reluctance to claim the right to the keyboard in the pair programming setting affected the possibility to learn at her own pace. Another student, Lisa, had one lab partner that also dominated the lab sessions. Lisa described:

> It was more like I was watching him write, and then he explained what he had done. Or that he explained what he had done before we came to the lab session. And then I added the comments to the functions.

The lab partner took over partly by doing almost all the coding by himself before the lab session, and Lisa just had to tag along not being able to write code at all. She felt she had nothing to contribute with to the solution, which did not feel good. In preparation for the last lab session, she decided to also do the lab beforehand. Now she had the upper hand and

knew some solutions and could take over the keyboard. She described that this felt good and she also felt more competent after this last session. In this setting, Lisa used her ability to get out of a difficult position by altering her strategy before the last session and gained confidence something that Anna was not able to do. Most of the students in our study had one or two positive experiences of pair programming and were positive overall to the setting. They thought that it worked best when they worked with somebody at the same level of programming knowledge, and also preferred working with somebody they knew. Lars said that if you know somebody, you can tell them that some things are wrong in the code without it being socially awkward:

> I think that is important because you could say "no, I think that is wrong", without it becoming awkward. Otherwise you can easily think that the other person is being mean. [...] Yes, and that creates a bad atmosphere which influence how you work negatively.

Amongst the students there were some examples of pair programming partners that were more skilled and chose to take a step back and let the novices write the code to equal extent so that the novices got the chance to learn. The more experienced student initiated this, and the novices felt it were beneficial for them.

In other pairs, when the students had equal numbers of ideas, the writing switched back and forth and both students contributed. But when the students experienced that their partner was unequal in attitude, ambition, or in programming skills, this approach did not work. Per described how he came to have more time at the keyboard:

> I saw that what he wrote was wrong, and I thought "you can do like this" and I tried to sit back and say that "but try to write like this". But he didn't quite understand, so I had to show him [...] I tried to return it (the keyboard), but it was like he was a bit lost. So I ended up doing most of the labs myself because the other person was not capable enough to do them.

Per also described that he wanted to do the assignment ahead of time, before the lab session and expressed disappointment when his partner did not do that. He also expressed that his partner just "writes for the sake of writing" and when he asked him several times if he should hand over the keyboard his partner said it was fine. Sandra was also focused on writing code herself, and felt that testing ideas through writing code was most important for learning. Working with a lab partner that knew less that she did and did not contribute to solving the task, she experienced as a waste of time.

> He is just quiet, like both if he is supposed to write or if it's me that is supposed to write. [...] He wanted me to take the lead and I felt that I didn't want to lead.

The quotes above demonstrate that the more skilled and experienced student had the power to choose how the interaction played out. It is also evident that students' personalities came into play when time at the keyboard was negotiated during lab sessions. At the institutional level, the lab sessions were designed with the instruction to switch every 20 minutes. However, both in the observations and in the interviews, it was clear that this did not happen most of the time. The students often tried to divide the time evenly by switching the keyboard back and forth, but in some instances, depending both on interpersonal social aspects and on programming skill, this did not happen. All the students valued the time spent interacting with the computer (practical thinking while programming) and students who got less time at the keyboard felt more insecure and unprepared when starting the individual assignment.

C. Aesthetic values and norms in computer science

One observation from the lab session concerns students' and teaching assistants' expressed values about code. These values were mostly positive, in terms of "good-looking" code (in Swedish "snygg kod") but sometimes negative, "ugly code". Because of this observation, the students interviewed in 2018 got questions specifically about good-looking code. Another value that was expressed in the interviews was that it seemed to be valued more if you came up with a solution by yourself.

1) "Good-looking" code: The students had a sense of what made a code aesthetically good but, interestingly, they described two interpretations of this that were somewhat at odds with each other. The students stated that the functionality was not in focus for "good-looking" code. Two different codes could accomplish the same functionality but have different appearances. One way of understanding good looking code was that less code was better. Oskar said:

> Using less code to solve something is better-looking, if you can solve a problem with only a few lines of code instead of tens of lines, I would say that.... If the result is the same then it is preferable to have as few lines of code as possible.

This could be seen as a computer science norm of "abstraction", i.e. expressing a lot in a compressed form (short) is preferred. Some students also connected this with feelings like stress and maybe sense of control as Lisa:

> And then it is an enormous amount of code and it's very messy and you don't find anything, and the names of the variable are strange because a lack of imagination. And I at least get stressed by that.

However, some of the students did not think this was important. The other understanding of "good-looking" code was readability, so that the code could be understood by others. Even though all students had some idea of this they did not exactly know how they had come to form the idea of the aesthetics of the code. They had to think for a while, and then some stated that it was from comments from the TA and also from older, more advanced students. One student explicitly mentioned that more fancy methods were a part of good-looking code and that not everyone understood what a feature of good-looking code entailed. Oskar described this:

> Then good-looking code could also refer to what type of methods you use. Which types of command you use, particularly if it's a command that few of the others know about, in that case it will easily become a good-looking command.[...] I know this and the others don't.

Using shorter code was a way of expressing that you "controlled" the environment. In this case the value of short code was the opposite to readable code, so in the concept "good-looking code" you have some contradictory values.

2) Values in how to learn: One value expressed by the students was the value of being able to solve problems by yourself. This was not surprising considering one of the learning goals was to be able to solve a larger assignment individually. Besides that, some students expressed a value in figuring out a solution by themselves even if that would take more time than just ask a friend or a TA for help: Lisa said:

> At the same time I feel there is a value in like, sitting and thinking by yourself. Because of the times when you have seen the problem and actually been able to solve it. It has happened to me at least a few times, it is fantastic. You feel so proud afterwards, like 'I could do this'.

Most students expressed similar views relating to the pair programming sessions, they gained the most from expressing their own ideas in code, not learning as much when their partner tried to do the same. One student stood out regarding this by stating that he felt that he learned by sitting beside and gaining perspective by seeing other ways of "thinking" expressed in the code. Lars reflected on this:

> Like, you learn more by looking at others' writing. Because there are things you wouldn't have thought of otherwise. [...] But if I write it myself then I have created something, developed a thought by myself or how you should express it. And I think it is good to have both. Because if I only try my way forward it will take a really long time. But now when I got to see somebody else's way of thinking and really could compare and maybe it is better if I do like that? And like take the good things from my way and from their way.

The value of thinking and coming up with the solution by yourself is evident in the students' description of learning programming. It also seems taken for granted in the discipline, as showed in both the learning goals and the examination, and furthermore taken over as a value for the individual student as expressed by Lisa above.

D. Having a future in computer science?

This first programming course was mandatory for the students, but the subject is considered a minor in the education. After the first year the students will choose majors, including the possibility to specialize in CS. Of the students interviewed, three had decided on a computer science major, three will not continue with programming, and one was undecided. All of them based their decisions on experiences during the introductory programming course. Although these students were positive to the subject and liked the course, the experience of an emotional roller coaster played a part in the reasoning for some students. Oskar stated that emotions played a part in his reasoning:

> So absolutely it is often an emotional rollercoaster. [...] Because it becomes more than just a course that I should manage somehow, but it's more that maybe this is the first step of an additional four years of programming. And it becomes very much so that if you feel that you are good, then it feels ok - 'then maybe I'll choose CS.' [...] then it will be ok. But when you're are stuck and so on you feel — 'but should I really choose CS?'

For Oskar, it mattered how good he was compared to others in the course. He wanted to be better than most to choose this path in his education. He went back and forth in his decision.

Both Lisa and Olle experienced difficulties during the lab sessions but were able to get through them. Both experienced that they had gained both skills and confidence through this experience. They felt that they had gotten an upper hand compared to the other students when starting their individual assignment. They were both confident that they would continue with CS and the reason for that was that they thought it was so creative and fun to do. Both stated that the course was the reason for the decision. Lisa said:

> It is exciting and it's fun. It's something new also, like it is something I learn from scratch. And you don't have to feel bad just because you don't have a lot of prior knowledge, you can discover the subject as you go along and as you need to know things. That I think has been very...it helps, or it is that kind of thing that motivates me to learn.

The students that chose not to continue with computer science also stated that they needed more help from others in the individual assignment. One explanation may be that they did not enjoy the working alone part of the course as much as embracing the joy of being social when programming and the social dimension of the course situation.

VII. Discussion

According to a transactional theory of learning, the original unit is a transaction that gives rise to experiences. Experiences in an educational setting have as one goal to develop learning, making meaning, and this usually is expressed by explicitly stated knowledge goals and skills. But to take this transactional view seriously, not only the content of knowledge is important, but also the situations and circumstances in which the learning tookes place. Experiences have emotional aspects, creating what Johnson [25] calls emotional traces in cognitive knowledge. Students in our study described an emotional roller coaster when facing both setbacks and successes in learning to program. Several students expressed feelings of self-doubt and wondered if this subject is something that they could or wanted

to pursue in the future. This describes an emotional learning situation where it seems important not to be left alone with the problem. Pair programming seems to ease the emotional pressure and lower the threshold of getting into the subject as also shown in previous research [10], [12]. But how the pair-programming sessions plays out seems to affect both the cognitive outcome, the subjectification in the field, and also a lingering emotional trace associated with the cognitive content. In our interviews, the stated importance of social interaction in the pairs differed widely.

Practical thinking while programming is a process between an individual and the computer environment that has potential to enhance learning and confidence as students perform movements. However, this process builds on direct student interaction with the programming environment, and the pair programming setting is then an obstacle for this development. So, in a way, the practical thinking process necessary for learning programming is at odds with the pair programming setting of not being alone. For this reason, social interactions within the pair become crucial for the quality of the lab sessions. Personality seems to play a part in this, for example, if you are highly aware of social codes and exercise some caution, or if you have a more direct and 'pushy' way of being with others. As mention by other researchers, unequal pairs may be a problem [14]–[16]. Many students have prior experiences of programming and in an unequal pair with a novice, they have the upper hand of deciding how dominant a role they will take.

Learning instructions also play a part. Students should deliver a functional solution by the end of the session, which can result in focusing on completing the task rather than learning, especially if you already know the content. Novices have been reported to feel inadequate, superfluous and doubting their ability [8]. These emotions in the situation could explain students' considerations about whether they want to continue with programming. One student explicitly stated that the emotional roller coaster was a reason not to pursue a programming career. A related observation is that students who embraced working together with others as part of their programming also did not want to continue with the programming. In these considerations, the function of person-formation as identification are shown. Could I identify as a programmer being shy and having social needs to be comfortable?

In a sense, not embracing social or collaborative learning in the CS1 classroom is aligned with the value experienced by the student in solving problems by themselves. A learning goal in the course was to be able to write code to solve a larger problem by yourself, which was aligned with the individual programming assignment. This was mirrored in students' views that it was "better" to solve a problem with your own independent thinking than to ask others for help when needed. This is interpreted by us as a companion norm within the subject field of programming, picked up by some students.

Subjectification is shown in relation to practical thinking while programming. Students who experienced that they could express something of their own, something unique, and got the opportunity to develop confidence in this, said that they "have something to contribute" and that they could become somebody in this field/discipline. One part of this subjectification seems to be to think aesthetically about code and to form one's own opinion about that. Picking up on the subject's norms and aesthetic values is a way of identifying with the discipline, and, starting to think about your own expressions in relation to that is a way towards being a subject in programming. For some students, the social aspect of comparing their competence with their peers seems important in deciding whether to pursue programming as a profession. The value of writing short/compact code, inaccessible to others, is seen as a sign of belonging, and also seems to play part of identifying with the discipline.

VIII. Conclusion and future work

The aim of this study was to better understand how social dimensions come into play when students encounter programming as novices in a pair programming lab setting. From a pragmatic view of learning, the wholeness of students' experiences is considered. Social aspects affect emotions as well as learned content. As our results show, social interactions within the pair become crucial for the quality of the lab sessions. Time at the computer, withstanding an emotional roller coaster, and self-perception as a newcomer in the discipline, all come together with aesthetic values and norms. Novices or shy persons could only hope for a good match in the pairing, being at the mercy of more experienced or more pushy peers.

The solution, we believe, does not consist of stricter regulation of time spent at the keyboard. In our study, the keyboard goes back and forth between the students in the pairs that experienced good cooperation, allowing equal opportunities to interact with the code through the keyboard. The comfort level within a pair seems to greatly affect this equality. One suggestion that we think could equalize the students' possibility to express their own ideas in code is to have two keyboards and two mice connected to one computer. This can coincide with guiding instructions that emphasize the importance of the individual student's interaction with the code and other measures to facilitate the experienced comfort within the pairs.

Our results suggest that the social perspective could be one factor that affects the students' possibilities to engage with the computer by writing code and considering staying in discipline. For that reason, the social perspective should be considered when questions of gender and diversity are discussed. In future work, the the social interplay in different course settings, when novices encounter programming for the first time, should be examined further.

Acknowledgments

We wish to thank all the students who participated in this study. This project is supported by The Swedish Research Council, grant 2015-01920.

References

[1] P. Tuomi, J. Multisilta, P. Saarikoski, and J. Suominen, "Coding skills as a success factor for a society," *Education and Information Technologies*, vol. 23, no. 1, pp. 419–434, 2018.

[2] A. Robins, "Novice programmers and introductory programming," *The Cambridge Handbook of Computing Education Research, Cambridge Handbooks in Psychology*, pp. 327–376, 2019.

[3] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," in *Acm Sigcse Bulletin*, vol. 37, no. 3. ACM, 2005, pp. 14–18.

[4] L. J. Höök and A. Eckerdal, "On the bimodality in an introductory programming course: An analysis of student performance factors," in *2015 International Conference on Learning and Teaching in Computing and Engineering*, 2015, pp. 79–86.

[5] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo, "Introductory programming: a systematic literature review," in *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, pp. 55–106.

[6] K. von Hausswolff, "Hands-on in computer programming education," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ser. ICER '17. New York, NY, USA: ACM, 2017, pp. 279–280. [Online]. Available: http://doi.acm.org/10.1145/3105726.3105735

[7] N. Salleh, E. Mendes, and J. Grundy, "Empirical studies of pair programming for cs/se teaching in higher education: A systematic literature review," *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 509–525, 2011.

[8] C. Schulte and M. Knobelsdorf, "Attitudes towards computer science-computing experiences as a starting point and barrier to computer science," in *Proceedings of the Third International Workshop on Computing Education Research*, ser. ICER '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 27–38. [Online]. Available: https://doi.org/10.1145/1288580.1288585

[9] A. Peters and A. Pears, "Students' experiences and attitudes towards learning computer science," in *2012 Frontiers in Education Conference Proceedings*, 2012, pp. 1–6.

[10] B. Hanks, S. Fitzgerald, R. McCauley, L. Murphy, and C. Zander, "Pair programming in education: A literature review," *Computer Science Education*, vol. 21, no. 2, pp. 135–173, 2011.

[11] C. O'Donnell, J. Buckley, A. Mahdi, J. Nelson, and M. English, "Evaluating pair-programming for non-computer science major students," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 569–574. [Online]. Available: https://doi.org/10.1145/2676723.2677289

[12] M. Celepkolu and K. E. Boyer, "Thematic analysis of students' reflections on pair programming in cs1," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 771–776. [Online]. Available: https://doi.org/10.1145/3159450.3159516

[13] Lan Cao and Peng Xu, "Activity patterns of pair programming," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 88a–88a.

[14] N. A. Bowman, L. Jarratt, K. Culver, and A. M. Segre, "How prior programming experience affects students' pair programming experiences and outcomes," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 170–175. [Online]. Available: https://doi.org/10.1145/3304221.3319781

[15] C. M. Lewis and N. Shah, "How equity and inequity can emerge in pair programming," in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ser. ICER '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 41–50. [Online]. Available: https://doi.org/10.1145/2787622.2787716

[16] F. J. Rodríguez, K. M. Price, and K. E. Boyer, "Exploring the pair programming process: Characteristics of effective collaboration," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 507–512. [Online]. Available: https://doi.org/10.1145/3017680.3017748

[17] G. Biesta and N. C. Burbules, *Pragmatism and Educational Research*, 2003.

[18] L. Östman, K. Van Poeck, and J. Öhman, "A transactional theory on sustainability learning," in *Sustainable Development Teaching: Ethical and Political Challenges*. Routledge, 2019, pp. 127–139.

[19] P.-O. Wickman, *Aesthetic experience in science education: Learning and meaning-making as situated talk and action*. Routledge, 2006.

[20] K. Van Poeck and L. Östman, "Sustainable development teaching in view of qualification, socialisation and person-formation," in *Sustainable Development Teaching: Ethical and Political Challenges*. Routledge, 2019, pp. 59–69.

[21] G. Biesta, "Good education in an age of measurement: On the need to reconnect with the question of purpose in education," *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education)*, vol. 21, no. 1, pp. 33–46, 2009.

[22] P.-O. Wickman and L. Östman, "Learning as discourse change: A sociocultural mechanism," *Science education*, vol. 86, no. 5, pp. 601–623, 2002.

[23] W. James, "Essays in radical empiricism," 1912/2003.

[24] J. Dewey, *Experience and nature*. New York, NY: Dover Publications, 1925/1958.

[25] M. Johnson, *The Meaning of the Body: Aesthetics of Human Understanding*. the University of Chicago Press, 2007.

[26] A. Eckerdal, R. McCartney, J. E. Moström, K. Sanders, L. Thomas, and C. Zander, "From limen to lumen: computing students in liminal spaces," in *Proceedings of the third international workshop on Computing education research*. ACM, 2007, pp. 123–132.

[27] A. Eckerdal, "Relating theory and practice in laboratory work: A variation theoretical study," *Studies in Higher Education*, vol. 40, no. 5, pp. 867–880, 2015.

[28] B. Shneiderman, "Teaching programming: A spiral approach to syntax and semantics," *Computers & Education*, vol. 1, no. 4, pp. 193 – 197, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0360131577900082

[29] K. von Hausswolff, "Practical thinking in programming education," in *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, 2017, pp. 203–204.

[30] ——, "Practical thinking while programming: A deweyan approach to knowledge in computer science," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 268–269.

[31] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[32] S. Elo and H. Kyngäs, "The qualitative content analysis process," *Journal of advanced nursing*, vol. 62, no. 1, pp. 107–115, 2008.