Students' learning process in the computer laboratory

Anna Eckerdal
Uppsala Computing Education Research Group, UpCERG
Department of Information Technology
Uppsala University
Uppsala, Sweden
Anna.Eckerdal@it.uu.se

Anders Berglund
Uppsala Computing Education Research Group, UpCERG
Department of Information Technology
Uppsala University
Uppsala, Sweden
Anders.Berglund@it.uu.se

Michael Thuné
Uppsala Computing Education Research Group, UpCERG
Department of Information Technology
Uppsala University
Uppsala, Sweden
michthun@gmail.com

***Abstract*—In this full research-to-practice paper we study novice programming students learning process in the computer laboratory. Working with laboratory assignments is an important component when students learn to program. Here the assignments are intended to help students consolidate theoretical understanding and simultaneously train practice. However, it has been observed that the learning outcome of such laboratory sessions often is unsatisfactory. In this article we ask the question "How do novice students go about learning in the computer laboratory?" We analyse empirical data on novice students working in pairs in the laboratory, which is common in a first programming course. The data consists of video films of students where they discuss and solve programming problems, screen captures of what the students typed during the same laboratory session, and stimulated recall interviews with the students after the laboratory session. In the analysis we use an approach inspired by phenomenography and variation theory. We specifically focus on typical stages in the learning process when students learn in the programming laboratory. In doing so we have identified successful and less successful learning paths, where variation can play different roles. The stages identified in students' learning process are I. Students first need to become aware of a lack of clarity. In the data we have identified different ways in which this necessary awareness was trigged; II. If, and in that case how, they resolve the lack of clarity. In all the stages we found successful and less successful ways in which students' handle the situations. We illustrate the stages and discuss how and why variation may play different roles in the different stages of students' learning, specifically focusing of the unsuccessful learning paths. Lastly, we discuss what these findings can tell us about how programming labs could be designed to promote learning in terms of helping students to avoid the unsuccessful paths identified.**

Keywords—Programming, Laboratory, Learning process, Novices, Phenomenography, Variation Theory

I. Introduction

In this article we discuss preliminary findings on programming students' learning progress in the computer laboratory. Programming has become an increasingly important part of school education as well as many higher education programs. Introductory programming has however been reported to be difficult to learn, and despite much focus on how to teach introductory programming, and how students learn it, many problems seem to prevail [1], [2].

An important learning activity in computer programming courses is that students work with programming assignments in the computer laboratory. This is intended to help consolidate theoretical understanding of programming and to train the skills to solve programming problems in practice. This article reports on findings from a research project focused on the relationship between students' learning of theory and their learning of practice in the computer laboratory. Our research question is:

How do novice students go about learning in the computer laboratory?

The outline of the paper is as follows. Related work and the theoretical framework are presented in Section II, and the empirical study in Section III. We then describe the analysis in Section IV and the results in Section V. Section VI presents Discussions, and Section VII Conclusions.

II. Background

Novice programming is a well-studied research area. Some examples of such areas are problems with high dropout and failure rates [1], [2], [3]; emotional factors influencing students' learning like self-efficacy and motivation [4], [5], [6], [7]; tools to support students' learning [8], [1]; inequalities such as gender issues [9], [10]; different content and teaching techniques [6], [2]; and misconceptions [11], [12]. For an overview of research areas see [1].

The present study focuses on novice programming students' learning in the computer laboratory. Learning in the laboratory is highlighted as central for students' learning in important documents like the ACM/IEEE Computing Science Curricula [13]. Previous research points to the importance of as well as problems related to learning in the laboratory [14], [15]. In the computer laboratory students are expected to learn concepts [16], [17] as well as programming skills [18], [19]. Eckerdal et al., [20] found that students had problems learning both the theory and practical skills as well as "to translate from an abstract understanding to concrete implementation", that is to do the hands-on programming. Höök and Eckerdal [15] found that time spent at the computer programming hands-on was the most important factor for success in a first programming course. Most previous research focus on theory and practice separately, while the present study focus on the learning *process* where theory-

oriented as well as practice-oriented actions contribute and interplay in students' learning.

The theoretical underpinnings of the project can be found in phenomenography [21] and variation theory [22]. While phenomenography aims to explore the qualitatively different ways, in which something (usually labelled a *phenomenon* in phenomenographic literature) is experienced or understood by a certain group of people, variation theory takes these distinctions further, to base a pedagogy from the insights of phenomenographic research. The basic assumption here is that variation is a necessary, even though not sufficient, condition for learning. That is, while phenomenography aims at revealing the variation in how someone experiences (or understands or sees) something, variation theory deploys such variation as a mean to develop teaching. This is done for example by keeping aspects of the phenomena invariant (or stable) while another aspect(s) varies. It is argued that this would be a fruitful way to use for a teacher to make the varying aspect visible for the leaner, and that is in this way, a new aspect of the phenomenon available for the learner.

In this project we thus, among other things, explore aspects of variation that come to the fore during teaching in a general sense, for students learning programming. In this paper we focus on variation in students approaches to problem solving in the computer lab, specifically how they use theory-oriented and practice-oriented actions in their learning process.

As the project takes its theoretical point of reference in phenomenography, issues of the trustworthiness of the project are generally discussed in terms of credibility, transferability, dependability and confirmability, as suggested by Lincoln & Guba [23]. Following the rather well agreed procedures for phenomenographic research, we argue that the analysis of Sandberg [24] is relevant for our project.

III. The Empirical Study

The article present results from a larger study where we investigated the interplay between how novice students learn theory and how they learn practice in the computer laboratory [25], [26]. To this end we video filmed students from three Swedish universities in natural laboratory settings, scheduled in courses mandatory for the students to take. The learning goals for the laboratory assignments were similar but the programming language differed, Java at two universities and Python at one. During the particular laboratory sessions, we video filmed, students worked with similar topics. The Java exercises addressed methods that return value, and the Python assignments concerned functions.

29 students in 13 teams were video filmed, about one hour per team. Each team consisted of two or three students working together in front on one computer, solving programming problems. The students volunteered to participate in the study. Two of the authors attended each laboratory session. The teams were video filmed from behind to get an overview of how they all behaved, and the computer screens were captured together with the students voices so we could follow in detail their progress and discussions. The day after the laboratory session, or a few days after, we had stimulated recall interviews individually with all students. For the present analysis the main data source has been the films of the computer screen with the students' voices and what they typed. We have only to some extent used the film from behind and the interviews.

IV. Analysis

We first searched through all the films for sequences where we judged that students seemed to have learned something, or got a new insight in relation to the goal of the lab. We call these identified sequences *episodes*, inspired by Ingerman et al. [27]. The possible episodes were identified and analysed by the first author, and then discussed by all authors, until consensus was reached. The analysis presented in this article is built on eight such episodes, each one not more than a few minutes long.

We initially analysed each episode in a way proposed by [26], inspired by phenomenography and variation theory [21], [22]. This analysis highlights how theory and practice interact in students' learning and which role variation, in the sense described in variation theory [22], plays in students learning. In doing so we both consider the variation that was created by the teacher in the laboratory material, and the variation created by the students when discussing and trying out different suggested solutions to problems in code. The result from this analysis has been presented elsewhere [25], [26].

In a next step the results of the first analysis were studied in relation to each other with the aim to find common patterns across episodes using an inductive content analysis approach [28]. We focused on the evolution of the learning process in each episode. The three authors carried out the second step of the analysis collectively. In an interactive process, patterns were suggested, assessed in relation to the pool of results from the first step of analysis, revised, assessed again, etc., until consensus was reached.

V. Results

In this project, we call what the students say and do actions. We use the term *theory-oriented action* when, in our interpretation, students discuss general principles, for example what a programming construct like if-statements means. We use the term *practice-oriented action* when, in our interpretation, the students do something in a specific case, for example write an if-statement in the code they work on, or read from the lab material.

A. Examples from the data

In the following we show, for illustration, two examples of an identified episodes. To preserve anonymity, the student names in these excerpts are fictitious. Moreover, male and female names have been used randomly, without considering the actual gender of each student. Translation to English was done by the authors. The episodes are from a course where students first encounter Java constructs in laboratory sessions. The written instructions for a session contain a short introduction to one or more constructs. The students are asked to try them in exercises, before the constructs are explained by the teacher in a lecture.

Episode A.

When the episode begins, the students have written a method with the given head *int distClosestWall()* which is supposed to calculate and return the distance between a Turtle-object and the closest of the four "walls", that is the frame of the window the turtle appears in. The students copied the method head but initially printed out the distance instead of

returning it. They get a compiler error. One of the students finds a similar example in the lab instruction and reads aloud. We illustrate the interaction between practice-oriented and theory-oriented actions in the episodes with the letters ***P*** and ***T*** respectively. The distinction between the kinds of actions is not clear-cut, this is our interpretation. The episode develops as follows:

A1. *P* Bill: "before the method name it says double which means that the method should return a value of this type." [reads aloud]
A2. *T* Celia: [reads the error message] "missing return statement" Does it mean that nothing is there?
A3. *P* Bill: [types return]
A4. *T* [… The students discuss how to write the return statement]
A5. *T* Bill: But I understand that return is kind of…
A6. *T* Celia: System.print.out, isn't it? Or, what's the difference?
A7. *P* Bill: Returns an answer [reads from the lab instruction]
A8. *P* [… The students continue the discussion and change the code to return y, which is correct.]
A9. *P* Celia: When do we use System.print?
A10. *T* Adam: It's kind of the same thing
A11. *T* Celia: But why didn't it work here?
A12. *T* Adam: I suppose it's because we use int
A13. *P* Celia: Aha, int and return, and System.print.out is that void for example?
A14. *T* Bill: I guess int and double needs a stop signal kind of …
A15. *T* Celia: Okay

Table 1. An illustration of how theory-oriented (T) and practice-oriented (P) actions interact in students' learning and support a successful learning process.

In our interpretation, the learning outcome of this episode is that the students reach a partial understanding of the relation between the return value and what is returned from a method.

Episode B

In Episode B, two students are working with an exercise where the task is to write a method to draw a circle on the computer screen. According to the lab instructions, this should be a method which does not return a value. The only thing that should happen when the method is executed is that a circle should appear on the computer screen. The episode develops as follows:

B1. *P* [David types "public"]
B2. *P* David: Should we make an int?
B3. *T* Erica: But do we need to make an int? It won't come out a number, it should, yes.
[…]
B5. *T* Erica: We will let in a number. But then we perhaps don't need an int before?
B6. *T* David: Don't know.
B7. *T* Erica: It should only run now.
[…]
B8. *P* [David reads the description of to the assignment in the lab instruction: "Write a method void drawCircle(int radius)"]
B9. *P* David: Ok, it should be void [types "void". Types "public void" on the screen.]
B10. *T* Erica: What does void mean? I forget.
B11. *T* David: [Sighs] Yes what is void now again. I think void means that it doesn't…
B12. *T* Erica: It doesn't print anything [snaps her fingers]
B13. *T* David: … It doesn't print anything…
B14. *T* Erica: Bang, it just does something, that's it.
B15. *T* David: Hm, right, it doesn't expect anything … It shouldn't be an answer.
B16. *T* Erica: … but it should just do something.
B17. *P* David: And it should be called drawCircle [David types "drawCircle(int radius)" which makes the method head "public void drawCircle(int radius)"]

Table 2 Illustration of a second learning episode where students' variation of theory-oriented (T) and practice-oriented (P) actions leads to successful learning.

In this episode the students struggle with how to express the type of value to be returned by the method. At the end of the episode they have become aware of the keyword "void" in Java, referring to a method that does not return any value.

B. The learning process

After having identified and analysed several episodes like the ones used for illustration above, we found that the learning process in the episodes could be described in terms of two stages, with specific variants of each stage. Fig. 1 summaries our findings on the successful episodes, and adds examples of sequences that were not successful. The dark squares illustrate the successful episodes, while the light-coloured squares illustrate the unsuccessful ones. At the top and bottom of the figure are the identified approaches, typical for the successful and unsuccessful sequences respectively in squares with rounded corners.

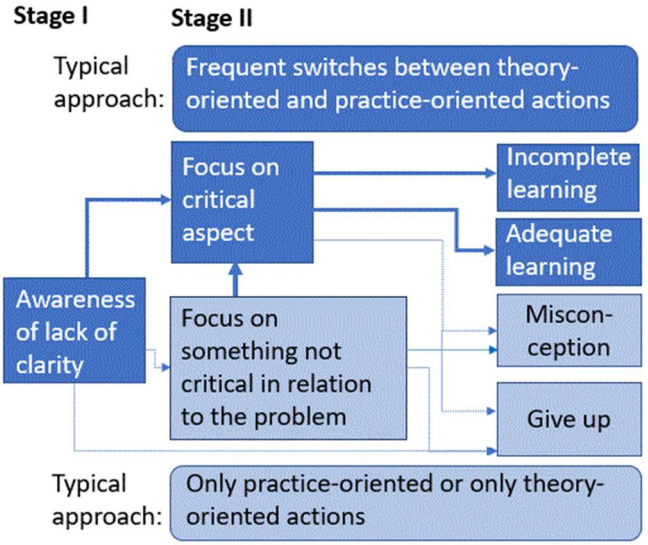

Fig. 1. Two stages of students' learning with successful episodes (dark) and unsuccessful sequences (light), and respective typical learning approaches identified.

Stage I: Becoming aware of a lack of clarity

A recurring pattern in our data is that *practice-oriented actions* cause students to become aware of a lack of clarity in relation to the object of learning. That is, the practice-oriented action helps students to discern what they do not know theoretically or are not capable of doing. This can make

students give up, or focus on something not relevant for the problem, or it can begin an episode of learning. This can then begin an episode of learning. In our data we have observed three different variants of this stage:

- The awareness comes as a result of *discernment of variation* in relation to some aspect of the object of learning.
- The awareness is triggered by students' immediate recognition that there is something in the lab exercises that *they do not know or understand*.
- The awareness of lack of clarity follows from the students *observing that an error occurs*, for example that the compiler issues an error message or that the program yields an incorrect result when executed.

Episode B illustrate the first variant of Stage I. When David types 'public' and considers to type 'int', a practice-oriented action, the students realize a lack of clarity, they don't know how to continue. In a theory-oriented discussion they then recall what 'void' means. Similarly, episode A illustrates the second variant of Stage I. The awareness of a lack of clarity comes from students' realizing that they don't know neither the syntax nor the semantics of return values in method heads, which starts a theory-oriented discussion.

Stage II: Resolving the lack of clarity

In Stage II of an episode, the students try to resolve the lack of clarity. In some cases, they did not manage to identify what was the problem but focus on something not relevant for the problem at hand. If they focus on the relevant critical aspect, they typically do that through a variation that is the *interplay between practice-oriented and theory-oriented actions*. The interplay is characterized by frequent switches between theory-oriented actions, like the discussion on the difference between return and print in episode A, and practice-oriented actions, like when Bill reads and types. By this interplay, the students first come to discern variation with respect to some aspect of the object of learning (or explore the corresponding dimension of variation further, in case they already discerned this dimension in Stage I). Then, in a second part of Stage II, the interplay is used to make meaning of the variation. We have observed two kinds of outcome of this stage of an episode:

- The meaning constituted by the students is adequate but *incomplete,* i.e., it does not fully address the problem at hand. Episode A is such a case. Eventually the students in that episode type the correct return statement (A8) and Adam and Bill try to explain why (A12, A14). The students seem to have obtained a first glimpse of the relation between the return value and what is returned. However, they neither appear to understand the syntax completely nor to grasp the full meaning. Consequently, this episode is an example where the students manage to partially resolve the lack of clarity.
- The meaning constituted by the students is *adequate and fully addresses the problem* at hand. For example, after action B8-9 in Episode B it is clear to the students that the right answer to the question of what to write after "public" in this case is "void". However, they do not leave it by that. Instead they continue with a theory-oriented assessment, in statement B10-16, where they try to make sense of "void" being the correct answer. They do not leave the issue until they have recalled the meaning of "void" (a concept that they had encountered in a previous part of the introductory course in programming) and they subsequently have established that this meaning agrees with their own understanding of what the method in the current exercise should do. Finally, they have resolved the lack of clarity completely.

In summary, in Stage I the students become aware of a lack of clarity, which initiates an episode of learning. In Stage II the actual learning takes place through *interplay between practice-oriented and theory-oriented actions* in frequent switches. This helps the students to discern variation with respect to some aspect of the object of learning and to make meaning of the variation.

Unsuccessful sequences

An episode is defined as a sequence of the laboratory session where the students manage to learn something on their own (i.e., without needing to ask the teacher for a full explanation). However, when we have looked for episodes in our data, we have often found sequences where students became aware of a lack of clarity, but where nevertheless no learning took place. Examples of these are light grey in Fig. 1. For this analysis however, we focus on which roles the theory-oriented and practice-oriented actions play. We have observed three typical cases of such unsuccessful sequences.

Case 1: The students are not able to resolve the lack of clarity, despite trying

In some sequences, students became aware of a lack of clarity and continued with a second stage, where they tried to resolve the lack of clarity, but failed. For example, we found in the data examples of students focusing on something not relevant to their problem, e.g. they suggest that the number of spaces in the Java code cause the compiler error which was not the case.

We observed that in such cases, the students often took *only practice-oriented actions*. They made a sequence of unsuccessful attempts to modify the program by practice-oriented trial and error, but finally they gave up. However, we also have examples of sequences that became unsuccessful since the students *only took theory-oriented actions*. They got stuck in theory-oriented discussions and gave up without having tried any of their ideas in practice. It seems vital for the learning that students are encouraged to make frequent switches between theory-oriented and practice-oriented actions during lab-sessions.

Case II: The students think that they reach clarity, but their conclusion is erroneous

There are also sequences where the students become aware of a lack of clarity, then try to resolve it and experience their efforts to be successful. The students think that they have clarified the issue, but we as researchers conclude that the meaning constituted by the students is inadequate to resolve the lack of clarity.

Case III: Continue without trying to resolve the lack of clarity

If it was possible to continue with the lab exercise despite the lack of clarity, some student groups did not take the learning opportunity. This can happen at any point in the learning as illustrated in Fig. 1. The following is an excerpt

from one of the stimulated recall interviews that were made after the video recorded lab sessions:

Fred: […] Because it says "public" there for example [pointing to the code], we didn't have a clue what it meant, but we saw that everyone had written it, so we assumed it was something that had to be there.

Apparently, the students in Fred's group were aware of a lack of clarity but decided to leave this unresolved. There are several examples of this kind of "unreflecting practice" in our data, where students do something based on how they or others have done it before, without trying to understand the reason behind.

VI. Discussion

This section discusses our findings and possible implications for teaching. We identified two stages in students' learning process in the laboratory.

In Stage I of the learning process the students need to become aware of a lack of clarity. We found that practice-oriented action often helps students to discern what they do not know theoretically or are not capable of doing. However, we have identified sequences in the data where students did not even realize that there was a problem in the first place.

In Stage II practice- and theory-oriented actions are taken in an attempt to resolve a lack of clarity. Practice-oriented actions can be seen as examples of what Schön refers to as 'reflection-in-action' [29]. In the same way, theory-oriented actions in Stage II can be seen as examples of 'reflection-on-action' [29]. Our data specifically points to the importance of the interplay between the two for learning in the laboratory.

We found episodes where students failed to identify what is critical in the problem. This has previously been discussed in phenomenographic research [30]. We believe this is common among novice programmers who cannot see through all details in the code and e.g. understand compiler messages. Even if students become aware of what the problem is, they do not always manage to resolve it, as discussed in Section V.B above. In Case I, we notice two distinct approaches that are not always successful. First, students might take theory-oriented actions only, or second, they take practice-oriented actions only. The first approach is less discussed as unsuccessful in educational literature and might be common specifically in programming education [14]. We see sequences in our data where students spend lots of time discussing their code without changing and re-compiling it. The students in the episodes study a program where they are probably used to focus on theories, which may explain the finding of this approach in the data. The second approach, where students only take practice-oriented actions, we label 'un-reflected practice'. We have seen two variants of this. When a lab exercise is designed so that students can continue with the task without resolving the lack of clarity, they may simply go on instead of taking the learning opportunity offered by the lack of clarity. Adam says in the interview:

Adam: I'm the kind of kind of person who, I don't need to know everything when I do things. I don't like to know, understand everything, I just want to do it, and then, if it works it works.

Another example of un-reflected practice is 'trial and error coding' where students seem to randomly, with no or little reflection, test different code structures with the sole goal to make the errors disappear when compiling. They may then find a solution that allows them to continue with the task, but they have still not resolved the lack of clarity.

Students using this trial and error approach miss a learning opportunity. However, successful interaction between practice-oriented and theory-oriented actions can also, at a cursory glance, be interpreted as trial and error. The difference is that in the successful cases, when the practical trial generates an error, the students assess this theoretically and base the next trial on a "hypothesis" resulting from this theoretical assessment. This can explain why previous research [31] found that trial and error is one of the strategies used by successful computing students to overcome difficulties and thus make progress in learning of both concepts and skills.

Our results indicate that it is vital for the learning that students are encouraged to make frequent switches between theory-oriented and practice-oriented actions during lab-sessions. This is in contrast to too long focus either on theory-oriented actions like discussions only, or practice-oriented actions like when students randomly try different code solutions without reflection. The teacher can make the importance of frequent switches explicit to the students, and construct the lab material in a way that encourages such switches. One way of doing this could be to include many theoretical questions closely connected to detailed aspects of the labs' learning goals as reflected in the coding exercises. We emphasize detailed questions since it is obvious in our data that novices get stuck on details that the teacher might not even realise is a problem. An example of this from our data is a group of students who, in a long sequence of discussions and practical trial, tried to understand both the meaning and syntax of curly braces in Java.

Another suggestion to make it likely that students frequently switch between practice-oriented and theory-oriented actions is to make material for both kinds of actions readily available. That many student groups in our data pool *did* switch frequently between practice-oriented and theory-oriented actions, may be due to the fact that the teachers who designed the laboratory exercises provided easy access to the theory by including explanations to relevant concepts together with the assignments, and in addition readily available online documents with short explanations of concepts like the if-statement, the for-loop, the main-method etc. as part of the laboratory material. We believe this was important since novices might find it hard to find relevant information online if not guided, and not students buy the text book. Episode A illustrates this. In A1 and A7, Bill is reading such explanation of return values. Our interpretation is that this lowered the threshold for the students to take a fruitful approach, switching between practice-oriented and theory-oriented actions as they go between reading the theory documents, discussing with each other, and writing code while solving the assignments.

We suggest that these approaches can support students learning and might discourage students from believing their erroneous solution to a problem is correct, and even prevent them from leaving a problem unresolved

VII. Conclusions

In this article we asked *How do novice students go about learning in the computer laboratory?* We investigated university students' learning process when they worked with programming problems in groups of two or three, in the

computer laboratory. When the learning process is successful (see Stage II), students can discern and make meaning of aspects of the object of learning not previously discerned. This however requires that students first realize that there exists a lack of clarity, what that lack of clarity is, and that they then try, and manage, to resolve it. One problem is that students might believe that they have solved the lack of clarity if, for example, they get no compiler error, while they in fact have not solved it. One such example from the data was a group of students who did not understand what "return a value" means and changed the method head from public int to public void, then let the method print the result and believed the assignment was done. Further, even if the students discern a lack of clarity but the laboratory assignment is constructed in a way that allows students to continue without trying to resolve it, an important learning opportunity might be lost. For the students to make an effort to resolve the lack of clarity requires that they have good strategies and that there are sufficient resources to support the students so they don't give up. We suggest that carefully constructed lab material, which encourages frequent switches between theory-oriented and practice-oriented actions, and with theoretical and practical resources readily available, can support successful learning.

ACKNOWLEDGMENTS

This work was supported by the Swedish Research Council under grant 2011-5924. We want to thank our colleagues Lennart Rolandsson and Inga-Britt Skogh, and the students who participated in this study.

REFERENCES

[1] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, ... 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. ACM

[2] A. V. Robins, "Novice Programmers and Introductory Programming." In Fincher, S. A., & Robins, A. V. (Eds.). *The Cambridge Handbook of Comp Edu Res*. Cambridge Uni. Press. 2019, 1: 327-376.

[3] P. Kinnunen, and L. Malmi. "Why students drop out CS1 course?". In *Proceedings of the second international workshop on Computing education research.* 2006, September, pp. 97-108.

[4] A. J. Mendes, L. Paquete, A. Cardoso and A. Gomes. "Increasing student commitment in introductory programming learning". In *Frontiers in Education Conference* (FIE) 2012, pp. 1–6. New York, NY: IEEE.

[5] M. Papastergiou. Are computer science and information technology still masculine fields? High school students' perceptions and career choices. *Computers & education*, *51*(2), 594-608, 2008.

[6] R. P. Medeiros, G. L. Ramalho, and T. P. Falcão. "A systematic literature review on teaching and learning introductory programming in higher education." *IEEE Transactions on Education*, *62*(2), 77-90, 2018.

[7] M. J. Tsai, C. Y. Wang, and P. F. Hsu. "Developing the computer programming self-efficacy scale for computer literacy education." *Journal of Educational Computing Research*, *56*(8), 1345-1360, 2019.

[8] P. Gross, K. Powers. "Evaluating assessments of novice programming environments." In *Proceedings of the first international workshop on Computing education research* 2005, October, pp. 99-110.

[9] J. Wang, H. Hong, J. Ravitz, J., and M. Ivory. "Gender differences in factors influencing pursuit of computer science and related fields." In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education,* 2015, pp. 117-122.

[10] T. J. Weston, W. M. Dubow and A. Kaminsky. "Predicting women's persistence in computer science-and technology-related majors from high school to college." *ACM Transactions on Computing Education (TOCE)*, *20*(1), 2019, 1-16.

[11] K. Sanders, L. Thomas. "Checklists for grading object-oriented CS1 programs: concepts and misconceptions." *ACM SIGCSE Bulletin*, *39*(3), 166-170, 2007.

[12] K. Sanders, J. Boustedt, A. Eckerdal, R. McCartney, J. E. Moström, L. Thomas, and C. Zander. "Student understanding of object-oriented programming as expressed in concept maps". In *Proceedings of the 39th SIGCSE technical symposium, 2008* (pp. 332-336).

[13] "Computer Science Curriculum, Joint Task Force on Computing Curricula", Association for Computing Machinery (ACM), IEEE Computer Society, 2013.

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

[15] A. Robins, J. Rountree, and N. Rountree. "Learning and teaching programming: A review and discussion." *Computer science education*, *13*(2), 137-172. 2003

[16] A. Eckerdal, and M. Thuné. "Novice Java programmers' conceptions of" object" and" class", and variation theory". *ACM SIGCSE Bulletin*, 2005, *37*(3), 89-93

[17] N. Thota, and A. Berglund. "Learning computer science: Dimensions of variation within what Chinese students learn". *ACM Transactions on Computing Education (TOCE)*, 16(3), 1-27, 2016

[18] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas and C. Zander. "Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers." *Computer Science Education*, *18*(2), 93-116. 2008

[19] T. Clear, J. Whalley, P. Robbins, A. Philpott, A. Eckerdal, M. Laakso, and R. Lister. "Report on the final BRACElet workshop". *Journal of Applied Computing and Information Technology*, *15*(1), 2011.

[20] Eckerdal, A., McCartney, R., Moström, J. E., Sanders, K., Thomas, L., & Zander, C. (2007). From Limen to Lumen: computing students in liminal spaces. In *Proceedings of the third international workshop on Computing education research* (pp. 123-132). ACM.

[21] F. Marton and S. A. Booth. *Learning and awareness*. 1997. Psychology Press.

[22] F. Marton and A.B.M. Tsui. *Classroom discourse and the space of learning*. 2004 Mahwah, NJ: Lawrence Erlbaum Associates.

[23] Y. S. Lincoln and E. G. Guba. Naturalistic Inquiry. Newbury Park, CA. SAGE. 1984

[24] J. Sandbergh (1997) Are Phenomenographic Results Reliable?, Higher Education Research & Development, 16:2, 203-212,1997

[25] A. Berglund, & A. Eckerdal. "Learning practice and theory in programming education: Students' lived experience". In *LaTiCE 2015* (pp. 180-186). IEEE Computer Society.

[26] M. Thuné and A. Eckerdal. Analysis of Students' learning of computer programming in a computer laboratory context. *European Journal of Engineering Education*, 1-18, 2018.

[27] Å. Ingerman, C. Linder and D. Marshall. "The learners' experience of variation: following students' threads of learning physics in computer simulation sessions." *Instructional science*, *37*(3), 273-292. 2009

[28] S. Elo and H. Kyngäs. "The qualitative content analysis process." *Journal of advanced nursing*, *62*(1), 107-115. 2008

[29] D. A. Schön. *Educating the reflective legal practitioner*. Clinical L. Rev., 2, 231-250. 1995.

[30] M. L. Lo and P. P. M. Chik. "Two horizons of fusion." *Scandinavian Journal of Educational Research*, 60(3), 296-308. 2016

[31] R. McCartney, A. Eckerdal, J. E. Moström, K. Sanders, and C. Zander. Successful students' strategies for getting unstuck. In ACM SIGCSE Bulletin 39(3), 156-160, 2007. ACM