ASPA: A Static Analyser to Support Learning and Continuous Feedback on Programming Courses. An Empirical Validation

Roope Luukkainen
LUT University
Lappeenranta, Finland
roope.luukkainen@lut.fi

Jussi Kasurinen
LUT University
Lappeenranta, Finland
jussi.kasurinen@lut.fi

Uolevi Nikula
LUT University
Lappeenranta, Finland
uolevi.nikula@lut.fi

Valentina Lenarduzzi
University of Oulu
Oulu, Finland
valentina.lenarduzzi@oulu.fi

ABSTRACT

For decades there have been arguments how to teach programming in the basic courses. Supportive intervention methods to improve students' learning and methods to improve assessment process have been widely studied. There are various successful methods to each topic separately, but only a few of them fit for both. In this work, we aimed at validating ASPA a static analyser tool that supports learning and continuous feedback on programming courses. For this purpose, we designed and conduct an empirical study among 236 students enrolled in the basic programming course, that voluntary adopted the tools during the project development activities. We first profiled the students, then, evaluated the attitude toward using ASPA, the perceived ease of use, and the perceived usefulness. Results showed that ASPA is a good helper for the entire course and especially the student's programming assignments, and it also helps to improve the students' grades.

KEYWORDS

Static Analysis Tools, Software Education, Programming Education, Empirical Software Engineering, CS1

ACM Reference Format:
Roope Luukkainen, Jussi Kasurinen, Uolevi Nikula, and Valentina Lenarduzzi. 2022. ASPA: A Static Analyser to Support Learning and Continuous Feedback on Programming Courses. An Empirical Validation. In *44nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3510456.3514149

1 INTRODUCTION

Traditionally programming has been studied at the beginning of a computer science (CS) degree [14]. But for decades there have been discussions on how programming should be taught and assessed. There is no agreement on which programming language or even which paradigm should be used. [4, 21]. This study focuses on introductory level programming course, i.e. the first computer science course (CS1) and how to support self-study and improve feedback during the course. CS1 has been lectured in our university ever since the computer science department was established in 1986. The course has had varying implementations over the years. The current course implementation in LUT University is based on systematic rehabilitation and two major changes.

According to our records, between 2002 and 2009 the course pass rates varied from 36 % to 68 %. During this time constant revisions to the learning process and the course contents were done, a process which is documented in Nikula et al. [25] and Nikula et al. [26]. The major changes were an integration of an automatic code checker in 2001, and a change of programming language from C to Python in 2006 [16]. Due to the rehabilitation process and continuous improvements during the last four years pass rate has stabilised between 60 % and 70 %.

Currently, CS1 can be completed twice in every academic year. The traditional version during an autumn semester and as a self-study course during a summer period, with around 550 and 100 participants, respectively. For the majority of the participants, the course is a compulsory part of major or minor studies. There are students from science and business degree programmes. The used programming language is Python and there are four assignment types. Weekly assignments and quizzes which are graded automatically as pass or fail, as well as, a course project and an electronic examination, which are manually graded from 0 to 5, 1 being the lowest passing grade.

The course has a heavy programming focus and the students are required to write 30-60 small programs as weekly programming assignments, one larger programming project, and in the electronic exam they also need to write a working program. Consequently, in the end of the course most of the students can produce a working program, and thus we have moved one step further and we expect the students to write understandable, maintainable, and extendable programs. Since these characteristics are typically subjective, we developed a style guide for the course which defines what is acceptable and what is not in this course. The style guide includes basic rules like use of functions, parameters, and return values and prohibits the use of global variables. That is, the style guide focuses on standard structural programming rules that are also covered in

ICSE-SEET '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9225-9/22/05…$15.00
https://doi.org/10.1145/3510456.3514149

the course style guide, programming guide, lectures, and all programming examples used in the course. Still many students fail to follow these rules and our current analysis suggests that not all the students read and follow the course study materials. This issue becomes evident when the staff checks the student programs and gives feedback on them. Since checking all the student submissions is a very laborious task, the students get feedback only for their course project and exam submissions.

To provide students timely feedback on their programs and especially how they comply with the course style rules, we developed ASPA that analyses Python programs against the course style guide and provides feedback on it. In this work, we aim at validating ASPA, a static analyser tool that supports learning and continuous feedback on programming courses.

For this purpose, we designed and conduct an empirical study among 236 students enrolled in the basic programming course, that voluntary adopted the tools during the project development activities. We first profiled the students, then, evaluated the attitude toward using ASPA, the perceived ease of use, and the perceived usefulness.

Results shows that ASPA is positively perceived by the students as a good helper during the development process both for the entire course and the assignment tasks. Moreover, ASPA helped them to improve their development performances.

Paper Structure. Section 2 introduces the tool objective of this empirical study, while Section 3 presents the empirical study design to validate the tool. Section 4 reports the obtained results, while Section 5 discusses them. Section 6 highlights the threats to validity of this work and Section 8 concludes the work introducing some future research directions.

2 STATIC ANALYZER TOOL - ASPA

ASPA was developed to support the first programming course with Python, so it was implemented with Python and Tkinter. ASPA is a static analysis tool that performs a structural analysis with standard AST-library and forms an Abstract Syntax Tree of the program under analysis. The AST format makes it possible to analyze the use of programming structures properly. For example, are variables, classes, and functions defined within functions or globally etc.

Python is a flexible programming language that supports object oriented and functional programming but fits also for structural programming. The simplicity of Python is demonstrated by the basic "Hello World" program that consists of only one line 'print("Hello World!")', which is a good starting point for a first programming language. However, the flexibility makes it possible to write programs in many different ways which can lead to programs that are hard to understand and maintain. This issue is typically solved by introducing a company or project specific style guide that defines structural requirements for programs. In practice such style guides are not enforced by programming languages, but separate static analyzers are needed for that which is also the case with Python.

We developed a style guide for our course and implemented respective rules in ASPA. That is, ASPA conducts a structural analysis for a program and checks it against the style guide rules. Figure 1 demonstrates how ASPA detects and reports use of global variables with the following three structural rules. First, there must be a main function in the file. Second, there can be only one function call in the main level and it calls the main function. Third, variables are not defined in the global namespace. Python syntax allows any number of functions, 0 to N, but our first rule requires always one function in the program, the main function. The second rule requires that all statements must be written inside functions except for the call to the main function. In practice these rules simulate the C-program structure, were all statements are within functions. The third rule focuses on variables and covers also their special case fixed values. Namely, Python does not support constants, but we recommend use of global fixed values. This controversy is solved by checking all variables defined in the global namespace for value changes anywhere in the program, and if a change is found, like in the example below, the variable is deemed global and reported to the user. However, if a variable is not modified after initialization, it is considered a fixed value and good programming style.

```
1 global_list = []
2 def main():
3     global_list.append("abc")
4     print(global_list)
5     return None
6 main()
```

ASPA feedback for the program above:

```
Functions, detected:
Line 1: Global variable 'global_list'.
```

Figure 1: Detection of a global variable

Focus of ASPA feedback is to tell students what is wrong and where. This is done by highlighting line number and used variable or function names within violation message, in addition technical terms are avoided when possible. Detected violations are chosen for students and their common problems, e.g. detection of missing parenthesis when creating an object or calling file handle's *close* method, which is not always the case in professional linters and style checkers as their target users are more advanced programmers.

Altogether ASPA includes 41 programming style rules. The rule set evolves as needed and after the first version was released we have added two variations of global variables in the rule base. Namely, some programs use classes as global variables so we added a rule that checks whether a member variable of an object is used, or if the class itself is used to pass values. And as students started using function attributes as global variables, it was added as a violation of good programming style in ASPA, too.

All the violations are not equally important, and thus they have been classified in three categories: errors, warnings, and notes. Errors, like use of global variables, always require a fix. Warnings are estimated in the context of all the errors, warnings, and notes, and are less severe. Notes are typical beginner mistakes, e.g. instead of returning variable *sum*, a string "sum" is returned by having quote marks around the variable name. The general idea of ASPA is always the same - if ASPA reports something, the student should check that line if it really is what the student meant.

Same reporting approach is used with false negatives and false positives, which are always possible with static analyzers. However, in context of CS1 course we decided to minimize false negatives and avoid student complains about undetected violations by reporting a

violation if there seems to be a violation. This leads to small amount of false positives, which are often in gray area between good and bad style and therefore it is better for student to check detected violation anyway. For example, creation of a function called *read* and calling function without exception handling causes ASPA to detect this as a built-in read function call and reporting missing exception handling in file operation.

The style rules have been classified in six packages in ASPA: Basic commands, Functions, File handling, Data structures, Library use, and Exception handling. Functions are the basis for the program structure and students can start using ASPA after they are introduced in the course lectures. After that the next four lectures each introduce a new programming style package. The rule packages make it possible to avoid irrelevant feedback, e.g., in minimal programs with all code in the main level in the last lectures of the course. This way we can benefit from the simplicity of Python in the small programs, and increase the role of style guide as programs get longer and more complex. The ASPA graphical user interface (GUI) is shown in Figure 2: the structures to analyze are selected with the check-buttons on the left, and a file or a folder can be selected for evaluation with the buttons on the top. Selecting a folder makes it possible to check a program consisting of multiple files. The GUI has Finnish and English versions and the language can be chosen in settings.

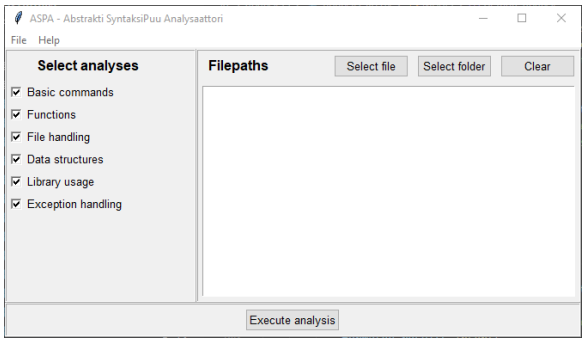

Figure 2: ASPA graphical user interface

The students get ASPA as a single Python file that they can run on their own computer. The staff uses ASPA as a starting point when evaluating the projects and electronic exam programs to assure consistent analysis of the code. Because of students' local use, ASPA is intentionally not sending any analysis data from student computer to staff members. A more detailed description of ASPA and its development can be found at [19], and the program is available at [20].

3 THE EMPIRICAL STUDY

In this Section, we describe our empirical study including goal, research questions, study design, execution, and the data analysis protocol.

3.1 Goal and Research Questions

The *goal* of this study was to empirically validate ASPA with the *purpose* of evaluating acceptance of tool, the perceived ease of use and the perceived usefulness. The *perspective* is that of the students, who are willing to assess the feasibility of using the tool during their basic programming activities as a constructive help and continuous learning method.

Based on our goal, we defined the following three Research Questions (RQs):

RQ$_1$ Which is the attitude toward using ASPA?
RQ$_2$ Which is the perceived ease of use of ASPA?
RQ$_3$ Which is the perceived usefulness of ASPA?

The first research question (**RQ$_1$**) provided us an idea about the feeling the students had about adopting a new tool in their course. We wanted to investigate if the students decided to adopt the tool, if they used it for all the course activities or if they abandoned its usage after certain time. This response helped us to improve our approaches with the students and the process for further new tools adoption. **RQ$_2$** allowed us to investigate the perceived ease of use of ASPA. Students provided their opinion that we compared with the period of usage, while with **RQ$_3$**, we investigated the perceived tool usefulness. We focused on the issues that can compromise the perception such as course difficulty, or the tool interaction. These RQs helped us to validate ASPA, to identify issues we can solve and some further direction to improve the tool and see if students accept AST-based tool to report their coding convention violations.

3.2 Study Design

3.2.1 Survey Design. The survey was composed of the following main sections according to our RQs:

(1) *Participant background.* We profiled the participants via questionnaire in order to describe their background in terms of skills and experience as reported in Table 1.
(2) *Attitude Toward Using.* We needed to evaluate the attitude of students to adopt a tool to perform specific tasks. We collected information about 1) how many students decided to adopt ASPA and 2) how many times they used it. Moreover, we investigated the reasons for not choosing to use the tool among the ones who did not adopted it. The answers were provided via multiple choice questions.
(3) *Perceived Ease of Use.* In this part of the survey, we collected information about the perceived ease of use regarding ASPA. We asked to the students to rank how easy they found the tool adoption. The ranking was based on a 5-point Likert scale, where 1 means absolutely not and 5 absolutely yes.
(4) *Perceived Usefulness.* In the last part of the survey, we investigated how useful the students perceived ASPA. In particular, we asked the students to rate some statements about the tool, such as the course difficulty, and the tool interaction. The rating was based on a 5-point Likert scale, where 1 means strongly disagree and 5 strongly agree.

Survey questions for perceived ease of use and usefulness in sections 3 and 4 are adopted from Technology Acceptance Models (TAM) [9, 30]. Moreover, at the end of the survey we asked the students for feedback about ASPA that can be useful in the tool validation and in adding new features in the future. We also compared the grades for students downloading and not downloading ASPA to see if there was a difference between these groups.

Table 1: Students' profile questionnaire

Questions	Type
1. Do you have previous degrees? a) Undergraduate b) Bachelor's degree c) Master's degree d) Other, what (open field)	Multiple choice*
2. What is the goal of your current studies? a) Bachelor's degree b) Master's degree c) Adult education and training d) Open university cooperation e) Upper secondary school cooperation f) Other, what (open field)	Multiple choice
3. What is the major subject of your studies? a) Energy Technology b) Business Administration c) Chemical Engineering d) Mechanical Engineering e) Computational Engineering and Analytics f) Electrical Engineering g) Software Engineering h) Industrial Engineering and Management i) Environmental Engineering	Multiple choice
4. What is the year of your studies. a) Bachelor 1st year b) Bachelor 2nd year c) Bachelor 3rd year d) Bachelor 4th year or more e) Master 1st year f) Master 2nd year g) Master 3rd year or more	Multiple choice
5. What was your programming knowledge before this course? a) I did not know how to program b) I knew basics of programming c) I knew how to program	
6. I have programmed with following programming languages.	Open question
7. I have studied programming a) In primary school *[In Finland 7-15 years]* b) In secondary education e.g. upper secondary school c) In tertiary education e.g. universities d) On my own, self study e.g. online course e) Other, what (open field)	Multiple choice*
8. Amount of previous studies in ECTS. 1 ECTS is about 27 hours university studies or a single course in upper secondary school a) 1-2 ECTS b) 3-4 ECTS c) 5 ECTS or more	Multiple choice
9. Do you have working experience? Name job positions, working time as months, and used programming languages	Open question
10. Other programming experience, e.g. hobby Describe in short what you have done and how much.	Open question
11. For following statements, select the most suitable options for you. (1 - definitely not ... 5 - definitely yes) a) "I am interested in programming" b) "Completing this course this year is very important to me".	Likert scale
12. Open feedback	Open question

* means that the students can selected more than one answer

3.2.2 Participant Selection. ASPA was developed for CS1 with Python, so we adopted it in the LUT University CS1 course organized by the software engineering degree programme. In the fall 2020 516 students registered to this course and everyone was encouraged to use ASPA. In total 375 students downloaded ASPA from online learning platform (Moodle) and 236 answered a feedback questionnaire.

Participation to the study was voluntary. No extra grades or other types of benefits were provided to the students who used ASPA. However, one extra point for programming assignment was promised, and granted, for those who answered the feedback questionnaire. The maximum points for programming assignments category was 60. Answering the questionnaire did not require usage of the tool, because it was covered by the first question (see Table 2). Moreover, before the participation, students were informed on the study goal.

3.2.3 Tasks. Due to the stand-alone nature of ASPA, the first step was to download it from course online learning platform Moodle. Once downloaded, students were able to run ASPA as any other Python program. The main GUI is shown in Figure 2, from which the student can select which analyses are performed and which file, or files, are analysed. The analyses are selected with checkboxes, and file(s) are selected with Select file or Select folder buttons, respectively, using operating system's standard dialogues. If a folder is selected, every Python file in the folder is analysed. The selected analyses are performed with Execute analysis button, and the results are shown in a text box and written to a text file. Students can return to the main page by pressing Return button, which restores the analyses and files from the previous time to support iterative analysis and editing. The list of selected files can be cleared with Clear button.

To participate in the study students had to answer the feedback survey for which a link was sent via email at the beginning of December 2020.

3.3 Study Execution

The empirical study started when the course started and ended when the feedback survey was closed, so it lasted for four months from **31st of August 2020** to **31st of December 2020**. A background survey was sent to students in early September. The survey had a question branching in questions 2 and 5. In question 2, if student's current study degree was c, d or e, i.e., Adult education and training, open university cooperation or upper secondary school cooperation, questions 3 and 4 were skipped. Similarly in question 5, if student did not have prior programming knowledge, i.e. answer a, questions from 6 to 10 were skipped.

On the fifth week of the course ASPA was introduced to the students in the lecture, and on the sixth week it was available for use. At that point programming assignments focused on functions and were advanced enough for feedback on the program structure. The students could download ASPA from the course platform Moodle, which also counted the downloads. Students were encouraged to use ASPA and it was also brought up that staff would use it to analyze the course projects and exams. The students were able to ask for help both in online and classroom exercises, and in the course online discussion forum.

The course assistants used ASPA to analyse the course projects. If any violations were detected, the ASPA report was included in the grading feedback.

After the final deadline for the course project had passed, the feedback survey was sent to every student on the course. One week later, a remainder email was sent for those who did not answer during the first week. The survey was closed at the end of the

Table 2: The survey design

RQ	Questions	Type
RQ_1	**1. Did you download ASPA during the course?** a) Yes, I downloaded and used it to assist me with assignments b) Yes, I downloaded but did not use it to assist me with assignments c) No, I did not download it	Multiple choice
RQ_1	**2. Why didn't you download ASPA tool?** a) I didn't know about it b) I didn't need help, because I was able to do assignments without it c) I didn't need help, because I dropped the course d) I didn't find it e) Other reason (open field)	Multiple choice*
RQ_1	**3. Why didn't you use ASPA tool?** a) I didn't need help with assignments b) I was not interested in using an extra tool c) I didn't feel that it would help me with assignments d) I found it too hard to use it e) Technical problem (open field) f) Other reason (open field)	Multiple choice*
RQ_1	**4. Did you analyze the course project with ASPA** a) Yes, once b) Yes, 2-3 times c) Yes, many times d) No I didn't use it with course project	Multiple choice
RQ_1	**5. Estimate in how many weekly assignments you analyzed with ASPA** Range 0-36 *(there were 36 assignments after ASPA was introduced)*	Integer number
RQ_2	**6. Do you find ASPA easy to use for** (Likert 5 scale from Very hard to Very easy) a) Selecting which analysis to perform b) Selecting files for analysis c) Understanding the results	Likert scale
RQ_3	**7. Rate the following statements with scale Strongly disagree - Strongly agree?** (Likert 5 scale) a) The course would have been more difficult to complete without ASPA b) ASPA improved my grade for the weekly assignments c) ASPA improved my grade for the course project d) ASPA increased my learning in the course e) I often became confused when I used ASPA f) Interacting with ASPA was often frustrating g) I found ASPA cumbersome to use h) ASPA provided helpful guidance in the most important aspects of the course i) Interacting with ASPA was fun	Likert scale
RQ_3	**8. Was ASPA useful while doing course project or weekly assignments?** If yes explain how. If not explain why it was not useful.	Open questions
RQ_3	**9. Did ASPA work incorrectly or was it misleading?** For example, did you notice that ASPA did not detect a style violation or feedback was misleading.	Open questions
RQ_3	**10. Would you like to have some new feature in ASPA or would you change any of the current features?** If yes, which feature?	Open questions
All RQs	**11. Open feedback about ASPA or anything related to it**	Open questions

* means that the students can selected more than one answer

year 2020. Question order and branching were designed so that everyone in the course was able to answer the survey regardless of their ASPA usage. The first question, shown in Table 2, covered downloading and using ASPA in general level and based on the chosen answer questionnaire branched in three. If the answer was *a) Yes, I downloaded and used it to assist me with assignments*, all questions from 4 to 10 were included. If the answer was *b) Yes, I downloaded but did not use it to assist me with assignments*, questions 3, 9, 10 and 11 were included. And if the answer was *c) No, I did not download it*, only questions 2 and 11 were included. All the survey questions, except for open questions 9, 10 and 11, were mandatory.

3.4 Data Analysis

Two authors manually produced a transcript of each survey answer and then provided a hierarchical set of codes from all the transcribed answers, applying the open coding methodology [32]. The authors discussed and resolved coding discrepancies and then applied the axial coding methodology [32].

Nominal data was analyzed by determining the proportion of responses in each category. Ordinal data, such as 5-point Likert scales, was not converted into numerical equivalents since using a conversion from ordinal to numerical data entails the risk that any subsequent analysis will yield misleading results if the equidistance between the values cannot be guaranteed. Moreover, analyzing each value of the scale allowed us to better identify the potential distribution of the answers. Open questions were analyzed via open and selective coding [32]. The answers were interpreted by extracting concrete sets of similar answers and grouping them based on their perceived similarity.

3.5 Verifiability and Replicability

To enable full verifiability and replicability, the complete raw data is available in our online appendix[1].

[1] https://figshare.com/s/7a6555783163lff6bfa2

4 RESULTS

In this Section, we report the obtained results in order to answer our research questions.

4.1 Student's Profiling

The profiling survey was answered by 424 students, 82% of the course participants, and the vast majority of the them, more than 80%, were undergraduate and enrolled in the bachelor degree program at the first year. Moreover, the students were mainly novice in programming but really interested in it. The complete results are reported in the replication package [1].

4.2 Attitude toward using ASPA (RQ_1)

The feedback survey was answered by 236 students and 205 of them downloaded and used ASPA to assist them with assignments, while 18 downloaded but did not use it. Only 13 did not download ASPA, as shown in Table 3.

Table 3: Attitude toward using ASPA (RQ_1)

Survey question 1 answers	#	%
Yes, I downloaded and used it to assist me with assignments	205	86.9%
Yes, I downloaded but did not use it to assist me with assignments	18	7.6%
No, did not download it	13	5.5%

Among the students who did not download or downloaded but did not use ASPA, we investigated the reasons for this. Students did not download the tool mainly because they felt confident to be able to complete the assignments without the tool (61.5%, Table 4). Some students did not know about the tool, 38.5%, and some could not find it, 15.4% (Table 4).

Table 4: Attitude toward using ASPA - Why not download (RQ_1)

Survey question 2 answers	#	%
I didn't know about ASPA	5	38.5%
I didn't need help, because I dropped the course	0	0%
I didn't need help, because I was able to do assignments without it	8	61.5%
I didn't find ASPA tool	2	15.4%
Other reason, open field	0	0%

The students who did not use the tool (Figure 3) were not interested in using an extra tool (44.4%) or they did not consider it helpful with the course assignments (27.8%). 22.2% reported that they had a technical problem and thus could not use it.

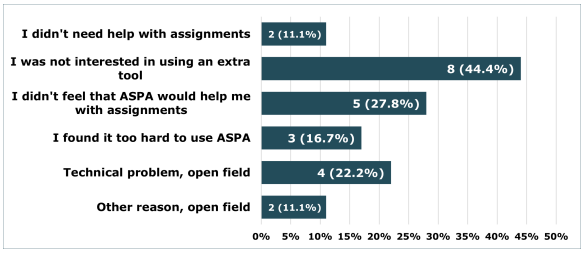

Figure 3: Attitude toward using ASPA - Why not use (RQ_1)

In total 205 students downloaded and used ASPA, either in weekly assignment, in course project or in both. In the course project 104 students (50.7%) used it more than 3 times, and 37.1% used it 2-3 times. The remaining students (12.2%) used the tool at most once as shown in Figure 4.

On average the students used ASPA for 10.90 weekly assignments, and some students used it for all the 36 assignments after the tool introduction, see Table 5.

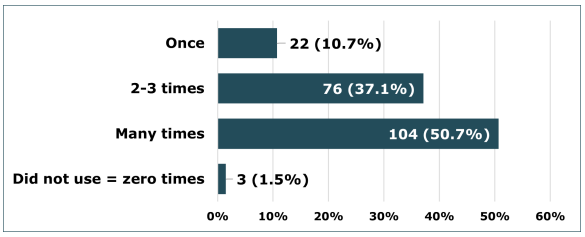

Figure 4: Attitude toward using ASPA - Use while doing the course project (RQ_1).

Table 5: ASPA usage in weekly assignments (RQ_1)

Min	Max	Avg	Sum	Std. Dev.
0	36	10.90	2,241	9.30

4.3 Perceived ease of use (RQ_2)

We asked to the students to evaluate the perceived ease of use of ASPA by considering the three main actions the students did while using it:

- *Selecting analyses*, i.e. clicking suitable checkboxes on left side of the GUI;
- *Selecting files*, i.e. clicking *Select file* or *Select folder button* on the top right corner of GUI and then selecting file/folder via operating system's standard file-dialog;
- *Understanding the results*, i.e. did they understand the feedback the tool gave.

As shown in Table 6 and Figure 5, majority of the respondents considered all actions *very easy* to use (at least 106 out of 205) or *easy* to use (at least 76 out of 205). Only few students, 3 or less, perceived the use hard or very hard. As for the previous RQ, few (maximum 14, 7%) of the respondents were not able to provide a clear opinion.

Table 6: Perceived ease of use of ASPA (RQ_2)

Survey questions	Very hard	Hard	Not hard or easy	Easy	Very easy
Selecting analyses	0	3	5	78	119
Selecting files	2	3	11	76	113
Understanding results	0	0	14	85	106

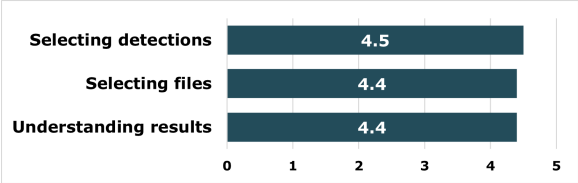

Figure 5: Perceived ease of use of ASPA (RQ_2), averages

4.4 Perceived usefulness (RQ_3)

We formulated four questions regarding the perceived usefulness of ASPA and all the 205 students who used the tool replied to the questions (Table 7). Answer averages are shown in Figure 6. More than 60% (agree or strongly agree) of the students recognized the usefulness of ASPA because they perceived it had reduced the course difficulties. 27% did not provide a clear opinion, and only 11% disagreed with the usefulness of ASPA.

ASPA helped 61% of the students to complete the weekly assignments, while only for the 10% the help perception was not recognizable. Also for this questions, few of them (29%) were not able to provide a clear evaluation. A positive evaluation was also provided considering the help during the entire course, with a percentage higher than limiting the evaluation only to the weekly assignments (81%).

ASPA also helped the student in better understanding the course goal and the relative task for more than 60% of them, while 29% did not provide a clear evaluation. Only 10% considered the tool not useful. This positive perception is confirmed also looking at the usefulness of the tool as guidance (50%).

The last four questions are related to the happiness in using the tool. The vast majority (86%) of the students did not find it frustrating to use, but the perception of fun was not clear (53%) and only 30% positively answered. Moreover, the 88% of the students found the tool clear.

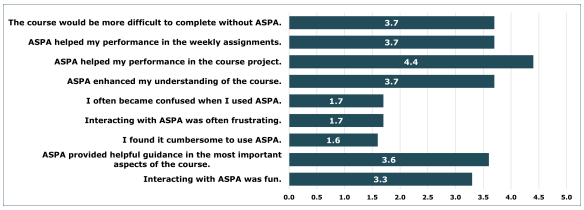

Figure 6: Perceived usefulness of ASPA (RQ_3)

Open questions in the feedback questionnaire had 203, 120, 92 and 92 non-empty answers, respectively. After open coding procedure answers were coded and answer had 1-4 distinct codes. In the question regarding the open feedback (question 11), same codes as in previous open questions were used if possible. In total 68 answer codes were included in previous categories and 9 new codes were selected. The most frequent codes of each question are shown in Figures 7, 8 and 9, in addition, answers from open feedback with same codes are indicated with an orange colour.

The open coding for question regarding ASPA usefulness (question 8) is shown in Figure 7 and the most frequent code, with 69 occurrences and 12 from open feedback, was *Course project* followed by *Careless mistakes* (45/2) and *Structures* (33/1). However, when including open feedback *General benefit* (11/28) become the third frequent category. Categories *Course project* and *Weekly assignments* were used when answer explicitly stated benefit for performance in these tasks, similarly *Structure*, *File handling* and *Exception handling* indicate explicit statement that ASPA assisted with these coding elements. On the other hand, *Careless mistake* and *Style Violation* indicate student statement that these were easier to fix or detect with ASPA and similarly *Find violation* indicate that generally detection of violations working well according to a student. *Give certainty* indicate explicit statement of getting assurance that the code was done properly when checking it with ASPA, *Learning* was used to for answers with statements about benefits for learning and *General benefit* includes answers where benefit or usefulness was explicitly stated but it topic was not specified. Finally, *Extend submission platform* was used for answers with mentions that ASPA had either extended or had better checks compared to the online submission platform, and *Other* includes all other codes with less than 10 occurrences.

Open coding for question regarding misleading with ASPA (question 9) is shown in Figure 8 and the most frequent code, with 85 occurrences, was *No errors*, which is relatively self-explanatory, i.e. student explicitly stated that he or she did not notice any anomalies or errors when working with ASPA. Misleading message, with 14 occurrences, was the second common code and it includes answers where meaning of message was stated to be misleading or meaningless, i.e., student mentioned that message did not help. *Course practice* was used when student complained about existence of a check, which checks any course practice, e.g., return-statement at the end of each function. *Usage problem* contains answers about either technical or other problem while using ASPA and *Other* contains rest of the codes with less than 8 occurrences.

The open coding related to the new features the student considered useful to have in ASPA (question 10) is shown in Figure 9 and the most frequent code, with 54 occurrences, was *Nothing to add*, which include all explicit statements that student would not add or change anything from ASPA. *File selection* indicate that student requested a change to the file selection, e.g. drag-and-drop feature and saving previous folder path when opening a filedialog. *User manual* include answers with request for better guidance or tutorial to use ASPA. Then six answers included request for feature which already existed, marked as *Existing feature*, and finally, *Other* contains rest of the codes with less than 6 occurrences.

The most frequent new codes for the last question (open feedbacks) are shown in Figure 10, and the most frequent code, with 30 occurrences, was *Good tool*, which indicate explicit mention of ASPA being a good tool, but no specification of how or why. Codes *Easy to use* and *Convenient* contains answers where these usage aspects were explicitly mentioned, and *More advertisement* include request for advertising ASPA more to the student, e.g., to make them notice that tool exist and it could be used. Finally, *Other* contains rest of the codes,e.g., *Suitable for novices* and *Fast*, with less than 5 occurrences.

Finally, we compared the grades for the students who did and did not download ASPA. The grading scale was from 0 to 5 with 1 as the lowest passing grade. Comparison results, with all grades

Table 7: ASPA perceived usefulness (RQ_3)

Question	Perceived Usefulness	#	Median
The course would have been more difficult to complete without ASPA	Strongly disagree	5	4
	Disagree	18	
	Do not agree nor disagree	55	
	Agree	90	
	Strongly agree	37	
ASPA improved my grade for the weekly assignments	Strongly disagree	7	4
	Disagree	14	
	Do not agree nor disagree	59	
	Agree	77	
	Strongly agree	48	
ASPA improved my grade the course project	Strongly disagree	0	4
	Disagree	8	
	Do not agree nor disagree	11	
	Agree	86	
	Strongly agree	100	
ASPA increased my learning of the course	Strongly disagree	3	4
	Disagree	18	
	Do not agree nor disagree	59	
	Agree	80	
	Strongly agree	45	
ASPA provided helpful guidance in the most important aspects of the course	Strongly disagree	6	4
	Disagree	19	
	Do not agree nor disagree	64	
	Agree	87	
	Strongly agree	29	
Interacting with ASPA was fun	Strongly disagree	4	3
	Disagree	23	
	Do not agree nor disagree	108	
	Agree	56	
	Strongly agree	14	
Interacting with ASPA was often frustrating	Strongly disagree	109	1
	Disagree	68	
	Do not agree nor disagree	19	
	Agree	8	
	Strongly agree	1	
I often became confused when I used ASPA	Strongly disagree	89	2
	Disagree	92	
	Do not agree nor disagree	20	
	Agree	43	
	Strongly agree	1	
I found ASPA cumbersome to use	Strongly disagree	108	1
	Disagree	73	
	Do not agree nor disagree	16	
	Agree	7	
	Strongly agree	1	

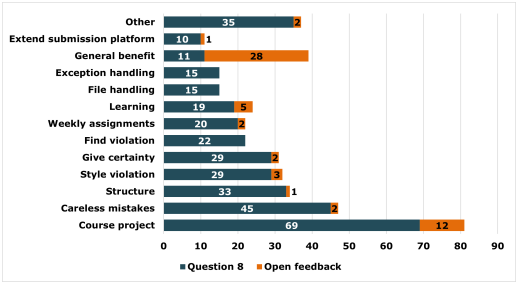

Figure 7: ASPA usefulness during the usage - open coding category frequencies (RQ_3)

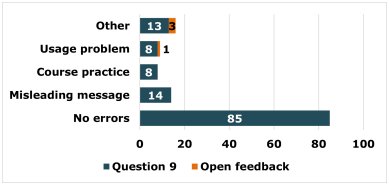

Figure 8: ASPA misleading - open coding category frequencies (RQ_3)

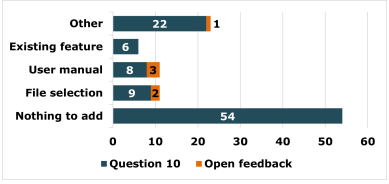

Figure 9: ASPA new features need - open coding category frequencies (RQ_3)

and only with passing grades, are shown in Table 8. After removing three students with missing data, the grade comparison was done for all the 515 students, regardless of their participation for survey. In total 375 downloaded ASPA, while 140 did not, and 324 vs. 20 students passed the course from these categories respectively.

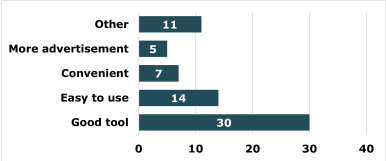

Figure 10: ASPA other feedbacks - open coding frequencies of new categories (RQ_3)

Mann-Whitney U test was used to compare the grades for the programming assignments. H_0 was rejected in all but one category, only passing examination grades, meaning that alternative hypothesis H_1, *downloading ASPA has a positive effect on grade*, is true.

Table 8: Average grades of students downloading and not downloading ASPA, and statistical significance of difference

	Weekly assignments	Course project	Examination	Final grade
All grades				
Downloaded (N=375)	3.38	2.92	2.45	2.90
No download (N=140)	0.69	0.48	0.44	0.39
p-value	< 0.001	< 0.001	< 0.001	< 0.001
Only passing grades				
Downloaded (N=324)	3.70	3.28	2.75	3.36
No download (N=20)	3.15	2.75	2.55	2.75
p-value	0.036	0.034	0.191	0.013

H_0: Downloading ASPA does not have a positive effect on grade.
H_1: Downloading ASPA has a positive effect on grade.

5 DISCUSSION

The results achieved from our empirical study revealed a number of insights that may lead to concrete implications for the software engineering education community, and that we discuss hereafter.

ASPA survey was answered by 236 students and it was frequently used during the assignments and the course. This good number of valid data points allowed us to derive some promising key points to validate the tool and to plan further actions to improve it. Moreover, the vast majority of the students (87%) who adopted the tool were novice to programming. This is a very important aspect because having participants who did not have any experience with similar tools and who could not program before the course avoids the prior knowledge bias that could affect the evaluation.

ASPA is ease to use. The first key outcome of the empirical validation reported is that ASPA is easy to use. The ease of use covers the selection of the analyses to perform, selection of files to analyze, and understanding the results and the feedback the tool gives.

ASPA is a useful tool. This second key outcome was reported by the vast majority of the respondents (more than 80% an average). The students perceived it less difficult to complete the assigned tasks with the tool. This is particularly evident considering the positive evaluation on the course project: students found the tool a great helper to improve their course project grade. In addition to student reported perceived usefulness, a statistical comparison of grades corroborated usefulness of ASPA. This initial analysis, however, requires a more thorough statistical analysis in the future. The student reported perceived usefulness might be a result of ASPA providing helpful guidance to the students for the most important and crucial tasks, and helping to better understand the course goals. The helpful guidance was designed by giving clear reports to help students to understand what was wrong in their code. This was done e.g. by avoiding too technical terms, highlighting location of violation with line number and including variable or function names when possible as shown in Section 2.

Moreover, the interactions with the tool were positively judged by the students. This observation highlights a positive synergy between ASPA and the students, and created a fertile ground for the tool adoption. In general AST proved a suitable technology for giving semiautomatic feedback to students and to support learning. However, it is important to have coding conventions that can be implemented in the selected tool and refined continuously to support evolving course needs.

The empirical validation of the tool provided also some suggestions on how to improve ASPA. These include having more fine-grained reports and request for more checks to cover also other aspects than structural and style violations.

6 THREATS TO VALIDITY

In this Section, we will introduce the threats to validity, presenting the different tactics adopted to mitigate them.

Construct Validity. The empirical study and the survey design, the execution, and the analysis followed a strict protocol, which allows replication of the survey. The open questions were analyzed qualitatively, which is always subjective to some extent, but the resulting codes were documented. However, we identified some social threats to construct validity regarding the participants selection. Participants may act differently than they do during the empirical study [31]. To mitigate this threat we clearly explained the purpose of the study and asked the participants to answer preliminary questions that allowed us to profile their experience. We also informed them that the survey was anonymous and the collected data was analyzed without considering their identities.

Internal Validity. A non rigorous experiment design can affect negatively the results [31]. To deal with this threat, we designed and conducted the empirical study according to the guidelines. We formulated the questions in a way that we have only direct questions requiring as little interpretation as possible, to avoid misunderstandings that would lead to meaningless answers. We performed a validation process for the designed survey to detect any inconsistency or misunderstanding before the execution.

External Validity. Threats to external validity are conditions that limit our ability to generalize the results [31]. The survey can only reveal the perceptions of the respondents which might not fully represent reality. However, we gave to the students the possibility to request additional information regarding unclear or imprecise statements. The responses were analyzed and quality-checked by a team of four researchers.

Conclusion Validity. This threat is concerned with the extent the data and the analysis are independent of the specific researchers [31]. The coding process regarding the open questions was performed separately by two researchers. Then, we discussed the results until

the consensus was reached. We focused on terms standardization in order to reduce the bias of a subjective process as much as possible.

7 RELATED WORK

In the computer science education discipline, the tools to visually aid understanding program structures, automating assignment collection, and running an analysis on the source code assignments is a long-standing topic. For example, the first tools for assessing the submitted code were released in the 1960's [17], with fully automated code graders first being implemented in the 1980's [13]. In 2005 over sixty different projects for helping the students to understand programming tasks were identified [17], and in general a systematic literature review compiled in 2018 [21] identified 265 studies, which focused on tool utilization as a part of the introductory programming courses.

Sorva et al. [29] studied 46 visualisation tools and concluded that there are multiple visualisation systems built to solve various problems the CS0/1-level students might encounter. However, these systems tended to be research prototypes with a short life span, relatively scarce resources for development, and rather limited scale of follow-up research on the long-term effectiveness, although there are some tools which defy this definition. The student-assisting visual programming environments such as Alice [7] which was first released in 1998 and included a direct connection to very popular video game franchise Sims, or ViLLE [15], which has existed in several forms since 2005 and has been adopted to several national-scale projects, can be mentioned as a few such examples.

However, there is an argument that the visualization by itself is not very effective [24]. The needed experience and skills are formed by designing and developing functional code, not just by purely observing the visualized presentation of source code [27], although the motivational aspects of giving additional clues and presentation on how the code behaves has its benefits. For example Higgins et al. [13] studied CourseMarker [10] in varying programming courses. When they combined CourseMarker's feedback with possibility to submit three times instead of one, 94% of the students got consistently better grades. Total improvement from first to last submission was 63% on average and the tool saved hundreds of working hours from academic staff without lowering the feedback quality. Similarly, Croft and England [8] conducted a study showing that Moodle CodeRunner can be used very successfully at introductory course with Python 3 and at the first object-oriented course with C++. Both courses had students with varying backgrounds and prior programming knowledge was not required. Usage of the CodeRunner increased average exam grade by 7% and pass rate by 10%. In addition student satisfaction to assessment and feedback was increased by 7%.

In general, source code quality analysis is an interesting venue to improve student performance [3]. While learning the programming language fundamentals it is not unusual for the students to overtly rely on the structures and patterns they know [28], because they do not possess the experience to understand when their solutions are overtly complex, approach the problem from the wrong direction, or simply start to pick up detrimental programming habits [22]. To provide a tool to identify and counteract this problem in their first-year functional programming course Gerdes et al. [12] created non-test-based strategies and program transformation to functionally assess students' Haskell programs, as well as, give explained correction guidance. Their results argue that the test-based method has various problems, such as test coverage and missing detection for good conventions. They used methods like strategy language and lambda calculus to check correctness, good conventions and design of the program, while reducing the teacher's workload from manually advising the revision work of 94 programs to 8 programs. Similar results for feature extraction-based approaches has also been reported by Carter et al. [5], on their study concerning what aspects of the meta-data are valuable for assessing student skills and progress.

From the student's perspective, these types of automatic grading tools such as zyBooks have been utilised to enable for example an automated assessment, an effective usage of multiple submissions per assignment, and occasionally code templates which guide the assignments towards the proper programming structures [1]. Allen et al. [2] combined these features with many small programs (MSP) approach, which led to reduced student's stress and better scores on subsequent CS course. Similarly, Garg and Keen [11] studied a tool called Earthworm, which give suggestions to decompose function to reduce complexity, basically refactoring feedback to improve the quality of the student-submitted source codes.

However, in the application with entry-level programming courses the application or enforcement of quality assessment tool can also cause negative side effects. For example Keuning et al. [18] observed that too advanced feedback can cause frustration and confusion on the novice programmers. Their study reported that novice programmers were unable to understand what the development environment, or the quality assessment aides, were instructing making the revisions and fixing the programs difficult, and that too restricting grading and tutoring systems may fail to give relevant instructions [6] because they might focus on the wrong concepts like advanced structures, or repository management. This problem with the programming environments providing unusable support for the novice programmers, was mentioned several times also in the systematic mapping study of CS1 course issues, conducted by [23].

8 CONCLUSION

This paper reports an empirical validation of a static analyser tool that supports learning and continuous feedback on programming courses. We designed and conducted an empirical study among 236 students enrolled in the basic programming course in order to evaluate the attitude toward using the tool, the perceived ease of use, and the perceived usefulness. Our key results indicate that ASPA is a good helper for the entire course and especially the student's programming assignments, and it also helps to improve the students' grades. Our future research agenda includes a series of external replication in different universities and we already identified and started planning some of them for the next academic year.

REFERENCES

[1] Joe Michael Allen, Frank Vahid, Kelly Downey, and Alex Daniel Edgcomb. 2018. Weekly Programs in a CS1 Class: Experiences with Auto-graded Many-small Programs (MSP). In *ASEE Annual Conference & Exposition*. ASEE Conferences, Salt Lake City, Utah, 13. https://doi.org/10.18260/1-2--31231

[2] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An analysis of using many small programs in cs1. In *Technical Symposium*

on Computer Science Education. ACM, Minneapolis, MN, USA, 585–591. https://doi.org/10.1145/3287324.3287466

[3] Raul Andrade and João Brunet. 2018. Can students help themselves? An investigation of students' feedback on the quality of the source code. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, San Jose, CA, USA, 1–8. https://doi.org/10.1109/FIE.2018.8658503

[4] Brett A. Becker and Keith Quille. 2019. 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. In *Symposium on Computer Science Education*. ACM, Minneapolis, MN, USA, 338–344.

[5] Janet Carter, Kirsti Ala-Mutka, Ursula Fuller, Martin Dick, John English, William Fone, and Judy Sheard. 2003. How shall we assess this? In *Working group reports from ITiCSE on Innovation and technology in computer science education*. ACM, New York, NY, USA, 107–123. https://doi.org/10.1145/960875.960539

[6] Sammi Chow, Kalina Yacef, Irena Koprinska, and James Curran. 2017. Automated data-driven hints for computer programming students. In *Conference on User Modeling, Adaptation and Personalization*. ACM, Bratislava, Slovakia, 5–10.

[7] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of computing sciences in colleges* 15, 5 (2000), 107–116.

[8] David Croft and Matthew England. 2020. Computing with CodeRunner at Coventry University: Automated summative assessment of Python and C++ code. In *4th Conference on Computing Education Practice 2020*. ACM, Durham, United Kingdom, 1–4.

[9] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (1989), 319–340. https://doi.org/10.2307/249008 Publisher: Management Information Systems Research Center, University of Minnesota.

[10] Eric Foxley, Colin Higgins, Tarek Hegazy, Pavlos Symeonidis, and Athanasios Tsintsifas. 2001. The coursemaster cba system: Improvements over Ceilidh. In *Proceedings of the 5th CAA Conference*. Loughborough University, Loughborough, United Kingdom, 12. https://hdl.handle.net/2134/1809

[11] Nupur Garg and Aaron W Keen. 2018. Earthworm: Automated Decomposition Suggestions. In *Koli Calling International Conference on Computing Education Research*. ACM, Koli, Finland, 1–5. https://doi.org/10.1145/3279720.3279736

[12] Alex Gerdes, Johan T Jeuring, and Bastiaan J Heeren. 2010. Using strategies for assessment of programming exercises. In *41st Technical symposium on Computer science education*. ACM, Milwaukee, Wisconsin, USA, 441–445. https://doi.org/10.1145/1734263.1734412

[13] Colin A Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. 2005. Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing (JERIC)* 5, 3 (2005), 5–es.

[14] Tony Jenkins. 2002. On the difficulty of learning to program. In *Annual Conference of the LTSN*. LTSN Centre for Information and Computer Sciences, Loughborough, United Kingdom, 53–58.

[15] Erkki Kaila, M-J Laakso, Teemu Rajala, and Einari Kurvinen. 2018. A model for gamifying programming education: University-level programming course quantified. In *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, IEEE, Opatija, Croatia, 0689–0694. https://doi.org/10.23919/MIPRO.2018.8400129

[16] Jussi Kasurinen. 2006. *Python as a programming language for the introductory programming courses*. Bachelor's thesis. Lappeenranta University of Technology.

[17] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)* 37, 2 (2005), 83–137.

[18] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. 2017. Code Quality Issues in Student Programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (2017-06-28) *(ITiCSE '17)*. ACM, Bologna, Italy, 110–115. https://doi.org/10.1145/3059009.3059061

[19] Roope Luukkainen. 2020. *ASPA: A static analyser to support learning and continuous feedback on the first programming course*. Master's thesis. LUT University.

[20] Roope Luukkainen. 2020. RoopeLuukkainen/ASPA: ASPA proof-of-concept release. Zenodo http://doi.org/10.5281/zenodo.3898125.

[21] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (2018-07-02) *(ITiCSE 2018 Companion)*. Association for Computing Machinery, Larnaca, Cyprus, 55–106. https://doi.org/10.1145/3293881.3295779

[22] Richard E Mayer, Jennifer L Dyck, and William Vilberg. 1986. Learning to program and learning to think: what's the connection? *Commun. ACM* 29, 7 (1986), 605–610.

[23] Rodrigo Pessoa Medeiros, Geber Lisboa Ramalho, and Taciana Pontual Falcão. 2018. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Trans. on Education* 62, 2 (2018), 77–90.

[24] Seppo Nevalainen and Jorma Sajaniemi. 2006. An Experiment on Short-Term Effects of Animated versus Static Visualization of Operations on Program Perception. In *Second International Workshop on Computing Education Research*. ACM, Canterbury, United Kingdom, 7–16. https://doi.org/10.1145/1151588.1151591

[25] Uolevi Nikula, Satu Alaoutinen, Jussi Kasurinen, and Toni Pirinen. 2009. Improving the Technical Infrastructure of a Programming Course. In *International Conference on Advanced Learning Technologies*. IEEE, Riga, Latvia, 374–376. https://doi.org/10.1109/ICALT.2009.52

[26] Uolevi Nikula, Orlena Gotel, and Jussi Kasurinen. 2011. A Motivation Guided Holistic Rehabilitation of the First Programming Course. *ACM Trans. on Computing Education* 11, 4 (Nov. 2011), 1–38.

[27] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. *Computer science education* 13, 2 (2003), 137–172.

[28] Elliot Soloway. 1984. What do novices know about programming. *Directions in Human-Comupter Interaction* (1984).

[29] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)* 13, 4 (2013), 1–64.

[30] Venkatesh, Viswanath, Morris, Michael G., Davis, Gordon B., and Davis, Fred D. 2003. User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly* 27, 3 (2003), 425–478. https://doi.org/10.2307/30036540

[31] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. 2012. *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg, Germany.

[32] Brad Wuetherick. 2010. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. *Canadian Journal of University Continuing Education* 36 (12 2010).