An Exploration of High School Students' Self-Confidence while Analysing Iterative Code

Claudio Mirolo*, Emanuele Scapin

University of Udine, 33100 Udine, Italy

Abstract

A number of studies on novice programming report that loops and conditionals can be potential sources of errors and misconceptions. We then felt the need to engage in a more systematic and in-depth investigation about the teaching and learning of iteration in some representative high schools of our regional area. As a medium-term outcome of this endeavour we expect to get fine-grained insights about the nature of students' difficulties, on the one hand, as well as to identify possible pedagogical approaches to be adopted by teachers, on the other. As a step of this project, we designed and administered a survey composed of a set of small tasks, addressing students' understanding of iteration in terms of *code reading* abilities. After summarising the motivations underlying the choice of the tasklets and the overall structure of the instrument, in this paper we will focus on a particular aspect which has not yet received extensive attention in the computer science education literature. Specifically, we will consider students' perception of self-confidence, in connection with their actual performance in each task, the specific program features, the cognitive demands (procedural vs. higher-level thinking skills), and the use of code vs. flow-charts. A noteworthy result of this analysis is that students' perception of self-confidence is poorly correlated to actual performance in the task at hand. The main implications of our study are twofold, pertaining our understanding of less conspicuous facets of the learning of iteration as well as possible pedagogical strategies to strengthen metacognitive skills.

Keywords

Informatics education, Program comprehension, High school, Iteration constructs, Novice programmers, Metacognition

1. Introduction

Learning to program is a complex, "slow and gradual process" [1]. Students' difficulties are well known and have been extensively investigated for tertiary education (e.g. [2, 3, 4]). Also such basic flow-control constructs as conditionals and loops turn out to be potential sources of several mistakes and misconceptions for novice learners [5, 6]. However, the overall picture as to the teaching and learning of iteration at the upper secondary level is still a bit fragmentary and calls for more systematic study.

In the attempt to make some progress in this direction, focusing on the high school context in our regional area, we are collecting teachers' and learners' insights from a range of perspectives,

Informatics in Schools. A step beyond digital education. 15th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2022, Vienna, Austria, September 26–28, 2022

*Corresponding author.

claudio.mirolo@uniud.it (C. Mirolo); emanuele.scapin@uniud.it (E. Scapin)

0000-0002-1462-8304 (C. Mirolo); 0000-0001-8384-8231 (E. Scapin)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

including subjective perception, instructional practice, ability and challenges to achieve program-related tasks. In particular, we have used an instrument designed to investigate students' ability and perception of self-confidence when analysing small programs based on iteration constructs. In accordance with Izu's et al. program comprehension perspective [7], the proposed test is based on a set of small *code reading* tasks, or *tasklets*.

After summarising the rationale behind the chosen tasklets and the structure of the instrument, in this paper we will mainly focus on a particular aspect which has not yet been deeply investigated in computing education, and particularly at pre-tertiary level. Specifically, we will consider students' perception of self-confidence in connection with their actual performance, the program features, the implied cognitive demands and the use of textual code vs. flow-charts.

Being able to monitor one's "current level of mastery and understanding" while trying to accomplish a task is indeed an important *metacognitive* competence [8], which appears to be particularly crucial in computer science [9]. As pointed out by Brown and Harris [10], "from both psychometric and learning theory perspectives, the accuracy of self-assessment [...] is critical. If self-assessment processes lead students to conclude wrongly that they are good or weak in some domain and they base personal decisions on such false interpretations, harm could be done, even in classroom settings."

The design of the tasklets and of the related questions was guided by the outcome of some preliminary work, namely; (i) A number of interviews asking experienced high-school teachers about the role of iteration in their practice and the related learning issues [11]; (ii) A "pilot" survey addressed to students, including questions about their subjective perception of difficulties as well as three short exercises on basic iteration constructs [12].

The rest of the paper is organised as follows. After mentioning in Sect. 2 some relevant background, Sect. 3 outlines scope, main features and organisation of the developed instrument. Then, Sect. 4 summarises the major findings concerning high school students' perception of self-confidence. Finally, in Sect. 5 we conclude with a few remarks and future perspectives.

2. Background

Starting from the pioneering work on the cognitive implications of programming tasks in the early 1980s, e.g. [13, 2], empirical research has persistently shown that flow-control constructs such as conditionals and loops are frequent sources of errors and misconceptions for novice learners [5, 6, 14], especially when combined into nested constructs [15]. The reasons may be manifold, ranging from lack of problem solving skills to the need for accuracy and strenuous practice. According to Perkins et al. [16] programming is indeed "problem-solving intensive," in fact it requires multiple skills [17], and students may fail to develop a viable model of the underlying *notional machine* [18] or may not be able to see code execution at higher levels of abstraction to infer a program's purpose [19]. Moreover, part of the students' difficulties may also be related to the habits and expectations of both teachers and learners [17].

Here the focus is on students' perceived self-confidence while achieving a task, in fact one of the issues raised by our previous pilot study [12]. Being able to fairly assess the degree of self-confidence in the devised solution is connected to metacognitive skills, that can be generally meant as "one's knowledge about one's cognitive processes" [20]. While extensively

investigated in other fields, e.g. [21], metacognition is recently receiving wider attention also for programming education [22, 23]. Most often metacognitive skills have been explored in connection with students' success in tasks or exams [24, 25, 26] and some educators see pedagogical value in providing learners with opportunities to reflect on their metacognitive awareness under the guidance of the teacher [24, 20]

3. Characterisation of the instrument

The results of our pilot studies focused on iteration involving teachers [11] and students [12], also in light of the issues discussed in the literature on novice programming, raised a number of questions that seemed to us worth further investigation, in particular:

Q1. To what extent are students at ease with a range of features implied by iteration? (E.g., loop conditions, nesting of flow-control constructs, …)

Q2. How does the effort to trace code execution impact on the analysis of more abstract program properties?

Q3. Are students more at ease when using flow-chart or textual code representations of programs?

Q4. To what extent does students' perception of self-confidence correspond to their actual achievement in a task?

It is precisely around the above questions that our investigation instrument has been designed. To this aim, we have first identified the general areas and topics outlined in Table 1, where each area is labeled with the questions Q1–4 it pertains to. Then, we have defined two sets of 6 tasklets, whose scope covers the areas and topics listed in Table 1 in as balanced a way as possible. The size and complexity of a *taskset* are chosen in such a way that students could reasonably complete their work on it in about one hour — i.e. before losing concentration.

Each individual tasklet presents a program based on iteration constructs and asks one or two multiple-choice questions about its related properties. When the question brings into play high-level thinking skills (area A in Table 1), the student is also required to assess in a 4-grade Likert scale her/his perceived level of self-confidence on the provided answer. In addition, the programs of three tasklets may be shown either as textual code or as flow-charts (line FC in Table 1). Finally, this instrument has been made accessible as an online survey, the taskset and the code/flow-chart format of selected tasklets being assigned randomly.

To accommodate for the common practice in the considered high schools, the textual code is Java-like. In the following we will elaborate a little more on each of the areas reported in Table 1. For more details about the individual tasklets, English translations of the four versions (two tasksets × code/flow-chart modes) of the test-survey are available via the links provided in the online appendix http://nid.dimi.uniud.it/additional_material/iteration_tasksets.pdf .

A. Tasklets addressing higher-order thinking skills. This is a central area for our investigation and each tasklet includes at least a question of this type. It covers two broad

Table 1
Areas and topics addressed by the tasklets.

A. Tasklets addressing higher-order thinking skills (Q2, Q4)
   1. Abstraction on the computational model
      a. Equivalence (nested constructs, `for`/`while`, `do-while`/`while` …)
      b. Reversibility
   2. Relationships with the application domain
      a. Completion (condition, expression, statement …)
      b. Functional purpose

B. Tasklets addressing code features (Q1, Q3)
   1. Structural features
      a. Plain loop
      b. Nested conditional
      c. Nested loop
   2. Processing plan
      a. Exit condition
      b. Loop control variable
      c. Downward `for` loop

B. (continued)
   3. Conditions
      a. Simple condition
      b. Composite condition
      c. Boolean expression/variable

C. Tasklets addressing code execution, conceivably via tracing (Q1, Q2)
   1. Output/final state
   2. Number of iterations

D. Tasklets addressing data types (Q1)
   1. Numerical data (only)
   2. Non-numerical data
   3. Array data

SC. Perception of self-confidence (Q4)

FC. Flow-chart versus code (Q3)

categories, concerning abstraction over the computation structure and functional abstraction in connection with some problem domain. To test students' abilities in the former category we ask *equivalence* [27] and *reversibility* [28] questions. The tasklets in the latter include more common types of questions which ask either to choose an appropriate item (e.g., a condition, an arithmetic expression, a statement) to complete a program with a given purpose or to identify the intended purpose of a given program — in fact an instance of the recurrent "Explain in Plain English" theme [19]. As an example, Figure 1 shows an equivalence question with four answer options. Both equivalence and reversibility tasks require students to reason about program behaviour comprehensively, generalising what could be ascertained by tracing code execution for specific input data. The role of reversibility in learning, in particular, dates back to Piaget's work on cognitive development, where it is considered as an indicator of achievement of the concrete operational stage, and according to a neo-Piagetian perspective the learning stages apply regardless of age when approaching new knowledge domains [29]. Thus, reversibility seems to be an appropriate instrument to assess their comprehension in the early stages of learning to program.

B. Tasklets addressing specific code features. The code features listed in Table 1 are easily recognisable in a tasklet by program inspection and are connected with the findings of our pilot work [11, 12], indicating loop conditions and nested constructs as major sources of difficulties — nested loops being seen as the hardest challenge in the learners' subjective perception. Widespread issues and misconceptions regarding nested constructs, in particular, have also been identified in a number of studies [30, 31, 32, 15]. Additional difficulties worth consideration that can be ascribed to loop structures arise in the treatment of loop-control variables, see e.g. [2],

```
int x = m;
int y = n;

while ( x != y ) {

  while ( x < y ) {
    x = x + m;
  }
  while ( x > y ) {
    y = y + n;
  }
}
output( "x = " + x );
```

reference program

```
int x = m;
int y = n;

while ( x != y ) {

  while ( x < y ) {
    x = x + x;
  }
  while ( x > y ) {
    y = y + y;
  }
}
output( "x = " + x );
```

option 1

```
int x = m;
int y = n;

while ( x != y ) {

  do {
    x = x + m;
  } while ( x < y );
  do {
    y = y + n;
  } while ( x > y );
}
output( "x = " + x );
```

option 2

```
int x = m;
int y = n;

while ( x != y ) {

  if ( (x < y) || (x > y) ) {
    x = x + m;
    y = y + n;
  }
}
output( "x = " + x );
```

option 3

```
int x = m;
int y = n;

while ( x != y ) {

  if ( x < y ) {
    x = x + m;
  } else {
    y = y + n;
  }
}
output( "x = " + x );
```

option 4

T1.4 (ii) – *Which option is equivalent to the reference program?* The input requirements are that both the values of `m` and `n` are positive (> 0) integers and two programs are meant to be equivalent if the final states of their executions are always the same, whenever the initial states are the same — and provided the initial states comply with the given input requirements.

Figure 1: Example of equivalence question.

and while dealing with down-counting loops as opposed to more stereotypical up-counting loops, as pointed out in [33].

C. Tasklets addressing code execution. Small problems that can be solved at a low *operational* level by tracing the code execution are quite common, since tracing is deemed to be a basic ability "to build [...] higher-level comprehension skills upon" it [34]. In any case such ability should not be taken for granted; many students struggle, for instance, with tracing loops, especially `while`-loops [35]. Here, however, our main purpose is to address the investigation question Q2: whether code tracing can to some extent support higher-order thinking *in the task at hand*. In [12] we found indeed some cues suggesting that students' performance on more abstract tasks may improve when they are actually led to engage in some careful tracing, but that they tend to elude this effort to check their conjectures about program behaviour. Thus, the idea is to ask questions pertaining to the "abstract" area A, either including or not a previous question that can be answered via tracing, in particular to determine the program output or the overall number of iterations for given input data.

D. Tasklets addressing data types. The covered data types are essentially *numbers*, *booleans*, *characters*, *strings*, and *arrays*. The indexed access to arrays, in particular, can be problematic for novices, especially in connection with iteration — see e.g. the recent work [36, 37].

SC. Perception of self-confidence. Each tasklet requires to reason about a given program by asking at least one question in the area A, and after answering this question students must also

indicate their perceived level of self-confidence in a Likert scale ranging from 1 (not confident at all) to 4 (fully confident). Besides the reasons mentioned in the introduction and the potential pedagogical implications [24, 20], we included this feature since our study [12] suggested that students' perception of difficulty and actual performance in a task tend not to be consistent.

FC. Flow-chart versus code. According to [38], "flowcharts support novice programmers [...] and give guidance to what they need to do next, similar to a road-map." They may also help to identify difficulties and misconceptions [14]. On the other hand, Ramsey's et al. findings [39] seem to indicate that the use of flow-charts may not be natural for students and that working with code often gives rise to better performances. Thus, we tried to investigate more broadly on this topic: the impact of using flow charts should result from comparing the outcomes for two randomly assigned versions of the same tasklet, where the program is presented as flow chart vs. textual code.

4. Data collection and analysis of the results

The test-survey has been administered to 225 students from 16 high schools disseminated in the considered geographical area. 76% of the students followed a technical program, 15% a scientific one and 9% other types of curricula. Their age range was 15–18; more specifically 18% were attending the 2nd year, 54% the 3rd and 28% the 4th year (over 5 years of upper secondary instruction). All the students have engaged in the test in controlled situations, either during classwork (166) or at the beginning of a summer workshop (59). The data provided through the survey have been collected and processed in anonymous form.[1]

Of the investigation questions introduced in Section 3, Q4 has an explicit focus on students' perception of self-confidence, whereas Q1–3 may be addressed both in terms of performance and self-confidence. As mentioned earlier, here we will take the latter perspective and consider students' performance only in order to answer Q4. In the following analysis, we will refer to the tasklets (available online — see section 3) by labels in the format T$S.K$, where $S = 1|2$ denotes the taskset and $K = 1 \div 6$ the K-th tasklet in that taskset, possibly followed by the specific question, either (i) or (ii), when two multiple choice questions are asked.

Q1 – Self-confidence for different code features. The bar diagram reported in Figure 2 shows the distribution of students' perceived levels of self-confidence on their answers concerning abstract properties of the programs occurring in the tasklets. In summary, we can make the following observations:

- Both tasklets requiring to deal with boolean variables (T1.3 and T2.1) have been perceived as especially challenging.
- Most students feel not self-confident with string-processing code (T1.6, T2.3, T2.5); the only possible exception is tasklet T2.6, but only about 20% of the subjects is fully confident on their achievement and this is in fact the tasklet that registered the worst performance.

[1] Since no personal information was shared with any third party, and neither the students nor their institutions could be identified through the presented data, the research policies of our country do not require the approval by an ethics commission.

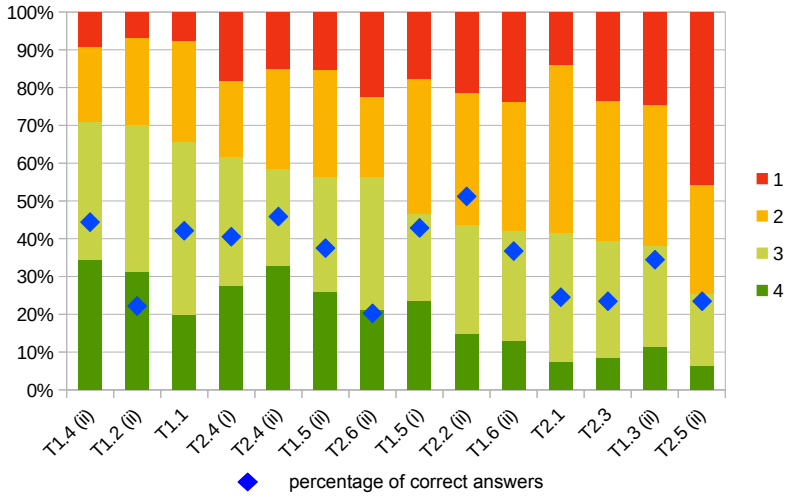

Figure 2: Students' perception of self-confidence for each question addressing higher-order thinking skills — sorted by decreasing "positive" level of self-confidence (Likert levels 3 and 4).

- Programs that carry out purely numerical (hence mathematical) computations, on the other hand, are perceived as easier to understand (T1.1, T1.2, T1.4); an exception in this respect is tasklet T2.2, which however requires to master the equivalence between different iteration constructs (`for`, `while` and `do-while`).
- Other code features, including nested constructs, seem to be perceived as more or less troublesome, depending on the context in which they occur.

For the sake of exemplification, the question shown in Figure 1 (T1.4) received the highest level of overall self-confidence. At the other extreme, of lowest self-confidence, the tasklet T2.5 includes a reversibility question for a string-processing program based on a plain loop.

It may be worth noting that most figures of our analysis do not distinguish between heterogeneous subgroups of students since we intend to convey a general idea about their perceived self-confidence, independently of specific contexts. However, the average overall level of self-confidence turned out to be essentially the same for technical (2.56) and scientific schools (2.55), and just a little lower for other types of schools (2.42). The differences appear more pronounced, on the other hand, over subsequent years of instruction, indicating an increase from the 2nd (2.20) to the 3rd year (2.70) and, unexpectedly, a decrease from the 3rd to the 4th year (2.46).

Q2 – Self-confidence in connection with tracing. We found no evidence that previous effort to trace code execution can increase the perceived level of self-confidence while analysing a program at a more abstract level. In particular, we may contrast the self-confidence bars in Figure 2 for T1.3(ii) and T2.4(ii). Both questions ask to identify a program's functional purpose: in the former case after tracing the program to count the iterations for a given input; in the latter after answering a more abstract reversibility question. The impact of tracing on program comprehension should however be investigated more accurately by means of a specifically designed instrument.

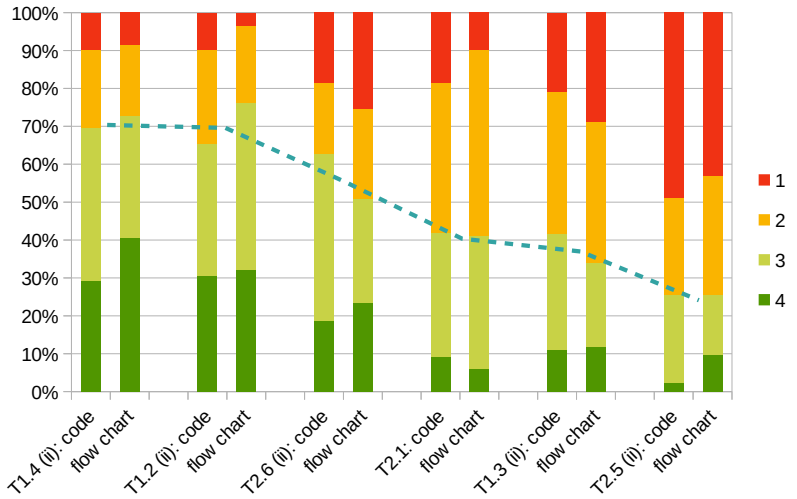

Figure 3: Students' self-confidence while analysing flow charts vs. textual code for all the tasklets whose programs could be presented in both forms; dashed green line: average percentages of "positive" levels of self-confidence (3 + 4).

Q3 – Impact of flow charts on self-confidence. As clearly illustrated by the bar diagram in Figure 3, students' self-confidence does not appear to be affected by reasoning on flow charts rather than on textual code — and in fact their performance does not either. Also in terms of average levels of self-confidence when using flow charts vs. textual code, the outcomes are hardly distinguishable: 2.76 vs. 2.69 for the first taskset and 2.26 vs. 2.25 for the second one.

Q4 – Self-confidence vs. performance. The most significant insight of the investigation, from a pedagogical perspective, is that students' perception of self-confidence on the provided solution is very poorly correlated with their actual performance in the task. To devise a quantitative measure, the options of the multiple-choice questions have been subdivided into three groups: *correct* answers, *incorrect* answers and *severely incorrect* answers.[2]

To begin with, after collecting the data the instrument turned out to be very well calibrated for the sake of the intended analysis, since it resulted into a balanced tripartition of the answers: 35% correct, 30% incorrect and 35% severely incorrect (more specifically, 37%, 29% and 34% for taskset 1; 33%, 31% and 36% for taskset 2).

Then, in order to establish a reasonable correspondence with the Likert range 1–4, these three groups have been assigned weights 4, 2 and 1, respectively. On this basis, Pearson's correlation coefficient between self-confidence and performance is only 0.235 for taskset 1 and 0.208 for taskset 2 (less than, for example, the correlation measured in similar tests in university courses [25, 26]). The correlation is low for most tasklets, the only exception being the 0.598 level for T2.4(ii), and it is not statistically significant for five items of the second taskset ($p > 0.05$); moreover, for three items of the first taskset $0.02 < p < 0.05$.

Alternatively, things can also be seen from a complementary standpoint by looking at individual subjects: in this respect the correlation between self-confidence and performance

[2]Refer to the online appendix to see how the options provided for each task have been classified in these terms.

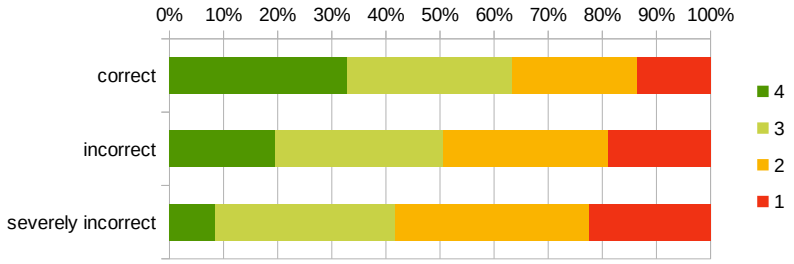

Figure 4: Visualisation of the poor correlation between self-confidence on the proposed solution and actual achievement in the tasklets.

is *negative* for 35% of the students. The general situation is probably better illustrated by the diagram in Figure 4, where we can see that, on the whole, about half of the incorrect answers as well as more than 40% of the severely incorrect answers are nevertheless associated with a positive perception of self-confidence on the provided solution. Conversely, more than 1/3 of the correct answers are paired with a negative perception of self-confidence.

In particular, remarkably high rates of positive self-confidence (Likert levels 3 and 4) for incorrect and severely incorrect answers have been registered for tasklet T1.2 (72%), for the equivalence question T1.4(ii) reported in Figure 1 (63%), for tasklet T1.1 (62%) and for T2.6(ii) (57%). Just to give a rough idea of the scope of these tasks, T1.2 asks a reversibility question about a `while` loop with a composite condition and a nested `if-else`. In T1.1 the student is required to identify the appropriate condition of a plain loop carrying out a simple numerical computation. Finally, T2.6 asks again to assess equivalence between simple `do-while` and `while` constructs used in string-processing programs.

5. Conclusions

In this paper we have attempted to analyse high school students' perception of self-confidence while engaging in small program comprehension tasks. In Section 3 we have outlined the main motivations underlying the choice of the tasks and the overall structure of the used instrument. In Section 4 we have then summarised a range of findings concerning high school students' perception of self-confidence. This will hopefully help to build a more detailed picture about the understanding of iteration in the high school context.

In a nutshell, the major results of our analysis can be stated as follows. First, students seem to be more at ease with mathematical computations than with string-processing tasks (what may be partly a consequence of the examples customarily presented in class), but boolean variables are probably troublesome to them. Second, being required to trace code execution does not appear to have a significant impact on the understanding of more abstract program properties. Third, our results do not support previous findings [39] that flow-charts may not be natural for students: the levels of self-confidence when using flow charts or textual code are essentially the same. The most salient outcome is, however, that students' perception of self-confidence is very poorly correlated to their actual performance in a task, what is likely to indicate weakness of metacognitive skills.

In light of the last observation as well as of the role of metacognitive skills in effective learning [40], the main implications of our study pertain to devising pedagogical strategies to strengthen learners' awareness in this respect. According to Schraw [21], indeed, metacognitive awareness *can* be taught. Thus, a more ambitious project could aim to elicit the *reasons why* students feel more or less confident about the provided solutions, from their own perspective, and to explore the impact of teaching metacognitive skills explicitly — in particular, if the type of tasks used in our present study could also be exploited for these purposes.

Currently, the test-survey has been administered to more than 200 students, but we are trying to broaden the scope of this empirical investigation, if possible by involving other interested educators working in different contexts. It would be especially interesting to compare the perception of self-confidence of girls versus boys. Indeed, besides noting that — unsurprisingly — their average level of self-confidence is slightly lower, we are unable to draw statistically significant insight from our present sample since the girls are only about 8% (19). It may also be worth elaborating on and extending the tasksets outlined here for instructional purposes. Indeed, we think that teachers could use them both as examples to illustrate different aspects connected to iteration and as instruments to assess students' understanding of this topic.

References

[1] E. W. Dijkstra, On the cruelty of really teaching computing science, Communications Of The ACM 32 (1989) 1398–1404.

[2] B. Du Boulay, Some difficulties of learning to program, Journal of Educational Computing Research 2 (1986) 57–73.

[3] A. Robins, J. Rountree, N. Rountree, Learning and teaching programming: A review and discussion, Computer Science Education 13 (2003) 137–172.

[4] A. Luxton-Reilly, Simon, et al., Introductory programming: A systematic literature review, in: Proc. Companion of the 23rd Annual ACM Conf. on Innovation and Technology in Computer Science Education, ITiCSE 2018, ACM, New York, USA, 2018, pp. 55–106.

[5] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, G. L. Herman, Identifying student misconceptions of programming, in: Proc. of the 41st ACM Technical Symp. on Computer Science Education, SIGCSE '10, ACM, New York, 2010, pp. 107–111.

[6] Y. Cherenkova, D. Zingaro, A. Petersen, Identifying challenging cs1 concepts in a large problem dataset, in: Proc. of the 45th ACM Tech. Symp. on Computer Science Education, SIGCSE '14, ACM, New York, NY, USA, 2014, pp. 695–700.

[7] C. Izu, C. Schulte, et al., Fostering program comprehension in novice programmers: Learning activities and learning trajectories, in: Proc. of the ITiCSE Working Group Reports, ITiCSE-WGR '19, ACM, New York, NY, USA, 2019, pp. 27–52.

[8] J. D. Bransford, A. L. Brown, R. R. Cocking (Eds.), How People Learn: Brain, Mind, Experience, and School – Expanded Ed., National Academy Press, Washington, D.C., 2000.

[9] K. Falkner, R. Vivian, N. J. Falkner, Identifying computer science self-regulated learning strategies, in: Proc. of the 2014 Conference on Innovation & Technology in Computer Science Education, ITiCSE '14, ACM, New York, NY, USA, 2014, pp. 291–296.

[10] G. T. L. Brown, L. R. Harris, Student self-assessment, in: SAGE Handbook of Research on Classroom Assessment, SAGE Publications, Inc., Thousand Oaks; Thousand Oaks, California, 2013, pp. 367–393.
[11] E. Scapin, C. Mirolo, An exploration of teachers' perspective about the learning of iteration-control constructs, in: S. N. Pozdniakov, V. Dagienė (Eds.), Proc. of ISSEP 2019 – Informatics in Schools: New Ideas in School Informatics, Springer, Cham, 2019, pp. 15–27.
[12] E. Scapin, C. Mirolo, An exploratory study of students' mastery of iteration in the high school, in: Local Proc. of ISSEP 2020, CEUR Workshop Proceedings, 2020, pp. 43–54.
[13] E. Soloway, J. Bonar, K. Ehrlich, Cognitive strategies and looping constructs: An empirical study, Commun. ACM 26 (1983) 853–860.
[14] E. Rahimi, E. Barendsen, I. Henze, Identifying students' misconceptions on basic algorithmic concepts through flowchart analysis, in: V. Dagienė, A. Hellas (Eds.), Proc. of ISSEP 2017 – Informatics in Schools: Focus on Learning Programming, Springer International Publishing, Cham, 2017, pp. 155–168.
[15] I. Cetin, et al., Teaching loops concept through visualization construction, Informatics in Education - An International Journal 19 (2020) 589–609.
[16] D. Perkins, S. Schwartz, R. Simmons, Instructional strategies for the problems of novice programmers, in: R. E. Mayer (Ed.), Teaching and Learning Computer Programming, Routledge, New York, USA, 1988, pp. 153–178.
[17] T. Jenkins, On the difficulty of learning to program, in: Proceedings of the 3rd annual LTSN ICS Conference, 2002, pp. 53–58.
[18] J. Sorva, Notional machines and introductory programming education, Trans. Comput. Educ. 13 (2013) 8:1–8:31.
[19] R. Lister, B. Simon, E. Thompson, J. L. Whalley, C. Prasad, Not seeing the forest for the trees: Novice programmers and the solo taxonomy, in: Proc. of the 11th Annual SIGCSE Conf. on Innovation and Technology in Computer Science Education, ITICSE '06, ACM, New York, USA, 2006, pp. 118–122.
[20] M. Mani, Q. Mazumder, Incorporating metacognition into learning, in: Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, ACM, New York, USA, 2013, pp. 53–58.
[21] G. Schraw, Promoting general metacognitive awareness, Instructional science 26 (1998) 113–125.
[22] P. Steinhorst, A. Petersen, J. Vahrenhold, Revisiting self-efficacy in introductory programming, in: Proceedings of the 2020 ACM Conference on International Computing Education Research, 2020, pp. 158–169.
[23] D. Loksa, L. Margulieux, B. A. Becker, M. Craig, P. Denny, R. Pettit, J. Prather, Metacognition and self-regulation in programming education: Theories and exemplars of use, ACM Trans. Comput. Educ. To appear (2022).
[24] L. Murphy, J. Tenenberg, Do computer science students know what they know? a calibration study of data structure knowledge, in: Proc. of the 10th Annual SIGCSE Conf. on Innovation and Technology in Computer Science Education, ITiCSE '05, ACM, New York, USA, 2005, pp. 148–152.
[25] P. Denny, A. Luxton-Reilly, J. Hamer, D. B. Dahlstrom, H. C. Purchase, Self-predicted and actual performance in an introductory programming course, in: Proc. of the 15th Annual

Conf. on Innovation and Technology in Computer Science Education, ITiCSE '10, ACM, New York, USA, 2010, pp. 118–122.
[26] P. Lee, S. N. Liao, Targeting metacognition by incorporating student-reported confidence estimates on self-assessment quizzes, in: Proc. of the 52nd ACM Technical Symp. on Computer Science Education, SIGCSE '21, ACM, New York, USA, 2021, pp. 431–437.
[27] C. Izu, C. Mirolo, Comparing small programs for equivalence: A code comprehension task for novice programmers, in: Proc. of the 2020 ACM Conf. on Innovation and Technology in Computer Science Education, ITiCSE '20, ACM, New York, USA, 2020, pp. 466–472.
[28] C. Mirolo, C. Izu, E. Scapin, High-school students' mastery of basic flow-control constructs through the lens of reversibility, in: Proc. of the 15th Workshop on Primary and Secondary Computing Education, WiPSCE '20, ACM, New York, USA, 2020, pp. 1–10.
[29] P. Sutherland, The application of piagetian and neo-piagetian ideas to further and higher education, International J. of Lifelong Education 18 (1999) 286–294.
[30] D. Ginat, On novice loop boundaries and range conceptions, Computer Science Education 14 (2004) 165–181.
[31] I. Cetin, Student's understanding of loops and nested loops in computer programming: An APOS theory perspective, Canadian Journal of Science, Mathematics and Technology Education 15 (2015) 155–170.
[32] M. Mladenovic, I. Boljat, Ž. Žanko, Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level, Education and Information Technologies 23 (2018) 1483–1500.
[33] A. Kumar, G. Dancik, A tutor for counter-controlled loop concepts and its evaluation, in: 33rd Annual Frontiers in Education - FIE '03, volume 1, 2003, pp. T3C–7.
[34] R. Lister, E. Adams, S. Fitzgerald, et al., A multi-national study of reading and tracing skills in novice programmers, in: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, ITiCSE-WGR '04, ACM, New York, USA, 2004, pp. 119–150.
[35] M. Lopez, J. Whalley, P. Robbins, R. Lister, Relationships between reading, tracing and writing skills in introductory programming, in: Proc. 4th Int. Workshop on Comput. Educ. Research, ICER '08, ACM, New York, USA, 2008, pp. 101–112.
[36] L. Rigby, P. Denny, A. Luxton-Reilly, A miss is as good as a mile: Off-by-one errors and arrays in an introductory programming course, in: Proc. of the 22nd Australasian Computing Education Conf., ACM, New York, USA, 2020, pp. 31–38.
[37] C. S. Miller, A. Settle, Mixing and matching loop strategies: By value or by index?, in: Proc. of the 52nd ACM Technical Symp. on Computer Science Education, SIGCSE '21, ACM, New York, 2021, pp. 1048–1054.
[38] R. Smetsers-Weeda, S. Smetsers, Problem solving and algorithmic development with flowcharts, in: Proc. of the 12th Workshop on Primary and Secondary Computing Education, WiPSCE '17, ACM, New York, USA, 2017, pp. 25–34.
[39] H. R. Ramsey, M. E. Atwood, J. R. Van Doren, Flowcharts versus program design languages: An experimental comparison, Commun. ACM 26 (1983) 445–449.
[40] S. Bergin, R. Reilly, D. Traynor, Examining the role of self-regulated learning on introductory programming performance, in: Proc. of the 1st Int. Workshop on Comput. Educat. Research, ICER '05, ACM, New York, USA, 2005, pp. 81–86.