An Exploratory Study of Students' Mastery of Iteration in the High School

Emanuele Scapin and Claudio Mirolo

University of Udine, 33100 Udine, Italy
scapin.emanuele@spes.uniud.it claudio.mirolo@uniud.it

Abstract. Although a number of studies report about novices' difficulties with basic flow-control constructs, concerning both the understanding of the underlying notional machine and the logical connections with the application domain, this issues have not yet been extensively explored in the context of high-school education. As part of a project whose long-run goal is identifying methodological tools to improve the learning of iteration, we analyzed how a sample of 164 high-school students' approached three small programming tasks involving basic looping constructs, as well as two questions on their subjective perception of difficulty. If, on the one hand, most students seem to have developed a viable mental model of the basic workings of the underlying machine, on the other, dealing at a more abstract level with loop conditions and nested flow-control structures appears to be challenging. As to the implications for teachers, the results of the analysis suggest that more efforts should be addressed to develop a method for testing the conjectures about program behavior, as well as to the treatment of loop conditions in connection with the problem statement.

Keywords: Informatics education · Programming learning · High school · Iteration constructs · Novice programmers

1 Introduction

Students' difficulties to learn programming are well known to computer science educators, e.g. [6,25,19]. The reasons may be manifold, ranging from lack of problem solving skills to the need for accuracy and intensive practice. Programming is indeed problem-solving intensive: according to Gomes and Mendes [10] it requires "not a single, but a set of skills", and students may fail to develop a viable model of the underlying *notional machine* [27] or to be able to connect code execution with its functional purpose [16]. Part of the difficulties may also be related to the habits and expectations of both teachers and learners [13,24].

Significant problems and misconceptions are reported even for such basic flow-control constructs as conditionals and loops. Kaczmarczyk et al. [14], for instance, identified "a number of misconceptions all related to an inability to properly understand the process of while loop functioning", and Cherenkova

et al. [4] found that "students have significant trouble with conditionals and loops, with loops being particularly challenging". Although most related studies focus on tertiary CS1 courses, it is conceivable to expect that similar issues are especially significant at the high-school (i.e. upper secondary) level of instruction.

On this basis, we engaged on a project to investigate the teaching and learning of *iteration*, in the high school, as well as to identify methodological tools to improve code comprehension. The first steps of this project, discussed in [26], explored and contrasted teachers' vs. students' perceptions of the major difficulties related to programming, in general, and more specifically to iteration.

Now, the objective of the present analysis is to gain some preliminary insights into high-school students' mastery of basic iteration constructs. Here by mastery we mean *conceptual* mastery of code structures, focusing on program *comprehension* rather than construction [12]. We used as instrument a survey including three tiny problems, referred to as *tasklets* here, to address each of the learning dimensions introduced in [22], namely the understanding of the computation model underlying iteration, the ability to establish relations between the components of a loop and the statement of a problem, the ability to interpret the program structures based on iteration. In addition, in light of the presumed role of metacognitive skills in effective learning [3], we asked students two questions about their subjective perceptions of difficulty.

We can re-state the goals of our exploratory investigation as research questions as follows: To what extent are students proficient with the programming learning dimensions mentioned above? And do their subjective perceptions of difficulty correlate with their actual performance? Although it would have been more insightful to cover a larger and more varied set of programming tasks, we decided to assign only three small tasklets in order to limit the risk that teenage students lose their concentration and provide scarcely meaningful answers.

Having set the general objectives and background of this work, the rest of the paper is organized as follows. Section 2 presents the tasklets and the questions asked to students. Section 3 summarizes the results of the analysis. Finally, in Section 4 we discuss the findings and outline some future perspectives.

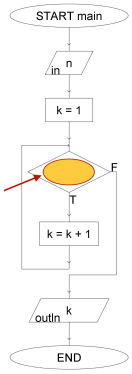
2 Tasklets and questions

2.1 Tasklet 1: identifying the correct loop condition

Tasklet 1 was aimed to explore the ability to draw connections between a simple loop condition and the statement of a problem by reasoning on a flow chart.

Problem statement: *The algorithm represented by the flow chart in Figure 1 (left) computes the number of bits of the binary representation of a positive integer n, i.e. the smallest exponent k such that* 2^k *is greater than n. Choose the appropriate condition among the four listed below.*

The four available options were: $2^k = n$, $2^k \leq n$, $2^k < n$ and $2^k > n$. To achieve this — supposedly easy — task, students were expected to read carefully the statement above and, for each of the listed conditions, figure out




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

while (x > 1 && y > 1 && x != y) {
    if(x < y)
        y = y - x;
    else
        x = x - y;
}

if (x == 1 || y == 1)
    printf("m=%d and n=%d are co-prime\n", m,n);
else
    printf("m=%d and n=%d are not co-prime\n", m,n);
```

Fig. 1: Flow chart of *tasklet 1* and code of *tasklet 2*.

the relationship between k and n after exiting the loop. The specific focus of this tasklet is suggested by the frequency of condition-related issues, see e.g. [4].

2.2 Tasklet 2: ascertaining the number of iterations

Tasklet 2 addressed students' mastery of the "mechanics" of the execution of a loop controlled by a non-trivial condition and including a nested `if`.

Problem statement: *The program shown in Figure 1 (right) checks if two positive integer m and n are co-prime. If the input values are m=15 and n=44, how many times the* `while` *loop will repeat?*

The above question could be answered by choosing among six options, namely: 0, 1, 2, 3, 4 *or more*, and *the loop never ends*. As we can see in Figure 1, the loop is characterized by a composite condition (using two `and`s) and a nested `if-else`. In order to identify the right option students were essentially required to trace the code execution carefully. Thus, this tasklet addresses tracing skills, which have been in the scope of several investigations, e.g. [18].

2.3 Tasklet 3: recognizing functionally equivalent programs

Tasklet 3, the most challenging one, asked to recognize equivalence between different programs in order to investigate the ability to grasp comprehensively nested combinations of conditionals and iteration constructs.

Problem statement: *Consider the five programs in Figure 2 and assume that the input values of m and n are always positive integers. Two such programs are equivalent if they compute and print the same output whenever they are run for the same input data. Identify the equivalent programs in Figure 2.*

To approach this problem on functional equivalence, students had to reason at a more abstract level. Each program involves nested constructs whose behavior must be grasped and dealt with comprehensively. The last tasklet is similar in structure as well as in spirit to that discussed in [11].

```c
// 1
x = m;
y = n;

while ( x != y ) {

  while ( x < y )
    x = x + m;
  while ( x > y )
    y = y + n;
}

printf("result: %d\n", x);
```

```c
// 2
x = m;
y = n;

while ( x != y ) {

  if ( x > y )
    y = y + m;
  else
    x = x + n;
}

printf("result: %d\n", x);
```

```c
// 3
x = m;
y = n;

while ( x != y ) {

  while ( x < y || x > y ) {
    x = x + m;
    y = y + n;
  }
}

printf("result: %d\n", x);
```

```c
// 4
x = m;
y = n;

while ( x != y ) {

  if ( x < y )
    x = x + m;
  else
    y = y + n;
}

printf("result: %d\n", x);
```

```c
// 5
x = m;
y = n;

while ( x != y ) {

  if ( x < y )
    x = x + x;
  else
    y = y + y;
}

printf("result: %d\n", x);
```

Fig. 2: The five programs to be compared in tasklet 3.

2.4 Subjective perception questions

Besides engaging in the three tasklets above, the students were asked two short questions about their subjective perception of difficulties. The first one was a multiple choice question: "*What do you find most difficult when you use loops*?" The five available options reported the difficulties that emerged as most significant from the teachers' interviews [26]:

i. To find the condition of a `while` or `do-while` loop;
ii. To define a complex condition including logical operators (AND, OR, NOT);
iii. To deal with nested loops;
iv. To understand, in general, when the loop should end;
v. To deal with the loop control variable.

The second question: "*What kind of mistakes affected your performance most significantly*?" was instead open, so the students could choose to indicate any source of error, either conceptual or of a different nature.

3 Data collection and results

The (anonymous) survey was administered to 164 students, most of whom attending the second or third year (age 15–17) of scientific and technical high

Table 1: Rates of chosen options for tasklet 1.

Condition	**Percentage**	
$2^k = n$	3.7%	
$2^k \leq n$	38.4%	*correct option*
$2^k < n$	20.1%	
$2^k > n$	37.8%	

schools, i.e. when the basic flow-control constructs are introduced. As we have seen in the section 2, all three tasklets are numerical in nature, but this choice is due to the fact that so are most of the examples students are exposed to in class. Anyway, they are just based on simple arithmetic, familiar to students.

We now present the main results of our investigation relative to the three tasklets and the two questions included in the survey. Overall, only about 8% of the students solved all the three problems correctly, 27% provided two correct answers, 39% one correct answer, and 26% were wrong on all tasklets.

3.1 Tasklet 1: identifying the correct loop condition

Table 1 summarizes the results concerning tasklet 1. A little less than 40% of the students provided the correct answer, namely $2^k \leq n$, whereas about as many selected one of the two seriously wrong options, either $2^k = n$ or $2^k > n$. Although the flow chart is rather simple, consisting of a very standard loop structure, and the problem specification is accurate, it turns out that students can easily be misled about the role or the interpretation of the loop condition.

3.2 Tasklet 2: ascertaining the number of iterations

As shown in Table 2, about 60% of the students chose the right option for tasklet 2, i.e. three iterations. It hence appears that a large majority of them is at ease with the functioning of iteration combined with a nested conditional, as well as with the interpretation of a composite (loop) condition including logical operators. It is conceivable that they identified the right option by tracing the code execution — what they probably did not try to do, on the other hand, to check their answer for tasklet 1.

3.3 Tasklet 3: recognizing functionally equivalent programs

The rates of recurrent answers relative to tasklet 3 are listed in Table 3. It was clearly the hardest challenge and, as we can see, less than one fifth of the students were able to recognize that program 1 and program 4 are the equivalent ones. In addition, most of the answers grouped in the last row of Table 3 are meaningless in that only one program was selected (about 30% of the whole sample), conceivably indicating that they just decided to skip this tasklet.

Table 2: Rates of chosen options for tasklet 2.

Number of iterations	**Percentage**	
0	3.7%	
1	9.1%	
2	15.2%	
3	60.4%	*correct option*
4 or more	6.1%	
the loop never ends	5.5%	

Table 3: Rates of recurrent answers for tasklet 3.

Equivalent programs	**Percentage**	
Programs 1 and 4	18.9%	*correct answer*
Programs 4 and 5	13.4%	
Programs 2 and 4	11.0%	
Programs 1 and 3	7.3%	
Programs 1, 4, 5	3.7%	
Meaningless or isolated answers	45.7%	

While such pairings as 1–3 or 4–5 (see Fig. 2) are likely to signal serious misconceptions, it is worth noting that regarding program 2 and program 4 as equivalent may be more simply ascribable to carelessness, i.e. not paying attention to the fact that the roles of `x` and `y` are swapped, but those of `m` and `n` are not. Here again, however, the frequency of incorrect answers indicates that students are not used to test their conjectures by tracing code execution.

3.4 Subjective perception questions

The pie chart in Fig. 3 summarizes students' answers to the question: "*What do you find most difficult when you use loops?*" As we can see, nested loops are the source of issues reported most frequently (41.5%), followed by the definition of composite conditions (23.2%), the latter possibly due to insufficient familiarity with Boolean logic. Then the rates of the other options are, in decreasing order: figuring out a suitable loop condition (15.20%), understanding when an iteration should end (12.8%), and dealing with loop control variables (7.3%).

What emerges by comparing the subjective perception of difficulty (when dealing with iteration) to the actual performance in the three tasklets is that students may underestimate their lack of mastery of loop conditions. If, on the one hand, more than 60% of them failed to choose the right option for tasklet 1 (in fact a straightforward condition), on the other only about 15% indicated the implied feature, i.e. identifying the loop condition, as a major source of difficulty.

Although the rate of choice of this feature is slightly higher (almost 18%) among those students who provided an incorrect answer for tasklet 1, there

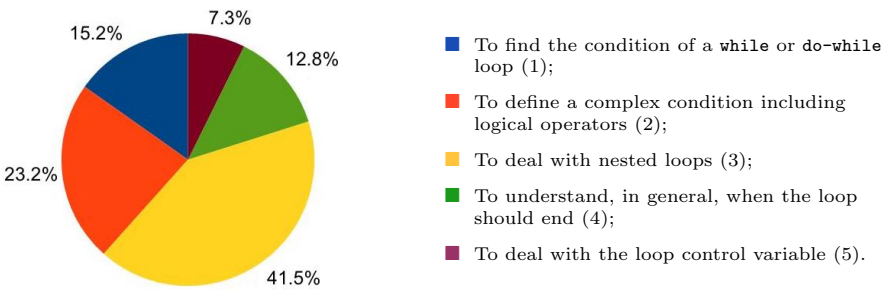


Fig. 3: Major difficulty with iteration in students' perception.

Table 4: Contingency table: correct/incorrect answers to tasklet 1 vs. perceived prominence or not of difficulties with loop conditions.

	difficulties with loop conditions	**other difficulties**
incorrect answer to tasklet 1	18	83
correct answer to tasklet 1	7	56

appear to be no statistically significant correlation between correct/incorrect answers to this tasklet and perceiving or not the related feature as a major source of difficulty: by cross-tabulating the corresponding counts, see Table 4, and subjecting them to a χ^2-test we get a *p-value* of about 0.35, meaning that the data are fairly consistent with the assumption of independence of the two variables (*null* hypothesis). In addition, the limited awareness of difficulties with loop conditions is also signaled by the observation that just one out of the ten students who failed only on tasklet 1 seems to give prominence to the problem. More in general, as can be elicited "pictorially" from the partitioned bars in Figure 4, we cannot find any statistical evidence of correlation between poor performance in subsets of the tasklets and subjective perception of difficulty with specific concepts.

To conclude the summary of the results of interest here, we consider the answers to the second question about the mistakes having had more severe implications in the students' subjective perception. An inductive analysis [20] of keywords occurring in the *open* answers gave rise to the categories summarized in Table 5. From the data in the right column it appears that of the 37.6% of students who identify some specific concept as a major source of mistakes, more than one fourth mention precisely iteration, so confirming they are aware of the relevance of this topic to their learning of programming. Other reported concepts, with similar or lower frequency, refer to procedural and object abstractions, language syntax, mathematical and logical prerequisites.

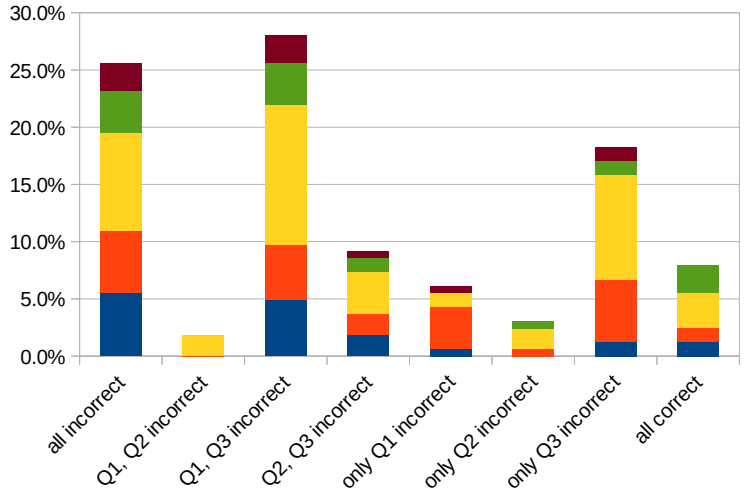


Fig. 4: Distribution of students' perception of difficulty vs. performance; the interpretation of the colors is the same as in Figure 3.

4 Discussion

Based on the data we collected, as can be seen from the bars in Fig. 4, almost two thirds of the students achieved successfully no more than one of the three proposed tasklets. Mastery of iteration, even in relatively simple programs, is then to be considered a cognitively demanding learning objective for the considered age range. The results outlined in the previous section appear to corroborate, in the high school context, the findings of previous work addressing students' difficulties with conditionals and loops, e.g. [4,14]. In particular, it turns out that dealing with nested flow-control structures and, perhaps to a minor extent, with loop conditions are especially challenging to novices, the former aspect also in their subjective perception.

Here is a summary of our interpretation of the findings.

i. When (essentially) required to trace the code execution, as in *tasklet 2*, a large majority of the students are able to determine the correct outcome, see Table 2. It is also conceivable that a number of the about 20% who opted for '2' or '4 *or more*' iterations made only minor computing mistakes. We can then presume that most high school students develop a viable and accurate enough mental model of the *notional machine* underlying code execution, including the functioning of nested constructs and the evaluation of relatively complex conditions.
ii. It is worth observing that 77% of the students who were correct in *tasklet 2* provided *seriously* wrong answers to *tasklet 1* or *tasklet 3*. Apparently, then, the students tend to not exploit their tracing abilities in order to test their conjectures about program behavior. This observation could be explained

Table 5: Major sources of mistakes in students' perception.

Sources of mistakes	**Percentage**
iteration, loops	10.4%
functions, subroutines	10.4%
syntax, instructions	7.3%
mathematics and logic	4.9%
objects, classes, methods	3.7%
general causes such as poor understanding of text, lack of time, insufficient practice, distraction	41.4%
elusive answers	11.6%
no answer provided	10.4%

either by some general lazy attitude or, what is more relevant from a pedagogical perspective, by lack of method to approach programming tasks.

iii. As shown in Table 1, more than 40% of the students chose seriously incorrect options in *tasklet 1* (first and last option). A similar performance shows that, as a matter of fact, a large part of them are unable to master the relationships between loop condition and accurate specification in the application domain, even in a straightforward situation. This may possibly be ascribed to confusion about the role of the loop condition, meant as an 'exit' condition instead of a 'continue' condition, or to some more basic lack of problem-solving skills.

iv. Overall, the students seem to underestimate their difficulties to deal with loop conditions — even simple conditions. On the one hand, they did not feel the need to check their solution to tasklet 1 by tracing the program execution for sample inputs (what could have been done very quickly). On the other hand, only a low percentage of those who made serious errors in *tasklet 1* and/or *tasklet 2* perceive their weakness in this respect as a major difficulty (see the chart in Figure 4).

v. By comparing students' performance in tasklets 2 and 3, it appears that their difficulties with nested constructs are not so much about the mechanics of code execution as about the ability of grasping code behavior at a more abstract level. So, the crucial point is how to develop students' abstraction skills, besides the understanding of the mechanical features of code (to illustrate which there are several widespread tools — see for example the paragraphs on program visualization in [19]).

Implications for instructors

A few provisional implications for the instructional practice can be drawn from the points raised above. In particular, we point out three potential insights, which are worth further, more accurate investigation. By referring to a *competency framework* for computing education [7], the first two pertain to the *skills* area and the third one to the *dispositions* area:

– Firstly, more efforts are to be addressed to the development of a method to approach programming tasks, in particular to identify suitable test cases in order to confirm or refute working conjectures.
– Secondly, more careful attention should be paid to the role and treatment of loop conditions, especially in connection with a problem's statement.
– Finally, at the meta level, students' attitude to think critically about their learning should be enhanced, for example by asking them to make explicit their degree of self-confidence in the achievement of a task or of a part of it.

Learning to program is however a slow and gradual process, as argued by Dijkstra in [5], and therefore the teacher must grant adequate learning time to be spent on several effective examples.

Future work and perspectives

To begin with, in order to validate (or refute) our provisional interpretation of the findings discussed above, our next step will be to design a more comprehensive survey, to be administered to a larger sample of students, from as wide an area of the country as possible.

We are also trying to envisage appropriate methodological approaches to the teaching and learning of iteration. In this respect, we think that it would be helpful to collect a rich and varied set of examples, not limited to the stereotypical code patterns mentioned in the teachers' interviews [26]. In particular, such examples should address "interesting" problems involving more complex loop conditions or (nested) combinations of flow-control structures.

As to the development of students' abstraction skills to interpret program behavior, a possible line of research could be based on de Raadt's and colleagues approach to explicitly teaching (and assessing) programming strategies [23], which are relevant to a comprehensive understanding of nested constructs. Another potential source of inspiration in this respect may be the instructional work that elaborates on the concept of *loop invariant* [28,1,9,8], suitably adapted to fit less formal learning styles [2].

As a further middle/long term objective, from a more general pedagogical standpoint, it may be interesting to explore the implications of the *productive failure* perspective [15,17] in a computing education context, especially in connection with the learner's self-confidence on the solution provided [21].

5 Conclusions

As part of a project aimed at identifying methodological tools to enhance a comprehensive understanding of iteration, in this paper we have analyzed the answers of a sample of 164 high school students to three small programming tasks and two questions on their perception of difficulty. The results appear to confirm that dealing with iteration, even in simple programs, is cognitively demanding to students of the considered age range.

Apparently, the problems faced by most of the students should not be ascribed to a flawed model of the notional machine. Rather, lack of carefulness and accuracy, i.e. of a method, while dealing with the program constructs may be at the root of several mistakes. In particular, interpreting loop conditions and abstracting on nested flow-control structures turned out to be major challenges to novices. To extend the scope of this exploratory analysis, we are now planning to design a survey to collect more data on students' performance in small programming tasks, as a basis to develop methodological tools to enhance students' mastery of iteration and to support their learning.

References

1. Arnow, D.: Teaching programming to liberal arts students: Using loop invariants. In: Proceedings of the 25th SIGCSE Symposium on Computer Science Education. pp. 141–144. SIGCSE '94, ACM, New York, NY, USA (1994)
2. Astrachan, O.: Pictures as invariants. In: Proceedings of the 22nd SIGCSE Technical Symposium on Computer Science Education. pp. 112–118. SIGCSE '91, ACM, New York, NY, USA (1991)
3. Bergin, S., Reilly, R., Traynor, D.: Examining the role of self-regulated learning on introductory programming performance. In: Proceedings of the 1st International Workshop on Computing Education Research. pp. 81–86. ICER '05, ACM, New York, NY, USA (2005)
4. Cherenkova, Y., Zingaro, D., Petersen, A.: Identifying challenging cs1 concepts in a large problem dataset. In: Proc. of the 45th ACM Tech. Symp. on Computer Science Education. pp. 695–700. SIGCSE '14, ACM, New York, NY, USA (2014)
5. Dijkstra, E.W.: On the cruelty of really teaching computing science. Communications Of The Acm **32**(12), 1398–1404 (1989)
6. Du Boulay, B.: Some difficulties of learning to program. Journal of Educational Computing Research **2**, 57–73 (01 1986)
7. Frezza, S., Daniels, M., Pears, A., Cajander, r., Kann, V., Kapoor, A., McDermott, R., Peters, A.K., Sabin, M., Wallace, C.: Modelling competencies for computing education beyond 2020: A research based approach to defining competencies in the computing disciplines. In: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. pp. 148–174. ITiCSE 2018 Companion, ACM, New York, NY, USA (2018)
8. Furia, C.A., Meyer, B., Velder, S.: Loop invariants: Analysis, classification, and examples. ACM Computing Surveys (CSUR) **46**(3), 1–51 (2014)
9. Ginat, D.: Seeking or skipping regularities? novice tendencies and the role of invariants. Informatics in Education **2**, 211–222 (2003)
10. Gomes, A., Mendes, A.: Learning to program - difficulties and solutions. In: International Conference on Engineering Education – ICEE. pp. 283–287 (01 2007)
11. Izu, C., Mirolo, C.: Comparing small programs for equivalence: A code comprehension task for novice programmers. In: Proc. of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. pp. 466–472. ITiCSE '20, ACM, New York, NY, USA (2020)
12. Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., Mirolo, C., et al.: Fostering program comprehension in novice programmers - learning activities and learning trajectories. In: Proc. of

the Working Group Reports on Innovation and Technology in Computer Science Education. pp. 27–52. ITiCSE-WGR '19, ACM, New York, NY, USA (2019)
13. Jenkins, T.: On the difficulty of learning to program. In: Proceedings of the 3rd annual LTSN ICS Conference (2002)
14. Kaczmarczyk, L.C., Petrick, E.R., East, J.P., Herman, G.L.: Identifying student misconceptions of programming. In: Proceedings of the 41st ACM Technical Symposium on Computer Science Education. pp. 107–111. SIGCSE '10, ACM, New York, NY, USA (2010)
15. Kapur, M.: Examining productive failure, productive success, unproductive failure, and unproductive success in learning. Educational Psychologist **51**, 1–11 (04 2016)
16. Lister, R., Simon, B., Thompson, E., Whalley, J.L., Prasad, C.: Not seeing the forest for the trees: Novice programmers and the solo taxonomy. In: Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. pp. 118–122. ITICSE '06, ACM, New York, NY, USA (2006)
17. Loibl, K., Roll, I., Rummel, N.: Towards a theory of when and how problem solving followed by instruction supports learning. Educational Psychology Review **29**(4), 693–715 (Dec 2017)
18. Lopez, M., Whalley, J., Robbins, P., Lister, R.: Relationships between reading, tracing and writing skills in introductory programming. In: Proc. 4th Int. Workshop on Comput. Educ. Research. pp. 101–112. ICER '08, ACM, New York, USA (2008)
19. Luxton-Reilly, A., Simon, Albluwi, I., Becker, B.A., Giannakos, M., Kumar, A.N., Ott, L., Paterson, J., Scott, M.J., Sheard, J., Szabo, C.: Introductory programming: A systematic literature review. In: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. pp. 55–106. ITiCSE 2018 Companion, ACM, New York, NY, USA (2018)
20. Mayring, P.: Qualitative Content Analysis: Theoretical Foundation, Basic Procedures and Software Solution. Klagenfurt (2014)
21. Metcalfe, J.: Learning from errors. Annual Review of Psychology **68**, 465–489 (Jan 2017)
22. Mirolo, C.: Is iteration really easier to learn than recursion for cs1 students? In: Proc. of the 9th Annual International Conference on International Computing Education Research. pp. 99–104. ICER '12, ACM, New York, NY, USA (2012)
23. de Raadt, M., Watson, R., Toleman, M.: Teaching and assessing programming strategies explicitly. In: Proceedings of the 11th Australasian Conference on Computing Education - Volume 95. pp. 45–54. ACE '09, Australian Computer Society, Inc., Darlinghurst, Australia (2009)
24. Robins, A., Haden, P., Garner, S.: Problem distributions in a cs1 course. In: Proc. of the 8th Australasian Conference on Computing Education - Volume 52. pp. 165–173. ACE '06, Australian Computer Society, Inc., Darlinghurst, Australia (2006)
25. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: A review and discussion. Computer Science Education **13**(2), 137–172 (2003)
26. Scapin, E., Mirolo, C.: An exploration of teachers' perspective about the learning of iteration-control constructs. In: Pozdniakov, S.N., Dagienė, V. (eds.) Informatics in Schools. New Ideas in School Informatics. pp. 15–27. Springer, Cham (2019)
27. Sorva, J.: Notional machines and introductory programming education. Trans. Comput. Educ. **13**(2), 8:1–8:31 (2013)
28. Tam, W.C.: Teaching loop invariants to beginners by examples. In: Proceedings of the 23rd SIGCSE Technical Symposium on Computer Science Education. pp. 92–96. SIGCSE '92, ACM, New York, NY, USA (1992)