Design and development of an instrument to investigate high-school students' understanding of iteration

Emanuele Scapin[0000−0001−8384−8231] and Claudio Mirolo[0000−0002−1462−8304]

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

1 Introduction

Loops and conditionals turn out to be potential sources of novices' misconceptions, e.g. [7,2]. In this respect, our purpose is to outline here the design and development of an instrument, composed of a set of small tasks, or *tasklets*, to investigate in more depth on high-school students' understanding of iteration in terms of *code reading* abilities, according to the perspective taken in [6]. In particular, we will try to summarise the motivations underlying the choice of the tasks and the overall structure of the instrument. A major aim of this contribution is indeed to invite the interested educators and researchers to take part in the project in order to broaden the scope of the empirical study.

Beyond the suggestions drawn from the literature, the instrument is based on the insights drawn from two previous studies: (i) A number of interviews asking high-school teachers about the role of iteration in their practice and their perception of students' difficulties [14]; (ii) A survey addressed to students, including questions on their subjective perception of difficulties as well as three tiny tasks involving basic iteration constructs [15]. In summary, we attempt to address the following main questions:

Q1. To what extent are students at ease with a range of "technical" features implied by iteration? (E.g., structure and role of loop conditions, loop-control variables, nesting of flow-control constructs, looping over arrays.)
Q2. Does the effort to trace code facilitate thinking of the overall computation at a higher abstraction level?
Q3. Are students' answers more accurate when using flow-chart or textual code representations of programs?
Q4. To what extent are students self-confident about their comprehension of a program's overall computation and purpose?

Starting from the pioneering work on the cognitive implications of programming tasks in the early 1980s, e.g. [17], empirical research has persistently reported that flow-control constructs such as conditionals and loops tend to be approached in stereotypical ways and are common sources of misconceptions for novice learners [7,2]. According to Gomes and Mendes [4] programming is indeed problem-solving intensive, in fact it requires "not a single, but a set of

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 (of 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)

skills", 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 [8]. Overall, the results surveyed in the literature seem to leave room for a more systematic analysis of a range of aspects connected to the understanding of iteration in the high school.

2 Characterisation of the instrument

The chosen tasklets cover the topics outlined in Table 1, where each area is labeled with the related investigation questions. While the technical program features (Q1) are intrinsic to any given tasklet, the correlation between tracing effort and abstraction (Q2) is addressed by asking two subsequent questions in the same task. The role of flow charts (Q3) is expected to result from comparing the outcomes for two randomly assigned versions of the same task, where the program is presented as flow chart vs. code. Finally, the indication of the subjective perception of self-confidence (Q4), in a 4-grade Likert scale, is required for each question concerning high-level thinking skills.

We have devised two tasksets, both consisting of 6 tasklets with a balanced distribution over the areas and topics of Table 1, arranged in such a way that students can be expected to complete the test in about one hour. Each tasklet asks one or two multiple-choice questions. In addition, the programs of three tasklets may be presented either as code or as flow-charts. The survey has then been made accessible online, the four resulting versions (2 tasksets $\times$ code/flow-chart modes) being assigned randomly. An English translation is available via the link: `http://nid.dimi.uniud.it/additional_material/iteration_survey.html`

To accommodate for the habits of most Italian high schools, the textual code is Java-like. We will now elaborate a little on the main themes listed in Table 1.

A. Tasklets addressing higher-order thinking skills. This area covers two broad 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 use *equivalence* [5] and *reversibility* [19] tasks. The tasklets in the latter include more widespread types of questions which ask either to choose an appropriate item to complete a program with a given purpose or to identify the purpose of a given program [8].

B. Tasklets addressing code features. The tasklets pertaining to this area are based on the data collected in our pilot work [14,15], identifying loop conditions and nested constructs as major sources of difficulties, and on some related findings, e.g. [3,1].

C. Tasklets addressing code execution. Small problems that can be solved at low levels of abstraction by tracing the code execution are quite common, but such ability cannot be taken for granted, since many students struggle with tracing loops, especially `while`-loops [9]. Our main purpose, however, is to address the investigation question Q2, namely, whether code tracing can to some extent support higher-order thinking *in the task at hand*. In [15] 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. Thus, the idea is to assign similar tasks pertaining to the "abstract" area A, either including or not a first question that can be answered via tracing.

D. Tasklets addressing data types. The covered data types are essentially *numbers*, *booleans*, *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 [13,10].

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). We included this feature since our survey [15] suggested that students' subjective perception of difficulty and actual performance in a task do not always correlate.

FC. Flow-chart versus code. According to [16], "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 [11]. Quite surprisingly, however, the preliminary data collected with our instrument appear to confirm Ramsey's et al. findings that the use of flow-charts is far from natural for students [12], so we want to investigate more broadly on that.

3 Work in progress...

Currently, the outlined survey has been administered to more than 200 students, but we are planning to broaden the sample. In particular, a major ambition would be to involve other educators in order to extend the scope of this empirical research to different contexts.

References

1. Cetin, I.: Student's understanding of loops and nested loops in computer programming: An APOS theory perspective. Canadian Journal of Science, Mathematics and Technology Education **15**(2), 155–170 (2015)
2. Cherenkova, Y., Zingaro, D., Petersen, A.: Identifying challenging cs1 concepts in a large problem dataset. In: Proc. of the 45th SIGCSE. pp. 695–700 (2014)
3. Ginat, D.: On novice loop boundaries and range conceptions. Computer Science Education **14**, 165–181 (2004)
4. Gomes, A., Mendes, A.: Learning to program - difficulties and solutions. In: International Conference on Engineering Education – ICEE. pp. 283–287 (2007)
5. Izu, C., Mirolo, C.: Comparing small programs for equivalence: A code comprehension task for novice programmers. In: Proc. of ITiCSE '20. pp. 466–472 (2020)
6. 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 ITiCSE-WGR '19. pp. 27–52 (2019)
7. Kaczmarczyk, L., Petrick, E., East, P., Herman, G.: Identifying student misconceptions of programming. In: Proc. of the 41st SIGCSE. pp. 107–111 (2010)
8. 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 ITICSE '06. pp. 118–122 (2006)
9. Lopez, M., Whalley, J., Robbins, P., Lister, R.: Relationships between reading, tracing and writing skills in introductory programming. In: Proc. of ICER '08. pp. 101–112 (2008)
10. Miller, C., Settle, A.: Mixing and matching loop strategies: By value or by index? In: Proc. of the 52nd SIGCSE. pp. 1048–1054. SIGCSE '21 (2021)
11. Rahimi, E., Barendsen, E., Henze, I.: Identifying students' misconceptions on basic algorithmic concepts through flowchart analysis. In: Dagienė, V., Hellas, A. (eds.) Proc. of ISSEP 2017. pp. 155–168. Springer, Cham (2017)
12. Ramsey, H.R., Atwood, M.E., Van Doren, J.R.: Flowcharts versus program design languages: An experimental comparison. Commun. ACM **26**(6), 445–449 (1983)
13. Rigby, L., Denny, P., Luxton-Reilly, A.: 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 Conference. pp. 31–38 (2020)
14. Scapin, E., Mirolo, C.: An exploration of teachers' perspective about the learning of iteration-control constructs. In: Pozdniakov, S.N., Dagienė, V. (eds.) Proc of ISSEP 2019. pp. 15–27. Springer, Cham (2019)
15. Scapin, E., Mirolo, C.: An exploratory study of students' mastery of iteration in the high school. In: Local Proc. of ISSEP 2020. pp. 43–54. CEUR Workshop Proceedings (2020), `http://ceur-ws.org/Vol-2755/`
16. Smetsers-Weeda, R., Smetsers, S.: Problem solving and algorithmic development with flowcharts. In: Proceedings of the 12th Workshop on Primary and Secondary Computing Education. pp. 25–34. WiPSCE '17, ACM, New York, NY, USA (2017)
17. Soloway, E., Bonar, J., Ehrlich, K.: Cognitive strategies and looping constructs: An empirical study. Commun. ACM **26**(11), 853–860 (1983)
18. Sorva, J.: Notional machines and introductory programming education. Trans. Comput. Educ. **13**(2), 8:1–8:31 (2013)
19. Teague, D., Lister, R.: Programming: Reading, writing and reversing. In: Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education. pp. 285–290. ITiCSE '14, ACM, New York, USA (2014)