Identifying Students' Misconceptions on Basic Algorithmic Concepts Through Flowchart Analysis

Ebrahim Rahimi[1], Erik Barendsen[2], and Ineke Henze[3]

[1] Radboud University, The Netherlands, e.rahimi@cs.ru.nl
[2] Radboud University and Open University, The Netherlands, e.barendsen@cs.ru.nl
[3] Delft University of Technology, The Netherlands, f.a.henze-rietveld@tudelft.nl

Abstract. In this paper, a flowchart-based approach to identifying secondary school students' misconceptions (in a broad sense) on basic algorithm concepts is introduced. This approach uses student-generated flowcharts as the units of analysis and examines them against plan composition and construct-based programming problems to identify students' misconceptions. In this study, 102 flowcharts, generated by 50 students in two informatics classes in the Netherlands, were examined and various sorts of misconceptions were identified. The results suggest that, given their abstract and language-independent nature, flowcharts can be considered as an effective tool for revealing students' difficulties in understanding algorithmic concepts. Our approach contrasts the more traditional use of program code to investigate students' misconceptions. We found several misconceptions mentioned in the literature, together with two misconceptions which appear not to have been described before. Our research contributes to the usage of flowcharts as a formative assessment tool, directing informatics teachers' instruction toward resolving these misconceptions.

1 Introduction

There is an ongoing global debate among educational policy makers, researchers, technology developers and practitioners around establishing computer science (CS) and in particular computer programming as an integral part of school curricula at different levels [6,1]. The advocates argue that due to the highly technology-driven and information-dependent nature of working and living in the 21st century, learning fundamental computer science concepts and programming has become a must for everyone [6]. They argue that CS represents an essential and unprecedented sort of literacy people need to survive in the knowledge era where thinking critically and computationally to solve complex, ill-defined, and open-ended problems is becoming more and more important [21]. This sort of literacy is meant to help people to not simply become technology consumers, but also to serve as technology creators and knowledge developers [6,10].

Programming generally includes two main tasks: algorithmic problem solving and coding. The problem solving part includes analyzing the problem, formulating a solution, and devising an algorithm for implementing the solution. The coding part consists of implementing the devised algorithm using a programming language, debugging and modification [15]. Algorithmic thinking thus plays an essential role in problem solving and programming and accordingly has a key position in computational thinking notions

[21]. Despite the important role of problem solving and algorithmic thinking in programming, in general, these aspects seem to have received less attention than coding within CS in secondary education [18].

Flowcharts are advocated as useful pictorial representations of algorithms, the program logic or its flow of control. They have been an integral element of programming since the introduction of computers in 1940s [15]. Flowcharts have been used as simple and effective tools to comprehend, devise, modify, visualize, debug, express and communicate algorithms and programs [12]. As remarked by [15], a flowchart represents "a high level definition of the solution to be implemented on a machine" (p. 1). Given their abstract and language-independent nature, flowcharts have been suggested as effective tools for the novice programmers to enhance their problem solving and algorithmic thinking competencies, and to promote a "thinking first" approach to programming [18]. As held by [2], *flowcharting* helps "distinguish between the procedure a computer program is written to express and the syntactical details of the language in which the program is written" (p. 53).

The purpose of this study is to investigate the usability of flowcharts as a tool for identifying and highlighting students' misconceptions on basic algorithmic concepts. Following [20], we will use the term *misconception* as an overall term covering students' lack of understanding, problems, mistakes, bugs, and difficulties with basic algorithm concepts. Flowcharts can be seen as a way to get into the students' minds and provide a better picture of their understanding of basic computer science concepts. By doing so, flowcharts ultimately serve to improve the quality of computer science education through a better understanding of what goes wrong [20]. In [18] an analytical approach to identifying programming misconceptions using flowcharts is introduced. This approach uses two categories of plan composition and construct-based problems introduced by [20] to detect students' misconceptions. It was used to detect programming misconceptions exhibited by 11 high school students in the Netherlands.

Our study adopted the approach introduced in [18] and modified it to detect students' misconceptions on each of basic algorithmic concepts including sequence, condition, iteration, and sub-algorithm. To this end, 102 flowcharts generated by 50 students in two informatics classes in the context of upper secondary education in the Netherlands were examined.

This study is a part of a bigger research project with the main objective of encouraging and facilitating the learning of computer science concepts through conducting *authentic design assignments* (see [9]). This study plays a multifold role in this research project: (*i*) highlighting the importance of student-generated flowcharts as intermediate design products, (*ii*) examining the usability of flowcharts as an effective formative assessment tool, (*iii*) helping the teachers and researchers in identifying students' misconceptions on basic algorithm concepts, and (*iv*) directing teachers instruction toward resolving these misconceptions. This way, applying flowcharts as a formative assessment tool might help to further develop the teachers' PCK (Pedagogical Content Knowledge) [16] on algorithms.

2 Programming Misconceptions

There are worldwide observations that students, regardless of institution or country, show generally poor performance in the introductory programming courses and learning programming is a difficult task for novices [3,17]. A reason for this difficulty stems from the fact that programming puts a high cognitive and knowledge demand on novices including knowledge on a specific programming language and knowledge and understanding of basic programming concepts and constructs such as variables, loops, conditions, abstraction, and procedures. Any misconception on these constructs might result in programming difficulties [17,18]. Therefore, realizing and resolving these misconceptions seems useful in diminishing the novices difficulties in programming.

Students' misconceptions on basic programming constructs and features have been vastly researched (see [4,8,11,14,17,19,20]). Four common examples of the identified programming misconceptions include considering classes as containers for objects, reverse assignment, interpreting assignment statements as mathematical equations, and boundary problem (i.e, choosing inappropriate boundary points) [17,20]. Seppälä et al. in [13] categorized students' errors on algorithm exercises into systematic and careless errors ('slips'). According to [13], slips result from randomly trying out algorithm exercises by students, whereas a systematic error is "a symptom of a misconception that could be corrected if recognized" (p. 244). According to [20], students' misconceptions on programming fall in two categories: *plan composition* and *construct-based* problems. Plan composition problems refer to the encountered misconceptions in composing a solution by putting the pieces of plans together. These misconceptions are concerned with the solution formulation, algorithm development, and planning the semantic and logic of the program [3]. On the other hand, construct-based problems concern with the misconceptions on language constructs and the syntax of the program. Tables 1 and 2 summarize plan composition and construct-base problems introduced by [18,20].

However, the majority of these misconceptions have been researched and recognized in the context of program's code in specific programming languages. There are very few studies examining the programming misconceptions at a more abstract level such as algorithms and flowcharts (see [18]). Arguably, the sorts of misconceptions exhibited by students in a programming language depend, to some extent, on the specifications of that programming language and might be shaped or filtered by its structures and specifications. In contrast, flowcharts provide a more abstract and language-independent way to get into the minds of students and seem useful in gaining a firsthand and more comprehensive picture of what goes wrong in their minds.

3 Study Setting

The participants in this study were 50 students of age 15 or 16 from two schools in upper secondary education in the Netherlands. All of the participants were at the university preparatory education level (VWO in Dutch) [7]. In the aforementioned surrounding research project (see the introduction section), the participants were provided with the background and design documents. The background document covered various aspects of algorithms including the definition, basic concepts (i.e. sequence, condition,

Table 1. Plan composition problems, adopted from [20]

Misconception	**Description**
Summarization problem	The complex combinations of plans are summarized in terms of some primary functions and secondary functions have been overlooked.
Optimization problem	Novices aim to optimize their plans but do not adequately check if the optimization can really be carried out.
Previous-experience (or pollution) problem	Novices constantly develop and tailor plans on the basis of previous experience. This error is introduced when inappropriate aspects of previous plans pollute a related plan that is being used in a new situation.
Specialization problem	Inappropriate and incorrect customization of an abstract plan developed for other situations.
Natural-language problem	Errors happen during the process of mapping from natural language to a programming language.
Boundary problem	Novices difficulties for deciding on appropriate boundary points in specializing a plan.
Unexpected case problem	The program is not working correctly for all cases (e.g. uncommon, unlikely, or boundary cases).
Interpretation problem	Not considering the implicit specifications of plans or misinterpreting them.
Cognitive load problem	Omitting and overlooking small but important parts of the plan or plan interactions.

Table 2. Construct-based problems, adopted from [18,20]

Misconception	**Description**
Human interpretation problem	Novices assumption that computers are able to interpret problems as people do.
Assignment misconception	Inverted assignment: the positions of the giver and receiver variables at the right and left side of the assignment operator are misplaced.
Condition misconception	Misconceptions about how the condition construct (or control structure) works.
Boolean statements	Misconceptions on how boolean statements behave.
Loops	Misconception on how a loop control variable works.
Method-related misconceptions	Misconceptions on method calling.
Control flow	Misconceptions such as incorrect use of print and return statements, omitting arrows in a flowchart, or omitting start or stop steps.

iteration, and sub-algorithms), trace table, and flowcharts. Each of these aspects was explained using several examples and questions with a different level of difficulty. The design document included seven authentic design assignments in the context of text analysis. In each assignment, students were asked to follow a step-wised approach to learning algorithms and programming. These steps include analyzing the design problem, formulating a conceptual solution, devising an algorithm, developing a flowchart to implement the algorithm, testing the flowchart using trace tables, and converting their flowcharts into a program using a programming language (either PHP or Python, as the students had learned one of these languages in the previous years). Students could use the background document whenever necessary for explanation and advice on conducting these steps. To embed formative assessment within this educational intervention, students were asked to answer four exit questions during their design endeavours on a weekly basis. These questions were derived from the background document and were meant to assess students' understanding of basic algorithm concepts underpinning the design assignments. In each question, the students were asked to draw a flowchart for solving the given problem. To foster collaboration and team working, the students were grouped into teams of 2-3 students and each team developed a flowchart for each of the questions. The flowcharts were written on paper or developed digitally using a specific tool called Draw.io and were handed over manually or via the school's learning management system. The teachers then could use these flowcharts to detect students' misconceptions and provide them with appropriate feedback. Table 3 presents these questions and their associated algorithm concepts.

Table 3. Questions asked to investigate students' understanding of basic algorithmic concepts

Question	**Concepts**
Q1: Write an algorithm that receives three numbers (a, b, c) and determines and reports the maximum and minimum numbers.	Sequence, condition
Q2: (*i*) Write two sub-algorithms *findmax (a,b)* and *findmin (a,b)* which receive two numbers a, b and determine and return the maximum and minimum numbers, respectively. (*ii*) Using these sub-algorithms write an algorithm that receives 10 numbers (i.e., a_0, a_1, ... , a_9) and determines and reports the maximum and minimum numbers.	Sub-algorithms, problem composition and decomposition, condition
Q3: Write an algorithm that receives a text value (i.e. t) as input and reverses it (i.e. t = "book", reverse of t = "koob").	Loop, condition

The following research question directed the data collection and analysis processes: *What misconceptions can be seen in students' flowcharts?*

The flowcharts developed by the participating students in response to the questions in Table 3 were used as the data source to answer this research question. A qualitative deductive content analysis approach was followed to analyze the flowcharts. Content analysis is a method used for analyzing written, verbal or visual communication messages and documents [5]. Following the process of deductive content analysis introduced by [5], three steps of preparation, organization, and reporting were taken. In the preparation step, all of the flowcharts (102 in total) as the unit of analysis were uploaded

into Atlas.ti software. In the organizing phase, first the plan composition and construct-based problems presented in tables 1 and 2 were chosen as the initial categories for coding the observed misconceptions in the flowcharts. Then following an iterative process of coding and revising, the misconceptions on basic algorithm concepts observed in the flowcharts were coded according to these categories. The coding process was flexible to allow emerging new codes which did not exist in these categories. Possible alternative interpretations of the observed misconceptions were discussed within the research team until a consensus was reached. Finally, the identified misconceptions on each of basic algorithm concepts were processed and reported as can be seen in the next section.

4 Results

Tables 4 and 5 present the identified *plan composition* and *construct-based* problems in the flowcharts, respectively.

Table 4. The plan composition problems observed in the flowcharts

Category	**Misconceptions**
Condition misconceptions	*Not considering equal inputs in plans (unexpected case problem)*: the situations where the equal numbers are excluded from the comparison operations.
	Incorrect adaptation (previous experience problem): refers to incorrect applying of a previous condition-related plan in another situation.
	Misinterpreting the condition concept (Interpretation problem)
Loop misconceptions	*Incorrect loop construction*
Sub-algorithm misconceptions	*Composition problem*: refers to students problems and mistakes with composing an algorithm by putting together sub-algorithms.
Translation	*Mapping problem*: incorrect mapping from natural language or programming languages to flowcharts.

• Plan composition problems

(i) *Condition misconceptions*: Three plan composition misconceptions for the *condition concept* were identified. *Not considering equal inputs in plans* is an *unexpected case problem* where the equal numbers are excluded from a solution, specifically for answering questions 1 and 2 in table 3. This was one of the most frequently observed misconceptions in the flowcharts. Interestingly, some of the students reacted to passing equal numbers to their flowcharts as a fatal error (see fig. 1 (a)). *Incorrect adaptation* is the next condition misconception and represents a *previous experience problem*. Fig 1 (b) illustrates an example of this misconception where an incorrect adaptation of the plan used for determining the maximum number (i.e. *max*) between a, b is used for calculating the minimum number (i.e. *min*). *Misinterpreting the logic of condition concept* is a sort of *interpretation problem* with regard to the condition concept, as depicted by fig 1 (c).

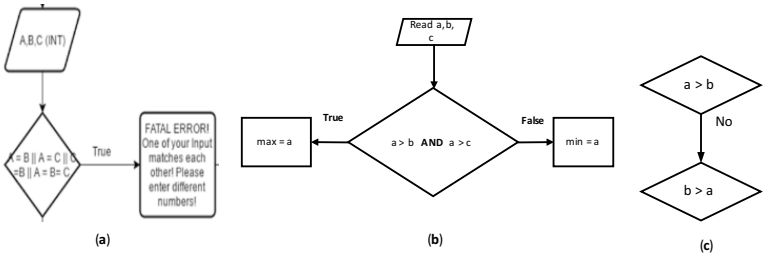


Fig. 1. Examples of identified plan composition problems related to the condition concept

(ii) *Iteration misconceptions*: *Incorrect loop construction* is the encountered misconception related to the implementation of the iteration concept. Fig 2 depicts an example of this misconception.

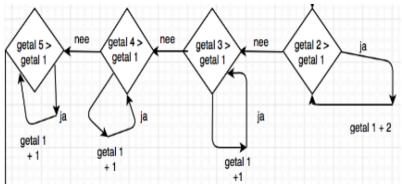


Fig. 2. An example of incorrect loop implementations

(iii) *sub-algorithm misconceptions*: The *composition problem* refers to students problems, difficulties, and mistakes with composing an algorithm using its sub-algorithms. The results showed that while many of the students devised the asked sub-algorithms (i.e. *findmax* and *findmin*) correctly, many of them were not able to use the developed sub-algorithms to compose the main algorithm.
(iv) *Translation misconceptions*: The *mapping problem* or incorrect translation from formal language or programming languages to flowcharts was another plan composition problem observed in the flowcharts. Fig 3 illustrates an example of this misconception. Surprisingly, as shown by this example, although some of the students already know how to program, but they cannot map their program into a flowchart.

• Construct-based Misconceptions

(i) *Condition misconceptions*: *Missing the false part* represents a misconception where only the *true* output of a condition statement is addressed. Fig 4 (a) presents an example of this misconception. The *3-output condition* is another misconception where 3 possible outputs are considered for a conditional statement, as shown in fig 4 (b).

(ii) *Sequence misconceptions*: The sequence misconceptions were among the frequent problems observed in the flowcharts. These misconceptions reflect students' misunderstanding about the flowchart's flow control and sequential execution of its steps.

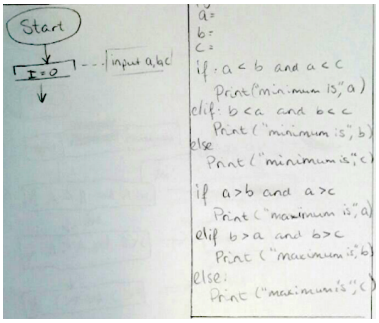
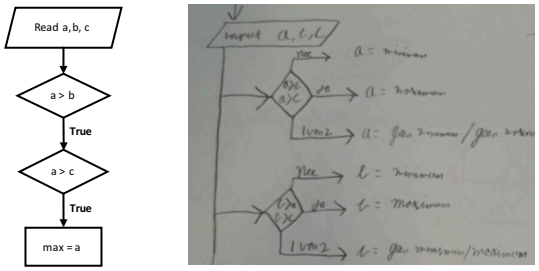


Fig. 3. An example of difficulty in mapping a plan from programming languages to a flowcharts



(a) Missing the False part (b) 3-output conditional statements

Fig. 4. Example of construct-based misconceptions for the condition concept

The *parallel execution misconception* reflects a perception held by some of the students that the flowchart's steps can run in parallel to each other, as shown by fig 5 (a). Another observed misconception with the sequence concept is called the *dense step* which refers to a flowchart's step that executes more than one operations. Fig 5 (b) shows two examples of this misconception. It seems that in these examples the students assumed that the flowcharts interpret these steps like what people do. The last sequence misconception concerns with students' *issues with Input/output and start/stop steps*. These issues refer to situations where students forgot to get input, report or return the final results, forgot to add start and stop steps in their flowcharts, or used return instead of report.

(iii) *Loop misconceptions*: Fig 6 illustrates an example of the identified construct-based loop misconception. This misconception is concerned with the *simultaneous initialization and check of the loop's control variable*. As can be realized from this example, this misconception might lead to either infinite or zero-repeating loops, depending on the value of *len(word1) - 1*.

(iv) *Assignment misconceptions*: Two construct-based misconceptions related to the assignment operation were discerned in the flowcharts. The most occurring error in the students flowcharts was the *inverted assignment* where the positions of the source and destination variables in the right and left of the assignment operator were misplaced. Fig. 7 (a) shows an example of this misconception. In this example the students were

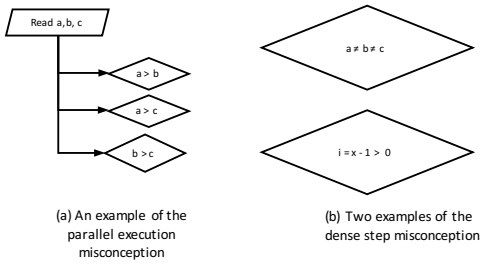


Fig. 5. Examples of construct-based misconceptions for the sequence concept

Table 5. The construct-based problems observed in the flowcharts

Category	**Misconceptions**
Condition misconceptions	*Missing the False part*: conditional statements where the false part (or the else part) is missing.
	3-output condition: conditional statements with 3 outputs.
Sequence misconceptions	*Dense step (Human interpreter problem)*: situations where more than one operations are processed in one step.
	Parallel execution of steps (flow control)
	Issues with Input/output and start/stop steps (flow control)
Loop misconceptions	*Simultaneous initializing and checking of the loop control variable*
Assignment misconceptions	*Inverted assignment*: situations where the positions of the source and destination statements of the assignment operator are misplaced.
	3-output condition: conditional statements with 3 outputs.
Sub-algorithm misconceptions	*Incorrect use of sub-algorithms*: for instance using a sub-algorithm at the left side of an assignment operator.
	Issues of calling/returning from sub-algorithms
Flowchart presentation	*Using incorrect shapes for the flowchart's constructs*

asked to find the maximum and minimum numbers among variables *a*, *b*, *c* and store them in variables *max* and *min*, respectively. Another type of the observed assignment issues is the *missing value* misconception where a value is processes but not stored in a variable. Fig. 7 (b) shows an example of this misconception.

(v) *Sub-algorithm misconceptions*: The *Incorrect use of sub-algorithms* exhibits an misunderstanding and wrong interpretation of the usage of sub-algorithms. For instance, it was observed that some of the students interpreted a sub-algorithm as a variable or container and used it at the left side of an assignment operator. *Issues of calling/returning from sub-algorithms* represent another sub-algorithm misconception. Examples of this misconception include: not passing input parameters to sub-algorithms and not returning results from sub-algorithms.

(vi) *Flowchart presentation*: This misconception refers to situations where incorrect shapes were used by the students to present various constructs of their flowcharts. For example, several students used rectangle instead of diamond for presenting conditional statements.

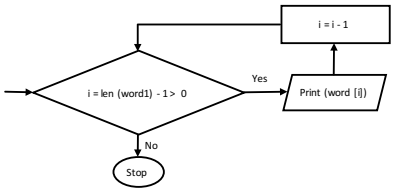
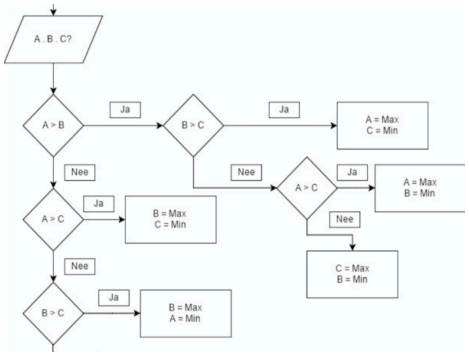
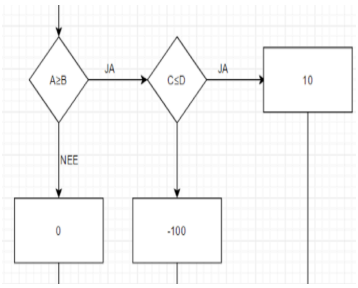


Fig. 6. An example of the loop misconception



(a) An example of the inverted assignment misconception



(b) An example of the missing value misconception

Fig. 7. Two examples of the assignment misconceptions

5 Conclusions and Discussion

In this study, a flowchart-based analysis approach, adapted from [18,20], was introduced and followed to identify the misconceptions of basic algorithm concepts exhibited by 50 students in upper secondary education in the Netherlands. Various types of plan composition and construct-based misconceptions on basic algorithm concepts have been identified in the analyzed flowcharts. The identified plan composition misconceptions are *not including equal numbers in plans* (an unexpected case misconception), *incorrect adaptation* (a previous experience misconception), *misinterpreting the condition concept* (an interpretation misconception), *incorrect loop construction*, *composition problem* (i.e. devising an algorithm using sub-algorithms), and *mapping problem*. The discerned construct-based misconceptions include *missing the false part* of conditional statements, *3-output condition*, *parallel execution of steps* (a flow control misconception), *dense step* (a human interpreter misconception), *missing input/output and start/stop steps*, *incorrect initialization and checking of the loop counter*, *inverted assignment*, *missing values*, *incorrect use of sub-algorithms*, *issues of calling/returning from sub-algorithms*, and *using incorrect shapes for the flowchart's statements*.

Computer programming consists of two interconnected contexts of high level, abstracted and language-independent algorithmic thinking, and detailed, step-wised, and language-dependent coding. Our findings suggest that some of the programming mis-

conceptions at the abstract level might be filtered by specifications of programming languages and accordingly not be visible for language-based misconceptions detecting approaches. On the basis of our findings, it can be concluded the flowcharts, by inheriting and bridging between the essence and key characteristics of algorithmic thinking and coding contexts, are useful means to detect misconceptions exhibited by students in the solution formulation and algorithmic thinking phases.

As observed in the results section, the students exhibited various sort of plan composition and construct-based problems on basic algorithm concepts in their flowcharts. The inverted assignment was the most occurring misconception encountered by the students. The same finding has been reported by other researchers (see for example [8,17]). Furthermore, the students showed several misconceptions related to the sub-algorithm concept including the composition of an algorithm using sub-algorithms. Arguably, a reason for this misconception stems from this fact that the majority of the students did not have any previous experience of working with abstract constructs such as sub-algorithms. Surprisingly, it has been observed that some of the students first wrote computer programs and then tried to convert their programs to flowcharts.

Given the programming misconceptions presented in tables 1 and 2 as the analytical framework, the results suggest that there are many overlaps between misconceptions encountered by students in the algorithmic and coding parts of the programming process. For example, human interpretation, previous plan experience, unexpected case, and inverted assignment misconceptions happen similarly in both parts [18]. However, there are two exceptions with regard to the *parallel execution* and *dense step* misconceptions. These misconceptions represent some of the students' assumption that flowchart's steps can run parallel to each other. To the best of our knowledge, these are newly identified misconceptions and have not been reported in other studies. A possible reason for this neglect might be due to this fact that the majority of research on programming misconceptions has been conducted in a specific programming language and the syntactical structure and specifications of the programming language filter emerging this sort of abstract misconceptions.

The use of flowcharts generated by groups of students as data, instead of individually generated flowcharts, can be seen as a limitation of the study. One might claim that group dynamics might amplify or filter misconceptions arising at an individual level. A second limitation stems from conducting the study with students who had previous experience of programming. Repeating the same study by students with less programming experience might result in detection of other kinds of misconceptions. Other possibilities for follow-up research include using the identified misconceptions to develop CS teaching materials for the algorithmic thinking and examining their effectiveness in resolving these misconceptions, and scrutinizing possible links between students' exhibited misconceptions in algorithmic thinking and coding phases of programming. As to the bigger research project on design-based education, our results suggest that using flowcharts to express intermediate design products could provide a more *authentic* way to incorporate formative assessment of fundamental concepts during students' project work.

References

1. Barendsen, E., Grgurina, N., Tolboom, J.: A new informatics curriculum for secondary education in the netherlands. In: International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, Springer (2016) 105–117
2. Bohl, M.: Flowcharting techniques. Science Research Associates (1971)
3. Chetty, J., van der Westhuizen, D.: Towards a pedagogical design for teaching novice programmers: design-based research as an empirical determinant for success. In: Proceedings of the 15th Koli Calling Conference on Computing Education Research, ACM (2015) 5–12
4. Clancy, M.: Misconceptions and attitudes that interfere with learning to program. Computer science education research (2004) 85–100
5. Elo, S., Kyngäs, H.: The qualitative content analysis process. Journal of advanced nursing **62**(1) (2008) 107–115
6. Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., Settle, A.: Computational thinking in k-9 education. In: Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference, ACM (2014) 1–29
7. Nuffic: The Dutch education system described (2015) Retrieved from `https://www.nuffic.nl/en/publications/find-a-publication/education-system-the-netherlands.pdf`, September 2017.
8. Putnam, R.T., Sleeman, D., Baxter, J.A., Kuspa, L.K.: A summary of misconceptions of high school basic programmers. Journal of Educational Computing Research **2**(4) (1986) 459–472
9. Rahimi, E., Barendsen, E., Henze, I.: Typifying informatics teachers' pck of designing digital artefacts in dutch upper secondary education. In: International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, Springer (2016) 65–77
10. Rahimi, E., van den Berg, J., Veen, W.: Investigating teachers' perception about the educational benefits of web 2.0 personal learning environments. (2013)
11. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: A review and discussion. Computer science education **13**(2) (2003) 137–172
12. Scanlan, D.A.: Structured flowcharts outperform pseudocode: An experimental comparison. IEEE software **6**(5) (1989) 28–36
13. Seppälä, O., Malmi, L., Korhonen, A.: Observations on student misconceptions—a case study of the build–heap algorithm. Computer Science Education **16**(3) (2006) 241–255
14. Sheard, J., Simon, S., Hamilton, M., Lönnberg, J.: Analysis of research into the teaching and learning of programming. In: Proceedings of the fifth international workshop on Computing education research workshop, ACM (2009) 93–104
15. Shneiderman, B., Mayer, R., McKay, D., Heller, P.: Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM **20**(6) (1977) 373–381
16. Shulman, L.S.: Those who understand: Knowledge growth in teaching. Educational researcher **15**(2) (1986) 4–14
17. Sirkiä, T., Sorva, J.: Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In: Proceedings of the 12th Koli Calling International Conference on Computing Education Research, ACM (2012) 19–28
18. Smetsers-Weeda, R.: Think... then act() flowcharts as a tool for novice programmers to enhance their problem solving skills. Master's thesis, TU Delft, the Netherlands (2016)
19. Sorva, J.: Notional machines and introductory programming education. ACM Transactions on Computing Education **13**(2) (2013) 8
20. Spohrer, J.C., Soloway, E.: Novice mistakes: Are the folk wisdoms correct? Communications of the ACM **29**(7) (1986) 624–632
21. Wing, J.M.: Computational thinking. Communications of the ACM **49**(3) (2006) 33–35